1-1-2013

Application Of Residue Codes For Error Detection In Mixed Signal Devices

Leila Feyzmohammadi
Ryerson University

APPLICATION OF RESIDUE CODES FOR ERROR DETECTION IN MIXED SIGNAL DEVICES

by
Leila Feyzmohammadi
MEng., Ryerson University, Toronto, Canada, 2012

A project report

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2013

Author’s Declaration

I hereby declare that I am the sole author of this project report.

I authorize Ryerson University to lend this project report to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this project report by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Application of Residue Codes for Error Detection in Mixed Signal Devices

Leila Feyzmohammadi
Master of Engineering
Electrical and Computer Engineering
Ryerson University, Toronto, 2013

Abstract

Testing methods based on residue codes are considered as simple, with high probability of detecting errors. Most of the literatures on arithmetic error control codes are mainly focused on applications of secure data transmission and testing digital circuits rather than testing mixed-signal systems. In both cases implementation of residue computing circuit (RCC), also known as the residue generator is an integral part of the hardware design. In this work a low-cost compactor circuit to calculate the residue for on-line testing of analog-to-digital converter has been presented. Aliasing rate and its relationship with the resolution of the ADC have been analyzed. Theory and operation of Linear Feedback Shift Registers have been applied for the implementation of the modulo adder circuit. The compaction circuits were simulated, and the result confirmed the theoretical analysis.

TABLE OF CONTENTS

TABLE OF FIGURES

1. CHAPTER 1
INTRODUCTION

1.1 Motivation

As the size of electronic devices decrease the probability occurrence of errors increases. At the same time testing of the circuit becomes more difficult as the number of gates increases. To ensure the reliability of the circuits, it is important to detect errors as soon as they happen, identify the faulty circuit and ensure the correct functionality of the circuit in the presence of fault. Error detecting codes, more specifically residue codes has been applied to provide testability in a wide range of very large integrated circuit (VLSI) circuits [1]. Also the effectiveness of residue code checking for on-line error detection in multipliers with different architectures has been evaluated experimentally [2].

Many electronic systems, such as those for signal processing, video compression and biometrics are essentially of a mixed-signal type. A mixed-signal system (MSS) incorporates both analog and digital circuits. It is highly desirable to assure the reliability of these systems. Many methods developed for testing analog-to-digital converters (ADC) are based on the estimation of their static and dynamic metrics such as, gain, offset, SNR etc. [3]. Although these metrics carry important information about the functionality of an ADC and its subunits, but their calculation needs complex equipment for digital processing of test responses. This work develops a test scheme based on residue calculation to test the functionality of ADCs.

1.2 Scope of this Project

This work extends the application of residue codes for on-line testing of ADCs. In order to detect all possible faults in an ADC, an exhaustive set of test patterns is applied as an input to the device under test (DUT). Such a test set consists of analog input stimuli covering the full scale (FSR) of an ADC. The output responses which are also referred to as output codes should be digitally compared with the expected values. If the comparison is done after each conversion more memory, proportional to the number of input stimuli, is needed. Off-line compaction schemes can be designed based on the estimation of the arithmetic sum (signature) of the output responses of the (DUT) [4]. The actual signature is then compared against the fault-free circuit's signature.

In this work, compaction schemes based on residue codes for different modulo are analyzed, simulated, and implemented on Altera FPGA board. The aliasing rate is estimated based on theoretically achievable testing accuracy for the tolerance bounds. The limitation of the method according to [5], is that the tolerance bounds are obtained from empirical data, which requires the availability of a reference device and does not guarantee the highest accuracy for the bounds.

1.3 Report Outline

The organization of the remainder of the report is as follows. A brief overview of required background of residue codes and a number of schemes for testing arithmetic circuits, and self-checking multipliers based on residue codes is presented in Chapter 2.

In Chapter 3 data compression technique for testing is explained, and testing ADCs based on analyzing the residue is presented. Several residue calculation schemes are presented. Aliasing rate is estimated and ways to improve the rate is discussed.

In Chapter 4 the result of the simulation of the residue calculation circuit (RCC) is presented. Finally, some conclusions are summarized in Chapter 5.

1.4 Project Contribution

Finding the best compaction scheme for testing ADC without the need for complex equipment is the major concern of this project. The contribution of this project to achieve this goal is simulation, and implementation of residue calculation circuit for testing ADC. Also an estimation of aliasing for the schemes was accomplished.

2. CHAPTER 2
BACKGROUND

2.1 Introduction

In this Chapter, basic definitions of finite fields, polynomials and arithmetic operations are provided. Basics of fault modeling are introduced. Residue codes and their application in testing arithmetic circuits are discussed in details in this chapter. Schemes of cost effective residue testing circuits that has been previously developed for testing arithmetic circuits are analyzed. Proofs of the theories are omitted for briefness.

2.2 Finite fields

In this section important definitions and theories of finite fields are summarized from [6]. Proofs are omitted. All of these definitions and theories can be seen in most of algebra texts.

2.2.1 Basic Definitions and Properties

Definition 2.1 A group is a set G with a binary operation '*' on G if it satisfies these conditions:

The binary operation '*' is associative:

$\forall a, b, c \in G \quad a * (b * c) = (a * b) * c$

There exists an identity element e in G:

$\forall a \in G \quad a * e = e * a = a$

There exists an inverse element for each element:

$\forall a \in G, \exists a^{-1} \in G \quad a * (a^{-1}) = a^{-1} * a = e$

A group is an *Abelian* or communicative if:

$\forall a, b \in G \quad a * b = b * a$

Definition 2.2 A set F with two operations denoted by '+' and '.' is a field if these conditions are met:

- (F,+) is an Abelian group and 0 is its identity element
- (F*, .) is an Abelian group and 1 is its identity element
- $\forall a, b, c \in F$ then $a.(b + c) = a.b + a.c$
- $(b + c).a = b.a + c.a$

Definition 2.3 A field that contains a finite number of elements is called a finite field known as *Galois field*.

Definition 2.4 The number of elements in a Galois field is called the order of the field and a Galois field with the order of q is denoted by GF(q).

Definition 2.5 Let a be an element of GF(q). The smallest positive integer m, such that $ma = 0$, is called the characteristic of the field.

Theorem 2.1 The characteristic of any finite field is prime.

Theorem 2.2 In a Galois field, the order of the field is a prime or a power of a prime.

2.2.2 Polynomials

A polynomial over *GF(p)* is an expression of the following form:

$$F(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x^1 + \cdots + a_n x^n$$

$a_i \in GF(P)$ and n, the degree of $F(x)$ is a nonnegative integer

Definition 2.6 A polynomial F(x) over GF(p) is called irreducible if it cannot be written as the product of some lower degree polynomials over GF(p).

Definition 2.7 F(x) is a polynomial over GF(p), F(0)≠0. The order of F(x) is the least positive integer t, such that F(x)|x^t-1

Definition 2.8 Let F(x) be a polynomial of degree m over GF(p). Polynomial F(x) is said to be a primitive polynomial if its order is $p^m - 1$.

Definition 2.9 A sequence of numbers $a_0, a_1, \ldots, a_m$ can be associated with a polynomial, called a generation function $G(X)$ [7].

$$G(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$$

2.2.3 Construction and Representation of $GF(p^m)$

Theorem 2.3 Let $F(x)$ be an irreducible polynomial of degree m over $GF(p)$. Then all polynomials over $GF(p)$ of degree less than m form a finite field $GF(p^m)$ of order p^m, if addition and multiplication are performed modulo $F(x)$.

Definition 2.10 If F and G are two fields such that $F \subset G$, then F is called a subfield of G and G is called an extension field of F. For example, $GF(2^m)$ is an extension field of $GF(2)$.

Definition 2.11 The binary extension field $GF(2^m)$ is constructed using an irreducible polynomial $F(z)$ of degree m:

$$f(z) = z^m + f_{m-1}z^{m-1} + \cdots + f_1 z + 1$$

Here $f_i \in \{0,1\}$ for $i = 1\ to\ m-1$. The constructed field is an extension of the basic field $Gf(2)$ and contains 2^m elements. Assuming that x is a root of $F(z)$, any field element of $GF(2^m)$ can be represented as a polynomial of degree m-1:

$$A = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \quad a_i \in \{0,1\}$$

2.2.4 Addition over $GF(2^m)$

The addition of the two field elements is carried out by pair-wise XOR operation [8]:

2.2.5 Multiplication over $GF(2^m)$ - General Bit-level Multiplications

Suppose $A, B \in GF(2^m) and\ F(x)$ is the modulus that defines the polynomial of the field. $a_i, b_i \in GF(2)$ are the bits.

$$A = \sum_{i=0}^{m-1} a_i x^i$$

$$B = \sum_{i=0}^{m-1} b_i x^i$$

$$C = AB\ mod F(x) = A \sum_{i=0}^{m-1} b_i x^i\ mod F(x)$$

$$= \sum_{i=0}^{m-1} A b_i x^i\ mod F(x)$$

$$= (b_0 A + b_1 xA + \cdots + b_{m-1}x^{m-1}A)\ mod F(x) \qquad (2.1)$$

Algorithm 2.1, and 2.2 show the bit-level multiplication from low bit to high bit and from high bit to low bit which are obtained according to equations (2.1), and (2.2) [9].

Algorithm 2.1: Bit-level algorithm of multiplication from low bit to high bit in $GF(2^m)$

$Input: A, B, F(x)$
$Output: C = AB \ mod \ F(x)$
$D \coloneqq A$
$C \coloneqq 0$
$For \ i = 0 \ to \ m - 1 \ do \ \{$
$C \coloneqq C + D.b_i$
$D \coloneqq xD \ mod \ F(x)$
$\}$

Bit-level multiplication can be done from high to low bit-level in $GF(2^m)$ according to equation (2.2):

$$C = AB \ mod \ F(x)$$

$$= A(b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_0) \ \ mod \ F(x)$$

$$= (\ldots(Ab_{m-1}x + b_{m-2})x + \cdots)x + Ab_0 \ \ mod \ F(x)$$

$$(\ldots(Ab_{m-1}x \ \ mod \ F(x) + b_{m-2})x \ \ mod \ \ F(x) + \cdots)x \ \ \ mod \ F(x) + Ab_0 \qquad (2.2)$$

Algorithm 2.2: Bit-level algorithm of multiplication from high-to-low bit in $GF(2^m)$

$Input: A, B, F(x)$
$Output: C = AB \ mod \ F(x)$
$C \coloneqq A.b_{m-1}$
$For \ i = 0 \ to \ m - 1 \ do \ \{$
$C \coloneqq xC \ \ mod \ \ F(x)$
$C \coloneqq C + A.b_i$
$\}$

2.3 Theory and Operation of Linear Feedback Shift Registers

Linear feedback shift register [LFSR] devices are used in built-in self test (BIST) designs. They can be used to carry out the response of the compression which will be explained in more details in chapter3.In this section some of the properties associated linear feedback shift registers are presented from [7].These circuits are based on cyclic redundancy checking (CRC) and have only clocks as their input. Symbolic form of a feedback shift register is shown in figure 2.1. Each cell is assumed to be a clocked D flip-flop. When these circuits are clocked repeatedly, they go through a fixed sequence of states. For example $0,1,\ldots,2^n$ is the sequence for a binary counter consisting of n flip-flops.

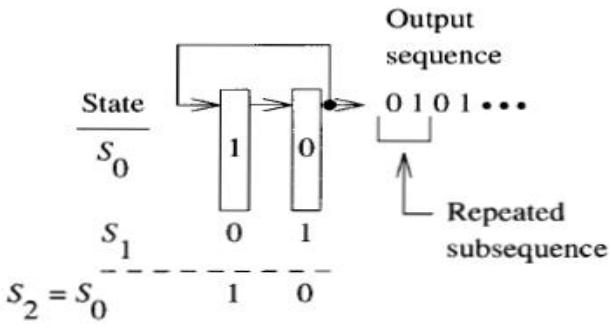


Figure 2-1 Feedback shift register

The maximum number of states for each device is 2^n. A linear circuit is a logic network of basic components: Flip-flops, modulo-2 adders, modulo-2 multipliers. All operations of such circuits are done modulo 2. The response to a linear combination of stimuli is the linear combination of the responses of the circuit to the individual stimuli [7]. Type 1 (external-XOR) and type 2 (internal-XOR) LFSR are shown in figure 2.2 and 2.3 respectively.

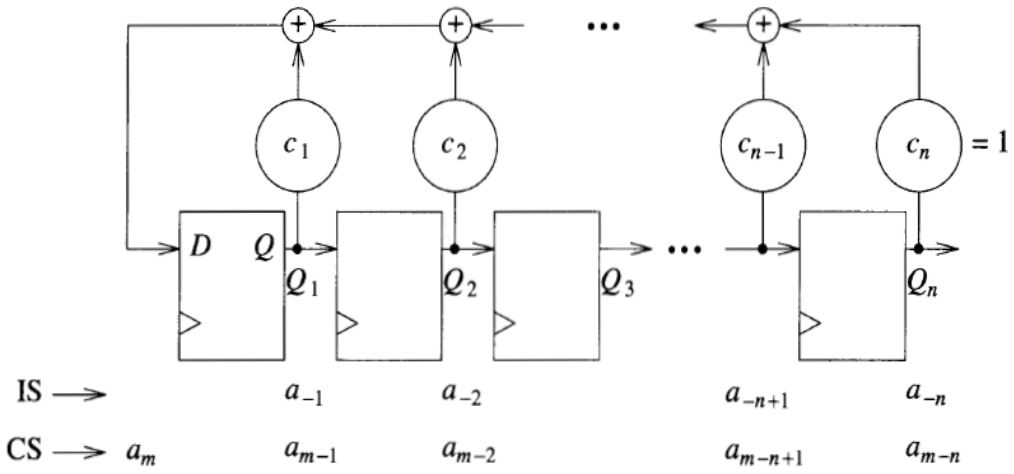


Figure 2-2 Type 1 LFSR

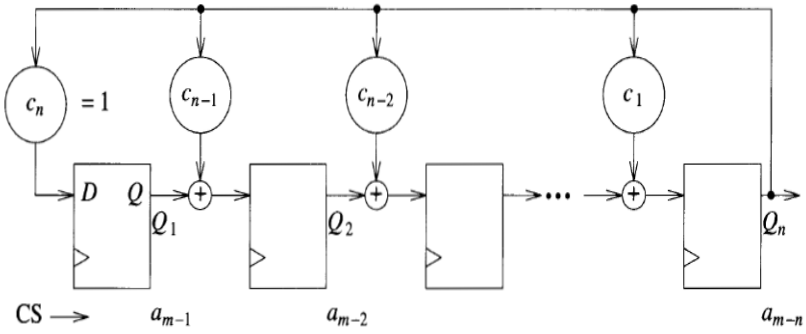


Figure 2-3 Type2 LFSR

Definition 2.9 Let $\{a_m\} = a_1, a_2, \ldots$ where$a_i = 0\ or\ 1$, represent the output sequence that is generated by an LFSR. Then the sequence can be expressed as:

$$G(x) = \sum_{m=0}^{\infty} x^m$$

Multiplication and division of the polynomials are done modulo 2 [7].

Structure of type 1 LFSR is such that if the current state (CS) of Q_i is a_{m-i}, then $a_m = \sum_{i=1}^{n} c_i a_i$. This means that the operation of the circuit is a recursive function. c_i is the feedback coefficient, and $a_{-1}, \ldots a_{-n}$ are initial state (IS) of the device.

It can be proved that $G(x)$ is a function of initial states of the LFSR and the feedback coefficients [7]. Equation (2.3) shows this property of function $G(x)$.

$$G(x) = \frac{\sum_{i=1}^{n} c_i x^i \left(a_{-1}x^{-i} + \cdots a_{-1}x^{-1}\right)}{1 + \sum_{1}^{n} c_i x^i} \tag{2.3}$$

Definition 2.12 The denominator of equation (2.3) is known as the characteristic polynomial of the sequence $\{a_m\}$ as is denoted as [7]:

$P(x) = 1 + c_1 x + c_2 x^2 \ldots + c_n x^n$.

If $a_{-1} = a_{-2} = \cdots = a_{1-n}$, and $a_{-n} = 1$, then G(x) will be reduced to $1/P(x)$. Assuming that sequence $\{a_m\}$ is cyclic with period p as shown in equation (2.4):

$$P^* = \frac{1}{P(x)} \quad = \frac{a_0 + a_1 + .. + a_{p-1}x^{p-1}}{1 - x^p} \tag{2.4}$$

Definition 2.13 If the initial state of an LFSR is $a_{-1} = a_{-2} = \cdots = a_{1-n} = 0$, $a_n = 1$, then the LFSR sequence $\{a_m\}$ is periodic; the period is the smallest integer *k* for which $P(x)$ divides $(1 - x^k)$

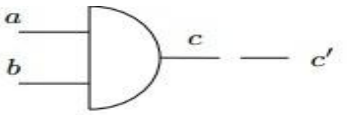
Definition 2.14 If the period of the sequence generated by an *n*-stage LFSR is $2^n - 1$, then it is called a *maximum length sequence*.

Definition 2.15 The characteristic polynomial associated with a maximum-length sequence is called a *primitive polynomial.*

2.4 Fault Modeling

Faults can be investigated at different levels of abstraction of circuits, such as gate level or architectural level. Gate level faults are open faults, short faults, and stuck-at faults. Figure 2.4 shows an open fault circuit with its corresponding truth table. C is the correct output and C' is the faulty one. In higher architectural-level, the error can be modeled as equation 2.5 [9]:

$$e_c = c \oplus c' \tag{2.5}$$



a	b	c	c'	e_c
0	0	0	1	1
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Figure 2-4 Open Fault Model and Truth Table

2.5 Detecting errors in finite field operations

In past, circuits capable of concurrent error detection were only dedicated to critical applications such as, railway control. But today with shrinking size of electronic devices and the fact that circuits are more susceptible to internal and external noises, self-checking circuits are more desirable. In order to detect errors occurred in any finite field operations, different approaches, such as, parity bits, scaling techniques has been proposed in literatures [7, 9].

- Parity Prediction: Works based on predicting the parity of the output from the parity of the input. This method can only find odd number of erroneous bits.
- Scaling technique: Representing the value N by the number AN, known as AN codes.
- Residue codes (or inverse residue codes): Computing a residue for each input and, then predicting the residue of the output based on them. It essentially means representing the value N by the pair (N,C) where C is N mod A or (N-N mod A) and is called the check part.
- Time redundancy based techniques: Re-computing the result with shifted operands.

The main problems associated with the design of self-checking circuits are hardware cost and time consumption of the design. Developed Parity prediction designs allow high fault coverage with low hardware cost in self-checking data paths for blocks such as, adders, Shifters, register files, etc. However, parity prediction self-checking multipliers has much higher overhead. Using parity codes for checking memory systems and register files provides fault secure property with low overhead, however, arithmetic operators produce output errors that are not detectable by parity code; in multipliers, fault secure design based on parity prediction requires hardware overhead in the range of 40% to 50% [10]. Therefore, in the case of multipliers, residue codes can be a good alternative to achieve fault secure property.

2.6 Residue Codes for Checking Arithmetic Operations

Modulo A (A is a positive integer) arithmetic circuits have found many applications in digital signal processing, computing convolution, elliptic encryption systems and fault tolerant digital systems [11].

Arithmetic codes in general, add some bits as an information part to the input before the operation, to be checked after the operation. The information part can be the original number multiplied by a constant number or, it can be a check part representing the modulo m of the information part. One of the most important classes of modulus M are Mersenne numbers that are in the form of $2^m - 1$. These circuits are known to have simple hardware implementation as there is no need for division to calculate the residue. Residue checking circuit requires compact hardware [10]. It needs a small arithmetic operator (adder, multiplier) which can add or multiply operands of the size of the check parts of the operands, A modulo generator which can compute the residue which is independent of the size of the operator. It also requires an arithmetic code checker and a code translator with the size proportional to the size of the operands.

Residue checking has been studied for error detection in arithmetic processors. Many papers have been published to evaluate residue checking in terms of delay, fault coverage and hardware overhead. So far the results show that it may not be the most efficient method for checking addition and subtraction circuits, however, it is shown that the method has a very good performance for checking multiplier circuits especially large multipliers [10].

Residue checking is a method to check the accuracy of an operation which can be done in parallel with the actual operation. This means that it can provide concurrent error detection. For example for checking an addition operation, the adder and the residue checker are implemented as an independent logic functions. Then an easy to implement modulus should be selected to check the residue of the operands and the result to see if they match. Figure 2.5 from [12] illustrates the idea of error detection using residue checking.

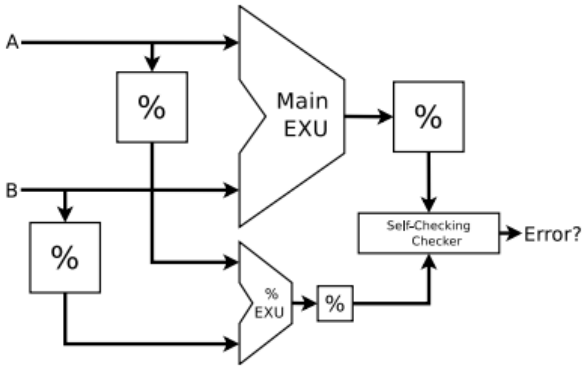


Figure 2-5 Error detection scheme

Another method to detect errors in an arithmetic operation is parity prediction and because of its low hardware implementation, it is sometime considered as more efficient than residue checking. The most important problem with this method is that it can only find odd number of errors and because of the occurrence of errors with random nature in arithmetic operations, there might be some errors that cannot be detected by parity prediction method.

In a number system two numbers A and B are shown as the following [14]:

$$A = a_{n-1}r^{n-1} + \cdots + a_0 r^0$$

$$B = b_{n-1}r^{n-1} + \cdots + b_0 r^0$$

The congruence relationship of the two numbers modulus m is produced by equation (2.6):

$$A = (quotient \times m) + R \qquad (2.6)$$

The benefit of the residue checking is that numbers that are congruent to the same modulus can be added, subtracted and multiplied, and the result is still a valid congruence [13]. This means that the residue of the sum is equal to the sum of the residues. For example, modulo 3 residues can detect single-bit errors using only two check bits in the radix 2 number system. The

structure is that, in parallel with the addition operation, separate residues for A and B will be obtained in two residue generator blocks. These residues are then added in an adder unit and the resulting sum is applied to another residue calculator. Final step is to compare the residue of the sum against the residue of the sum of the residues to indicate an effort if they do not match [14]. This structure can be illustrated in the mathematical equation of (2.7) as the following form:

$$R(A) \equiv A \; mod \; m$$

$$R(B) \equiv B mod \; m$$

$$R(A) + R(B) \equiv (A mod \; m) + (B mod \; m)$$

$$\equiv (A + B) mod \; m$$

$$\equiv R(A + B) \qquad (2.7)$$

Table (2.1) shows the result of 4-bit binary number modulo 3. Note that the residue representation needs only 2 bits. The division operation to find the residue was implemented by a sequence of shifts and addition or subtraction depending on the previous state of the previous high order carry-out [14]. The carry out determined the quotient bit at any level and also the next operation.

$$q_0 = \begin{cases} 0, if \; carry - out = 0; \;\; next \; operation \; is \; 2A + B \\ 1, if \; carry - out = 1; next \; operation \; is \; 2A - B \end{cases}$$

Table 2-1 4-Bit Binary Numbers and Their Residue Modulo-3

Binary Data	Modulo-3 Residue
0000	00
0001	01
0010	10
0011	00
0100	01
0101	10
0110	00
0111	01
1000	10
1001	00
1010	01
1011	10
1100	00
1101	01
1110	10
1111	00

The use of arithmetic codes for adders/ALUs has reduced interest because of the complexity and the large overhead of the circuit. The size of the multiplier however, is proportional to the square of the size of the operands, meaning that for large multipliers residue codes is a good solution [10].

A CAD system has been developed in [10] allows automatic generation of different schemes of self-checking multipliers based on residue codes. The experimental result of this paper confirmed that residue checking for multipliers bigger than 8×8 reduces the area overhead significantly, even down to 5% or 6% for 64×64 multipliers.

For simplicity it is assumed that faults affect a single gate at a time. Two different schemes for an adder cell can be defined to model fault which are shown in figures 2.6, and 2.7 from [10]. In the model shown in figure 2.6 sum and carry-out are independently computed as the result of XORing. In figure 2.7 however, A⊕B result which is shown as P, is to calculate carry-out. Clearly the latter model results in propagating error of P to C.

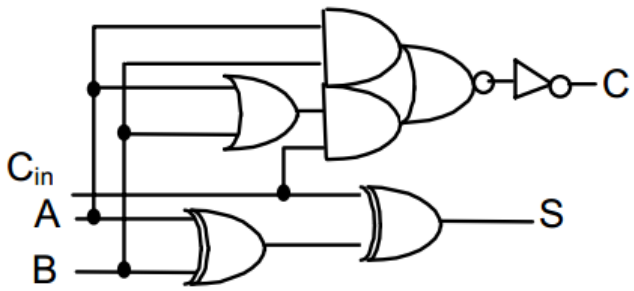


Figure 2-6 Adder cell with no logic shared

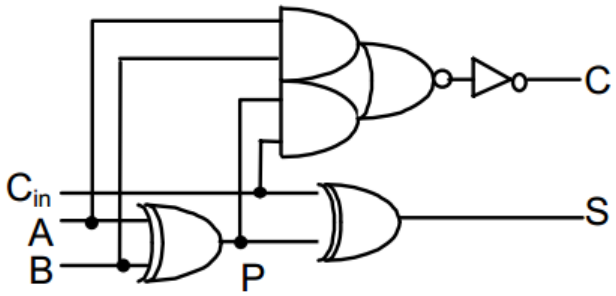


Figure 2-7 Adder cell with some logic shared

The result of applying residue codes for different schemes of multipliers are summarized here [10].

Array Multipliers

Array multipliers are implemented by network of full and half adders to calculate the partial products. To detect errors using residue codes in Array Multipliers or any other type of multiplier, it should be considered that if the arithmetic value of the error produced by multiplier is multiple of the base of the residue codes, then, the error would go undetected. Since we are working with binary system, any error at i^{th} bit of the output bits has the arithmetic value of $\pm 2^i$. So, $m = 3$ for the scheme of figure 2.7 can be selected as the check base because 2^i is never a multiple of 3; it is known as the cheapest check base because it only needs 2 bits to register the residue.

Check-base 3 however, cannot detect some of the faults in the scheme of 2.7. In this case, arithmetic value of the error will be the form of $\pm 3 \times 2^i$ which is multiple of 3. In this case base 7 can be used so that the base does not divide the error.

Multipliers with Wallace Trees

Accumulating partial products using Wallace trees improves the delay known for Array multipliers [14]. Same as the Array Multipliers, the network of ripple-carry adders that add the partial products will add the arithmetic value of the error to the final result. So here also base 3 achieves fault secure property. However, because of the delay known for the ripple-carry adder, the carry propagate adder must be implemented in a carry look-ahead manner [10]. Now the check base in this case would be different. Important fact need to be considered is that carries here are generated by a carry look-ahead function which has a structure different than the adder cell network. This leads to propagation of an error in a more complex way.

Table (2.2) summarizes the check bases that can be used for fast adders obtained by the software developed in [10].

Table 2-2 Check Bases

Check Base	16 bits	32 bits	64 bits
Kogge & Stone	$2^6 - 1$	$2^{12} - 1$	$2^{12} - 1$
Han & Carlson	$2^4 - 1$	$2^8 - 1$	$2^8 - 1$
Brent & Kung	7	7	7
Sklansky	3	3	3
Carry Lookahead Unit	3	3	3

Booth Multipliers

Booth multipliers are also considered as fast multipliers because it reduces the number of partial products by one half by encoding the biggest input operand. For this type of multiplier, the lowest check base for fault secureness is 7 [10].

2.7 Residue Generator

The residue of a binary number modulo $2^k - 1$ can be implemented by splitting the bits into bytes of k bits and performing modulo m addition of these bits. This means that residue generators of the form $m = 2^k - 1$ can be implemented with less cost as there is no need for performing division. Modulo m addition of two k-bit is performed using a k-bit adder in which the carry output is used as a carry input for the next adder. This is basically what Carry End-Around adders do. So a residue generator block can simply be implemented by a tree of k-bit Carry End-Around adders without need to do any division [15]. For instance consider $m = 7 = 2^3 - 1$, then applying congruence theory the residue will be calculated as shown in equation (2.8)

$$(2^3)^k \equiv 1 \; mod(7) \qquad (2.8)$$

Thus, the residue of powers of $2^0, 2^{3\times1}, 2^{3\times2}$, ... modulo 7 is 1. $2^1, 2^{3\times1+1}, 2^{3\times2+1}$, ... and so on. So a full adder can be used to add X_0, X_3, and so on. Another full adder then must be used to add X_1, X_4, and so on. In other word, the period of the bits is 3.

Modulo 9 also worth noting as the residue of 1, 8, 64, ... modulo 9 is +1, -1, +1, …

Therefore, $X_0, -X_3, X_1, X_6, \ldots$ has to be added. This operation can be done using a full adder and inverting the bit that has to be subtracted. Figure 2.8 and 2.9, show residue generators for a 8-bit number modulo 7 and 9 respectively [10].

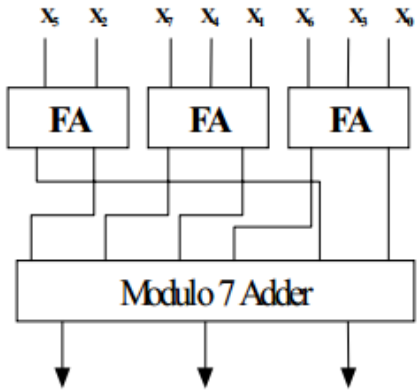


Figure 2-8 8-bit number modulo 7

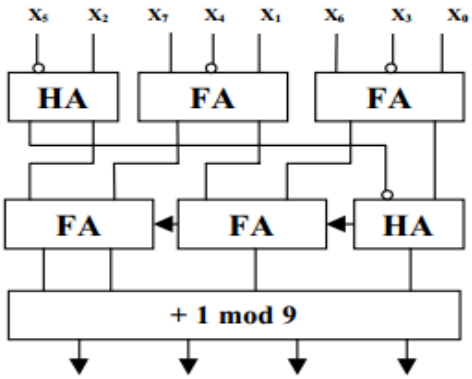


Figure 2-9 8-bit number modulo 9

The overall block diagram of the self-checking multiplier using residue codes is shown in figure (2.10).

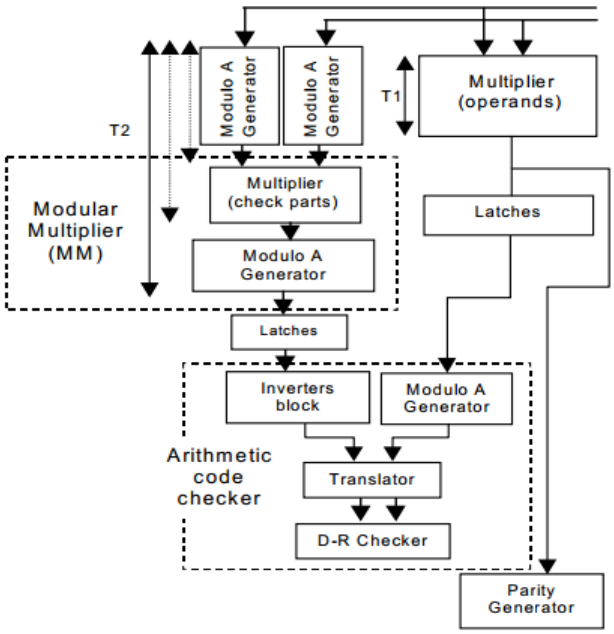


Figure 2-10 Self-checking Multiplier

3. CHAPTER 3
APPLICATION OF RESIDUE CODES FOR ERROR DETECTION IN MIXED-SIGNAL SYSTEMS

3.1 Introduction

This chapter focuses on application of residue calculation for arithmetic and algebraic compaction, to test mixed-signal systems. General aspects of compression techniques are also discussed here. Several low-cost arithmetic and algebraic compactor schemes are introduced. The tolerance bounds for the result of compaction are estimated and the aliasing rate is evaluated.

3.2 General Aspects of Compression Techniques

Conventional testing methods involved bit-by-bit compression of the output with the correct values that has been previously computed. Clearly, this approach requires a significant amount of memory. An alternative approach is to save the information in a compressed format known as a signature [16]. This concept is illustrated in figure 3.1 [17]. If the signature obtained from the device under test does not match the pre-calculated signature of a fault-free circuit, then a fault is detected. A practical compression technique should be easily implemented and should not introduce signal delays.

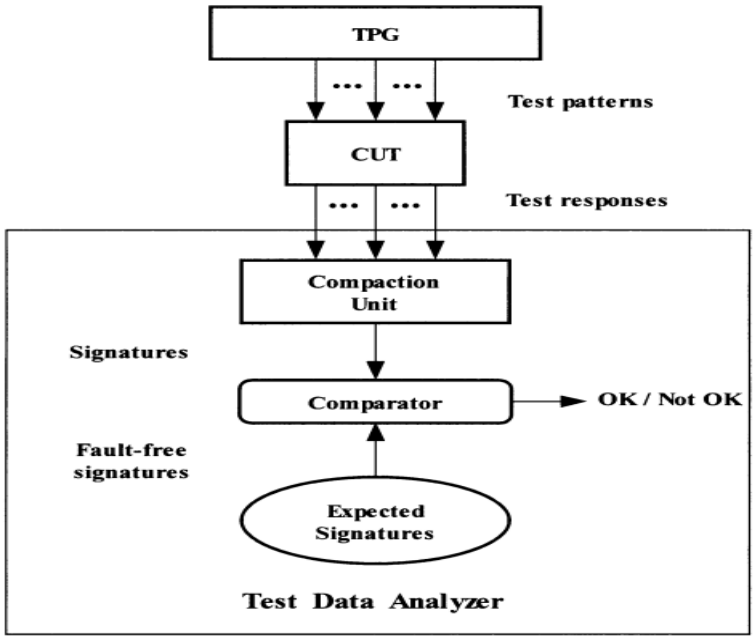


Figure 3-1 Test Data Analyzer

The challenge facing this method is insuring that the faulty and fault-free signatures are different because, a fault may produce offsetting errors. The erroneous output response is said to be an *alias* of the correct output response.

Three methods have been suggested in [7] to measure the aliasing associated with a compression technique:

1. Simulation of the circuit and the compaction technique which requires fault simulation. If the test sequence is long this method is expensive.
2. In this method, output responses are classified into categories, such as single-bit error or burst errors, which are errors that lie within a fixed number of patterns of one another.

3. The fraction of all possible erroneous response sequences that cause aliasing are computed. In this method it is assumed that all possible output sequences are equally likely.

3.3 Ones-Count Compression

If the output signature of a circuit is shown as $R = r_1 r_2 \dots r_m$, then in ones counting, the signature of the response, $IC(R)$ is the number of 1s appearing in R. The compressor is a counter [7] and the degree of compression is $\lceil \log_2(m+1) \rceil$. Figure 3.2 describes testing with one-counting compression technique.

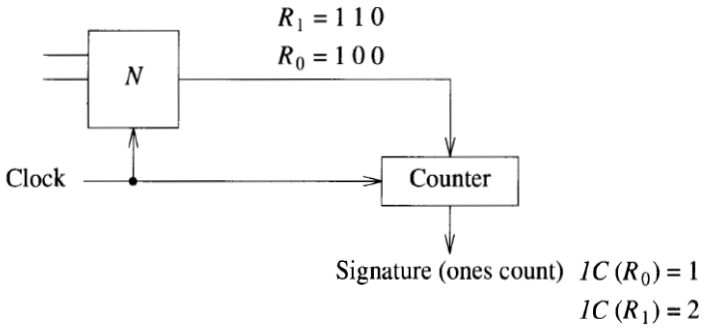


Figure 3-2 One-Count Compression

3.4 Transition-Count Compression

In this method of testing, is the number of transitions in the output data stream. This means that for a sequence of $R = r_1 r_2 \dots r_m$, the transition count will be obtained from equation (3.1) [7]:

$$TC(R) = \sum_{1}^{m} (r_i \oplus r_{i+1}) \tag{3.1}$$

This implies that the response-compression circuit needs a transition detector, and a counter with $\lceil \log_2(m+1) \rceil$ stages. Transition detector consists of a flip-flop and a XOR gate that is shown in figure 3.3.

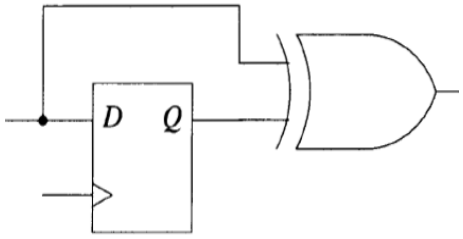


Figure 3-3 Transition Detector

3.5 Parity-Check Compression

Parity-check compression circuit is also explained in [7]; it consists of D-flip-flop and XOR gate. Its implementation is a linear feedback shift register whose primitive polynomial is $G(x) = x + 1$. If the initial state of the flip-flop is 0, the signature S is the parity of the circuit response. Same as it was said in the previous chapter, faults that create an even number of errors are not detected with this method. As the number of input streams increases, the probability of aliasing approaches to the value of 0.5. Figure 3.4 illustrates the parity-check compression scheme.

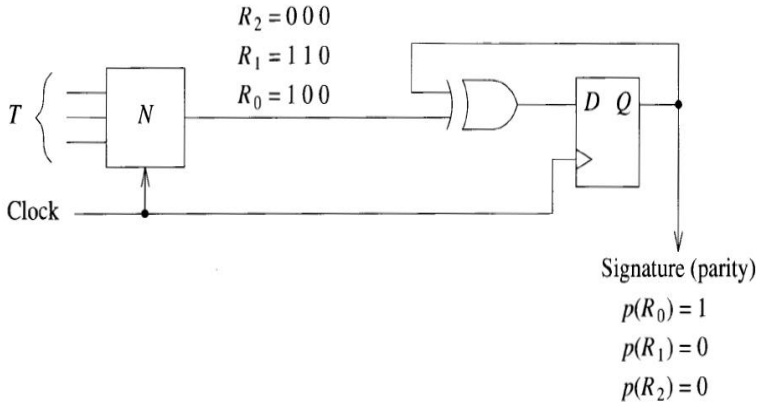


Figure 3-4 Parity Check Compression

Using parity-check compression for multiple-output circuits has two disadvantages. If it is implemented by replacing the input XOR gates by a multiple-input gate or a network of XORs, an error internal to a circuit may affect more than one output line and the error gets propagated to an even number of output lines. Also, if a separate parity-check compressor is used in each output, the hardware cost will be high. This means that this technique is not very efficient for testing multiple-output circuits.

3.6 Syndrome Testing

Applying all 2^n test vectors to an n-input combinational circuit is called syndrome testing [7], syndrome S (or signature) is the normalized number of 1s in the resulting bit stream. It means that if K is the number of minterms in the function f, then $S = K/2^n$. The syndrome of a 3-input AND gate is 1/8 and for a 3-input OR gate is 7/8. Any function can be realized in such a way that all single stuck-at faults is syndrome detectable.

If two circuits C_1 and C_2 have no shared input, and $S(F) = S_1$ and $S(G) = S_2$, then the input-output syndrome relation of the circuit as a function of the type of gates is listed in Table (3.1) [7].

Table 3-1 Syndrome for circuit having non-reconvergent fanout

Gate Type for C	Syndrome S_3
OR	$S_1 + S_2 - S_1S_2$
AND	S_1S_2
NAND	$1 - S_1S_2$
NOR	$1 - (S_1 + S_2 - S_1S_2)$
XOR	$S_1 + S_2 - 2S_1S_2$

3.7 LFSRs Used as Signature Analyzers

Signature analysis is a compression technique based on the concept of cyclic redundancy checking (CRC) [18]. Signature generator with a single-input LFSR is the simplest scheme of this technique. The contents of this register are the signature after the last input bit has been sampled. Response sequence which is fed to the signature analyzer is denoted by $P(x)$. This sequence then will be divided by the characteristic polynomial denoted by $G(x)$. This implies that signature analysis is based on polynomial division; the "remainder" left in the register after completion of the test process is the final signature. This can be represented in the equation form of (3.2).

$$P(x) = G(x)Q(x) + R(x) \quad (3.2)$$

Figure (3.5) from [7] explains how a single-input signature analyzer works. $P(x)^*$, as was defined previously, is the reciprocal characteristic polynomial of the LFSR; it is given in the example as:

$P(x)^* = 1 + x^2 + x^4 + x^5$

The input is the 8 bit test sequence 11110101 which is equivalent to $(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1$. As it is shown in figure 3.5, the result of division will be, $R(x) = x^2 + x^4$ and $Q(x) = x^2 + 1$.

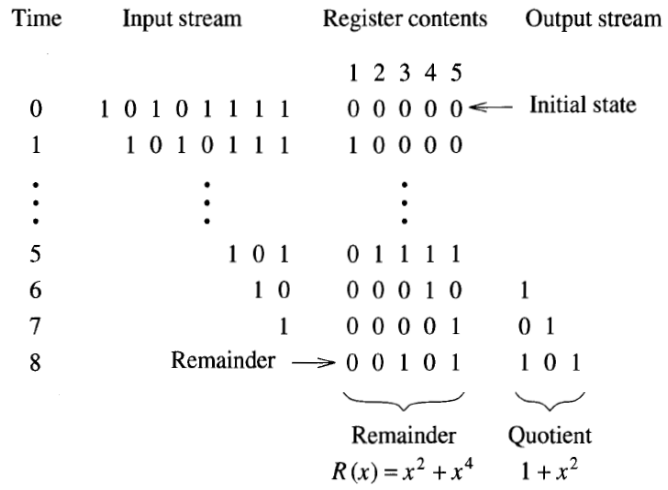


Figure 3-5 Example of single-input signature analyzer

3.8 Multiple-Input Signature Registers

It is possible to test circuits that have multiple outputs using signature analysis technique. The most common way to do that is using a multiple-input (MISR) register which is shown in figure 3.6 [19]. The circuit is assumed to have n inputs; it operates similar to n-input signature analyzer.

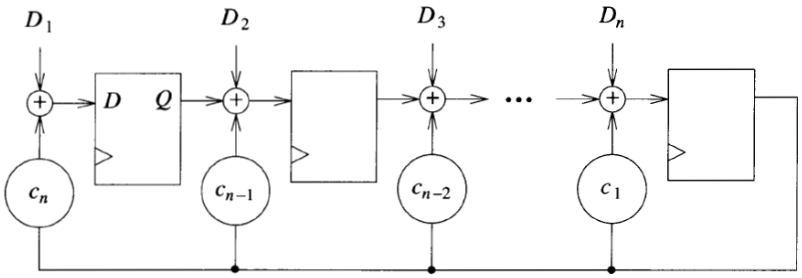


Figure 3-6 Multiple input register (MISR)

3.9 Arithmetic Compaction Circuits for Testing Mixed-Signal Systems

Many electronic systems, such as those for audio and image processing, video compression, speech recognition, digital communication etc., deal with signals that are mix of analog and digital nature. It is highly desirable to insure the reliability of these devices by detecting any faults as soon as they occur in the system and make sure that the system is immune to sudden or gradual failure. Gradual failures may remain unnoticed and become a greater threat in the future. To test this type of devices signature analysis method can be applied; analog stimuli can be used to cover nearly the full-scale range (FSR) of the device. Then the output responses in the digital form should be compared with the expected values. The comparison can be done after each test stimuli has been applied or after the complete test set has been run. If comparison is done after each test set is applied, the availability of memory will be an issue. Although performing the comparison after the full set of test stimuli has been applied might lead to undetected errors, since the error escape rate and the test hardware overhead are small, this method is considered more practical for built-in self-test [7].

Compaction methods are based on the estimation of the residue of an arithmetic sum which is referred to as the signature of output when the input is fed by test stimuli. The output signature

should be compared with the fault free circuit's signature and the fault signal should be generated if the error is out of the predefined tolerance bound. Erroneous patterns caused by modular operation which is called aliasing may go undetected. In the presence of analog circuits and an analog - to- digital converter aliasing phenomenon provokes, and produces errors which would distort the output codes and the compaction result. Therefore, compaction testing for mixed-signals deals with a set of permissible signatures, and not a single "reference" for digital systems. The cardinality of this set should be kept at a minimum in order to achieve a low aliasing rate [7].

Compaction circuits with an arbitrary modulus have been developed in this study. The set of permissible signatures that are within the tolerance bound from empirical data requires the availability of the reference system. So, the tolerance bound is computed analytically in this work to avoid the complexity.

3.10 Testing Mixed Signal Device

One approach to test mixed-signal system is based on the estimation of metrics such as gain, offset, INL, DNL, SNR, ENOB etc [20]. Although these static and dynamic metrics carry important information about the functionality of the system, however, their calculation involves extensive digital processing which would impose large overhead on the system and is not acceptable for built-in implementation.

Figure 3.7 shows the scheme of on-line testing for an analog-to-digital converter device (ADC), using compression technique. As it is shown in the figure, the Test Pattern Generator (TPG) generates digital pattern that is send to the digital-to-analog converter (DAC) which is then feed to the Analog Comparator (AC). When the operational signal that is been fed to the

device under test matches the output of the DAC, then a passing signal is generated by the AC unit. This HIT signal activates the Modulo Adder block which is responsible for calculation of the residue which is also referred as a compactor or signature analyzer [7].

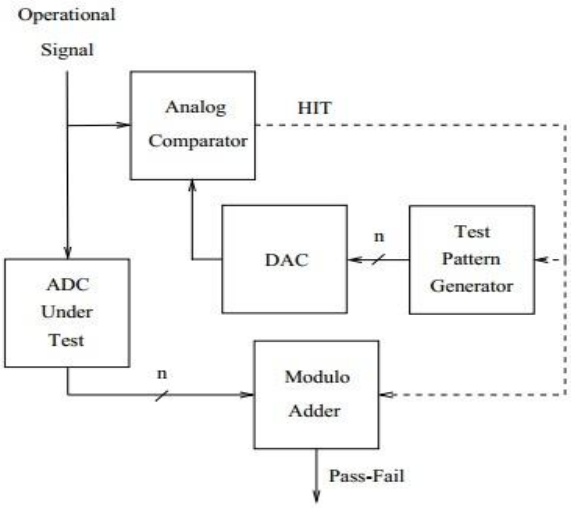


Figure 3-7 On-line Testing of ADC Device

Finding a real fault-free ADC device is a main challenge. Characteristic function of an ideal ADC is shown in figure 3.8. Another challenge which complicates the use of algebraic compaction for the scheme is caused by the uncertainty of the output coded voltage when an input voltage is in the interval that contains the middle point of the quantization bin; an interval code will be produced at the output, even though there is no fault.

Testing of a 3-bit fault-free MSS with FSR of 8 volt, with a transfer characteristic shown in figure (3.8), using compression technique is analyzed here. Several compaction circuits for testing this ADC device will be presented in the next chapter.

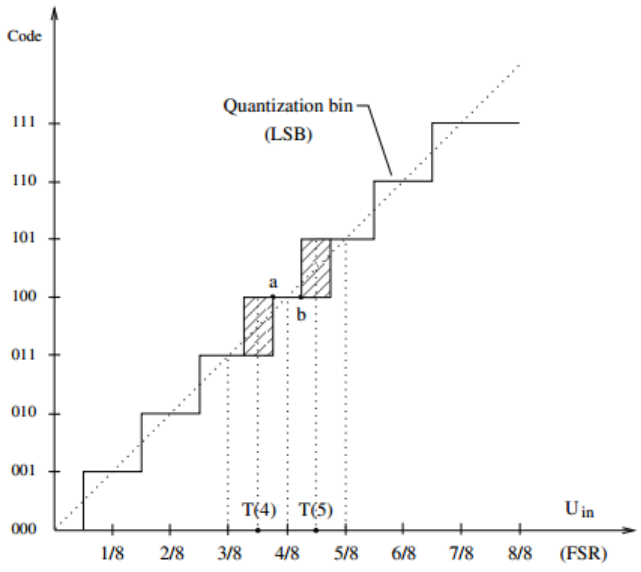


Figure 3-8 Transfer function of a 3-bit MSS

The transfer function of the system is such that the transition voltages between the steps, T (k), k= 1,…, 2n-1 (n is the resolution of the MSS) may fluctuate within the specific tolerance bounds. Inequality (3.3) shows this bound.

$$\frac{k-1}{2^n}FSR \leq T(k) \leq \frac{k}{2^n}FSR \qquad (3.3)$$

If n= 3 bits then (3.3) will be simplified to (3.4):

$$k-1 \leq T(k) \leq k \qquad (3.4)$$

The shaded boxes surrounding the ideal transitions show the bounds. Worst case happens when *a* and *b* coincide. If the input signal matches the middle points of the quantization bin, the output can take three values. For example if the stimuli match the middle point of 4 volt, the output will take three possible values 011, 100, 101. Excluding these points it can be assumed that test stimuli are in the neighborhoods of the corresponding transition voltages T (k) of the transfer function. With this assumption the number of acceptable output codes reduces to two per

input stimulus which leads to reducing the distortion of the final signature. For instance, if the input is set to be T (4) = 3.5 V, the output code will be 011 or 100.

To develop a valid fault model it can be assumed that faults in a MSS would change the transfer function such that the transition points, T (k), move beyond the permissible bound, i.e. (k) < k-1, or T (k)>k. Also we assume that it can only be a single transition within each of the intervals of [k-1, k], $k = 1, \dots, 2n - 1$

Now with the fault model described above, all the faults will be detected by the input stimuli set, which only includes single points from each of the intervals.

4. CHAPTER 4
RESIDUE CALCULATOR CIRCUIT ARCHITECTURE FOR MIXED-SIGNAL SYSTEMS

4.1 Operation of Arithmetic and Algebraic Compactor

Algebraic and arithmetic compactors are the two types of compactors that can be applied for testing circuits. Both methods utilize the same theory of computing the residue. The arithmetic compactor is referred to as the modulo sum circuit. Modulo sum is just a special case of a residue code.

4.2 Operation of Arithmetic and Algebraic Compactor

The operation of the algebraic compactor can be explained by a n-input analyzer. Assuming that the MSS produces all n bits simultaneously, a multiple input signature register (MISR) can be considered. In this study a 3-input signature register is examined which receives for example the following sequence of 3-bit data, α4, α2, α1, α0. This sequence of data can be described by expression (4.1) [7]:

$$(\alpha^4 y^3 + \alpha^2 y^2 + \alpha y + 1) \bmod G(y) \qquad (4.1)$$

$G(y)$ is equal to $y+\alpha$, where α is the primitive element of the field $GF(2^3)$. In this example α is considered as the root of the primitive polynomial $g(x) = x^3 + x + 1$. Expression (4.1) can be implemented using an iterative addition and shift operation by factorizing it:

$$(y(y(y(y.0+\alpha^4)+\alpha^2)+\alpha)+1)\text{mod}\,(y+\alpha) \qquad (4.2)$$

In general form expression (4.2) can be considered as a set of repetitive operation of the form of expression (4.3):

$$(\alpha^i+\alpha^j)\text{mod}\,(y+\alpha) \qquad (4.3)$$

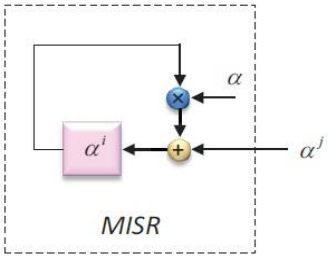


Figure 4-1 Operation of the 3-input signature register

Figure (4.1) shows the operation of MISR for implementing expression (4.3). The operation can be performed using theories of polynomial division in $GF(2^{3})$ that was introduced before as expression (4.4):

$$\alpha^i y+\alpha^i \quad | \frac{y+\alpha}{\alpha^i}$$

$$+\ \underline{\alpha^i y+\ \alpha^i\alpha}$$

$$\alpha^k=\alpha^i y+\alpha^j \qquad (4.4)$$

Figure 4.2 shows the register transfer level (RTL) scheme of 4.1. This circuit will produce α^1 as its signature which is equivalent to 010 in vector form.

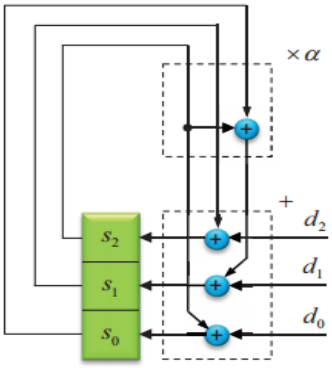


Figure 4-2 3-Input Residue Calculator in RTL Form

Another form of compactor circuit is called an arithmetic compactor which works similarly to the algebraic one. The difference is that here the circuit divides the input sequence by the number $G = 2^3 - 3$ where 3 is the primitive element of the arithmetic finite field GF(5) .This is similar to the algebraic circuit that divides the input sequence by the polynomial of G(y) = y-α .

Considering the same example if the sequence of $\alpha^4, \alpha^2, \alpha^1$ and α^0 is matched with numbers 6, 4, 2, and 1 which are fed to the compactor circuit, performing division of the octal number 6421 with respect to modulo 5.

Expression (4.5) illustrates the same compression procedure in an arithmetic circuit. Figure (4.3) shows a symbolic form of this operation.

(8(8(8(8.0+6)+4)+2)+1)mod5 (4.5)

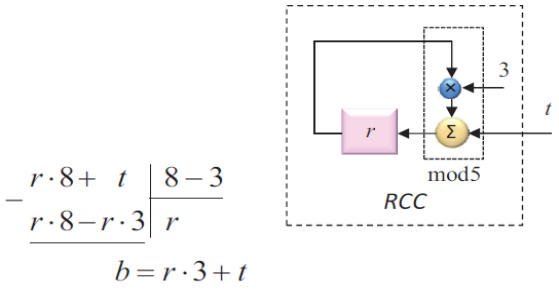


Figure 4-3 3-Input Arithmetic Compactor

Considering the special case of algebraic and arithmetic compactors when α and 3 are replaced by 0; then for both compactors the same circuits will be obtained which does not have feedbacks. The probability of the undetected error for this circuit is same as the one with feedback. Also a case can be considered where α and 3 are both replaced with 1. To implement the residue calculation circuit (RCC) in a general form, equation (4.6) should be considered.

$$r \times 8 + t \equiv r \times 3 + t \ (\text{mod}5) \tag{4.6}$$

Taking to account expression (4.6), RCC can be implemented as it is shown in figure 4.4.

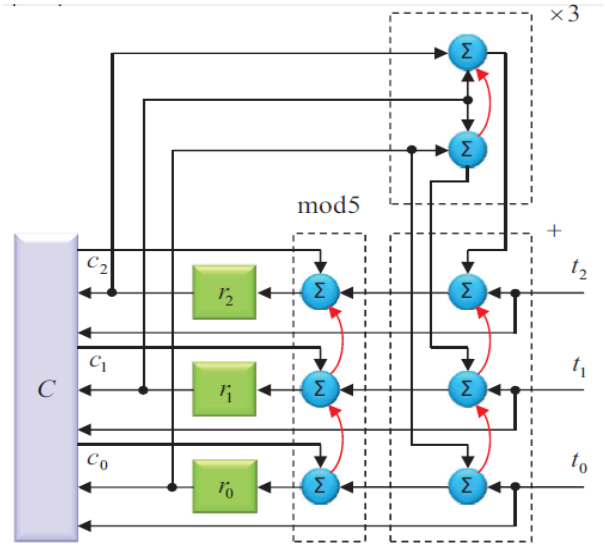


Figure 4-4 3-Input Ternary Arithmetic Compactor

C is a combinational circuit unit which provides feedback signals $c_2, c_1,$ and c_0 based on the input and the present state signals t and r respectively. Red arrows show the carry propagation.

$c_2, c_1,$ and c_0 can be obtained from equation (4.7), (4.8), and (4.9).

$$c_0 = r_0 t_1 \overline{(r_1 \oplus t_2)} + r_1 \bar{t_2}(\bar{r_0} + \overline{t_0 t_1}) + r_2 + \bar{r_1} t_2 (\bar{r_0} t_1 + t_0 \bar{t_1} + r_0 \bar{t_0}) \quad (4.7)$$

$$c_1 = r_2 \bar{t_2}(\bar{t_0} + \bar{t_1}) + \bar{r_2} t_2 (\bar{r_0} t_0 + \bar{r_1} t_1) + r_1 (\bar{r_0} + \bar{t_2}) + r_0 (t_1 \oplus t_2) \quad (4.8)$$

$$c_2 = r_0 \bar{r_1} t_0 t_1 t_2 + r_1 (r_0 t_1 \oplus t_2 + r_0 t_0 \bar{t_2} \quad (4.9)$$

If this circuit is fed by sequence of 3-bit octonary numbers 6, 4, 2, 1, applying equation (4.6), it can be said that the circuit is computing the residue of the ternary number $6421 \bmod 5$.

4.3 Low-Cost Compactor schemes

Now, if the RCC scheme shown in figure (4.4) is used as a compactor for a MSS, any distorted response will be fed back to the compactor, and then the distortion of each response will grow as it is multiplied by rising power of 8 or 3. Equation (4.10) explains this fact. This implies that the distortion of the final signature and consequently the aliasing rate will increase. However, in algebraic compactor, as shift is equivalent to multiplication by x^n, n being the size of the analyzer, the distortion of the feedback is not a significant problem. This problem with arithmetic compactor can be alleviated by adjusting the modulus such that the error will increase by power of 1 only as it is shown in equation (4.11).

$$6 \times 3^3 + 4 \times 3^2 + 2 \times 3 + 1 \equiv 6 \times 8^3 + 4 \times 8^2 + 2 \times 8 + 1 \bmod 5 \quad (4.10)$$

$$6 \times 8^3 + 4 \times 8^2 + 2 \times 8 + 1 \equiv 6 \times 1^3 + 4 \times 1^2 + 2 \times (1) + 1 \bmod 7 \quad (4.11)$$

So a compactor scheme that multiplies data by raising power of 1 or -1 will reduce the error.

When 3 is replaced with -1, the resultant circuit is safe from any offset existing in the incoming data or to unidirectional noise that might exist in the testing medium.

The feedback signal is generated based on equation (4.12) in the combinational unit shown by C:

$$c = (\bar{t_2} + r_2)[\bar{t_0} r_0 (\bar{t_1} + r_1) + \bar{t_1} r_1] + \bar{t_2} r_2 + r_3 \qquad (4.12)$$

Figure 4.5 shows the alternative compactor which was proved to have the minimum uncertainty.

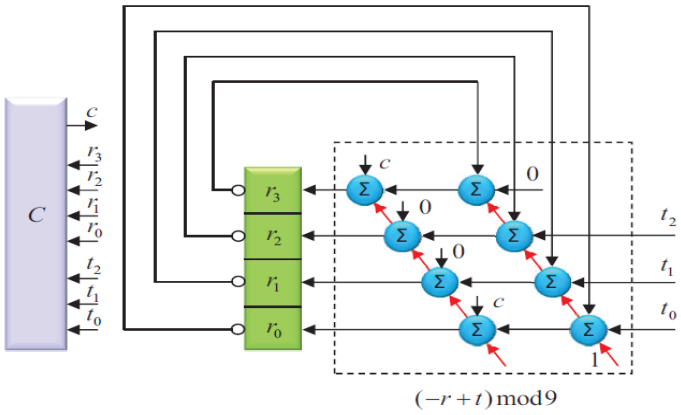


Figure 4-5 An Alternative 3-Input Arithmetic Compactor

4.4 Compaction Process

The process of arithmetic compaction modulo $= 2^n$, where n is the resolution of the MSS, is analyzed here. Let m be the number of distinct values the input signal x^0 can have which is sufficient to detect all faults of MSS. Assume x_i^0 is the value of x^0 at time t_i . The actual output code corresponding to the input value x_i^0 will be shown as expression (4.13):

$$y_i = [x_i^0 + 0.5q] + \delta_i = y_i^0 + \delta_i \quad (4.13)$$

i is equal to 1, …, m, and q is the width of the quantization bin. [x] represents the largest integer number less than x. $y_i^0$is the ideal output code and δ_i is the static error.

Let the lower and upper tolerance bounds be $\check{\delta}_i$and $\hat{\delta}_i$ respectively. So for the fault-free MSS:

$\check{\delta}_i \leq \delta_i \leq \hat{\delta}_i$ or, $\check{y}_i \leq y_i \leq \hat{y}_i$

Where $\check{y}_i = y_i^0 + \check{\delta}_i$ and $\hat{y}_i = y_i^0 + \hat{\delta}_i$

Now the result of adding all the output codes will be expression (4.14):

$$\sum_{i=1}^{m} y_i = \sum_{1}^{m} y_i^0 + \sum_{1}^{m} \delta_i \quad (4.14)$$

Let $Y = \sum_{i=1}^{m} y_i$, $Y^0 = \sum_{1}^{m} y_i^0$, and $\Delta = \sum_{1}^{m} \delta_i$

Using the notations introduced above, it can be obtained that $Y - Y^0 = \Delta$.

Y^0 is calculated using the transfer function the ideal MSS. Assuming symmetrical MSS, i.e. $\check{\delta}_i = \check{\delta}$, $\hat{\delta}_i = \hat{\delta}$ for every i = 1, …, m.

Now performing the modulo addition operation and introducing new notations shown as equation (4.15), and (4.16):

$$\check{\Delta} = (\sum_{i=1}^{m} \hat{\delta}_i) \bmod L \quad (4.15)$$

$$\hat{\Delta} = (\sum_{i=1}^{m} \check{\delta}_i) \bmod L \quad (4.16)$$

It can be shown that the MSS will be faulty if inequality (4.17) is hold [7]:

$$m\hat{\delta} < (Y - Y^0) \bmod L < L - m|\breve{\delta}| \quad (4.17)$$

Otherwise it can be assumed that the MSS is fault-free and $(Y - Y^0)\ mod\ L$ is the correct signature. Assuming that the actual signature is $R = (Y - Y^0) mod\ L$, and then the signature of the fault-free circuit belongs to one of the intervals of $[0, \Delta\hat{}]$, or $[\Delta\hat{}, L - 1]$.

The residue R is computed in the adder that is previously loaded with the two's complement value of Y^0. The two's complement $\overline{Y^0} = -Y^0 \bmod L$.

So equation (4.17) can be written in the following form:

$$m\hat{\delta} < \left(Y + \overline{Y^0}\right) \bmod L < L - m|\breve{\delta}| \quad (4.18)$$

The ratio of the number of all undetectable errors in the output response, to the number of all possible errors of that response is defined as the estimated aliasing rate for an ADC. For an ideal ADC, the output response contains $m \times n$ bits which will be compacted into n bits. Consequently, the number of faulty bit streams that will go undetected by producing the correct signature is $\frac{2^{mn}}{2^n} - 1$. Considering that there are $2^{mn} - 1$ erroneous streams, according to the definition of the aliasing rate that was said before, the aliasing rate for the system tested by the modulo adder method will be approximated by expression (4.19).

$$\frac{2^{mn-n}-1}{2^{mn}-1} \cong 2^{-n} \quad (4.19)$$

For a non-ideal ADC the number of faulty streams going undetected because of producing the correct signature would be expressed as the form (4.20).

$$\frac{2^{mn}}{2^n} - \left(|\breve{\delta}| + \hat{\delta} + 1\right)^m \quad (4.20)$$

Total number of erroneous streams will be obtained as (4.21):

$$2^{mn} - \left(\left|\check{\delta}\right| + \hat{\delta} + 1\right)^{m} \tag{4.21}$$

The aliasing rate P_{ADC}will be obtained from (4.22):

$$P_{ADC} = \frac{2^{mn-n} - (\left|\check{\delta}\right| + \hat{\delta} + 1)^{m}}{2^{mn} - (\left|\check{\delta}\right| + \hat{\delta} + 1)^{m}} \tag{4.22}$$

When the nominal value of the test signal lies at the middle point of the quantization bins, equation (4.19) and (4.22) becomes expression (4.23) and (4.24):

$$m < \left(Y + \overline{Y^{0}}\right) \bmod L < -m \; mod \; L \tag{4.23}$$

$$P_{ADC} = \frac{2^{mn-n} - (3)^{m}}{2^{mn} - (3)^{m}} \tag{4.24}$$

When the stimuli that matches the ideal transitions of the transfer characteristic is fed to the ADC expression (4.23) and (4.24) will be simplified to the form of (4.25) and (4.26):

$$0 < \left(Y + \overline{Y^{0}}\right) \bmod L < -m \; mod \; L \tag{4.25}$$

$$P_{ADC} = \frac{2^{mn-(m+n)}-1}{2^{mn-m}-1} \tag{4.26}$$

If mn is much larger than the value of (m+n), then P_{ADC} can be estimated as 2^{-n}. It is easy to see that the aliasing rate decreases when the resolution of the device increases or, when the size of the modulo adder increases. It should be noted that if the errors at the output of the ADC are equally likely, changing the value of *m* does not have much effect on the aliasing rate.

5. CHAPTER 5
SIMULATION AND ANALYSIS OF THE RESULTS

5.1 Simulation

As it was discussed in the previous chapter, the only modulus that can be used for testing ADC using compaction method are the ones in the form of $2^n \pm 1$. These are the only modulus that will prevent the growth of error caused by noise in the circuit. Simulation was planned based on the analysis that was done for 3-input arithmetic compactor. Although other modulus can be used for checking arithmetic circuits, for testing ADC modulo 7 and 9 were considered for simulation.

Simulation results were generated using ModelSim-Altera and implemented on an FPGA board from Altera. Figure 5.1 shows the result for a random 3-bit input signal B. The content of the signature register is a 3-bit signal Z. The carry-out signal is registered and '000' pattern is applied at the end. The input sequence are, (101), (110), (011), (101) equivalent to 5635 in decimal. Now using arithmetic compaction circuit introduced before, the residue of this number in base 8 modulo 7 would be (5+6+3+5) modulo 7 which is equal to the sequence (101) equal to 5 in decimal.

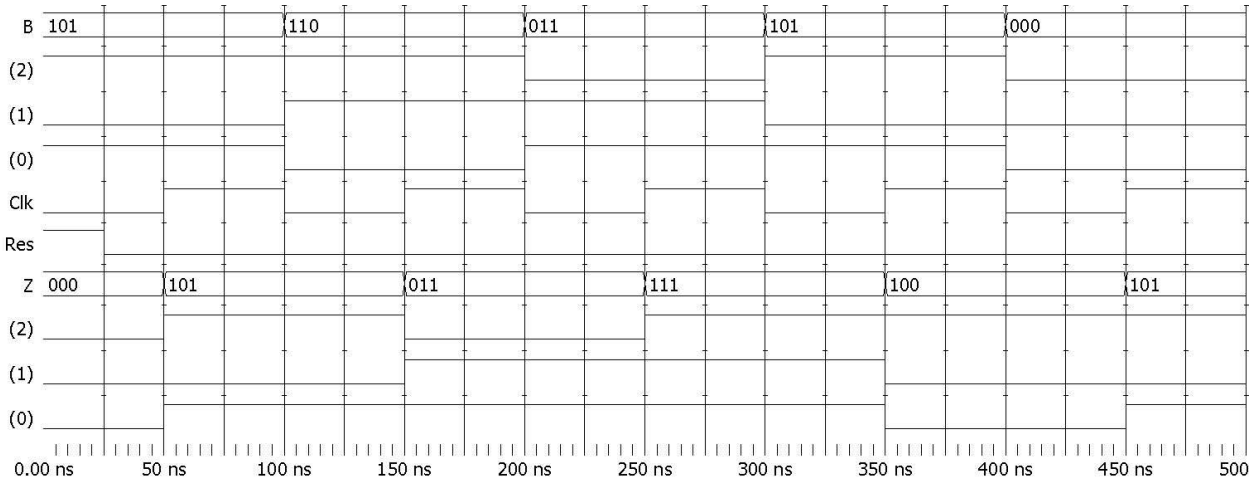


Figure 5-1 RCC with Modulo 5 Adder

Figure 5.2 shows the result of calculating the residue of number 6421_8 modulo 5. The result of this operation is 0. The VHDL code that generated the combinational unit for this circuit based on the input and present state signal (*t* and *r*) can be seen in the appendix section of the report.

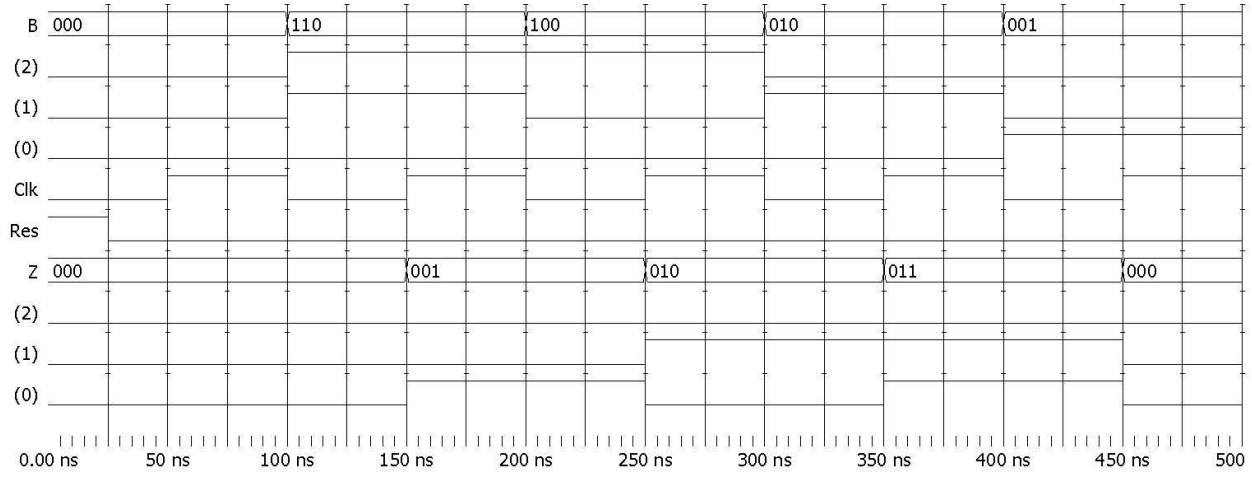


Figure 5-2 Arithmetic Compaction Circuit modulo 5

Figure 5.3 shows the low distortion modulo adder which is when modulus is number 9. In this simulation number 5730_8 modulo 9 was tested. The final value of the register is 8 (or -1). This circuit is the alternative compaction circuit that was introduced in the previous chapter. This scheme is immune to offset of the incoming data as well as noise of the testing environment. The VHDL code that generated the combination unit for this scheme is also provided in the appendix.

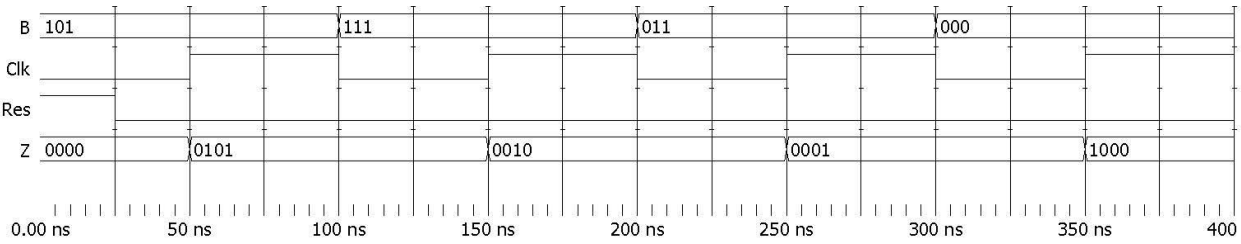


Figure 5-3 Arithmetic Compaction Modulo 9

Now consider a 3-bit ADC under test. It is known that two analog stimuli are enough to detect any fault in the ADC [16]. If these input signals cause the ideal ADC to produce 101 and 110 at the output, then a real non-faulty ADC with ± 1 permissible error will produce codes from the intervals of [100,110] and [101,111].

It is assumed in the simulation that a fault introduces an offset of -001 to the output codes of the ADC. Also, these codes are affected by noise which is uniformly distributed such that the first code is increased by 010 and the second code is decreased by 010. The resultant output code will be 101-001+010=110, and 101-001-010=010. In this case the seed value loaded to the signature register would be: $Y^0 = -(101 + 110) mod\ 111$ =011. Consequently the actual signature will be: $(011 + 100 + 101) mod\ 111 = 101$ and $(011 + 110 + 111) mod\ 111 = 010$. As it can be seen, the actual signature does not drop into these intervals which means that the ADC is faulty. The simulation result can be seen in figure 5.4

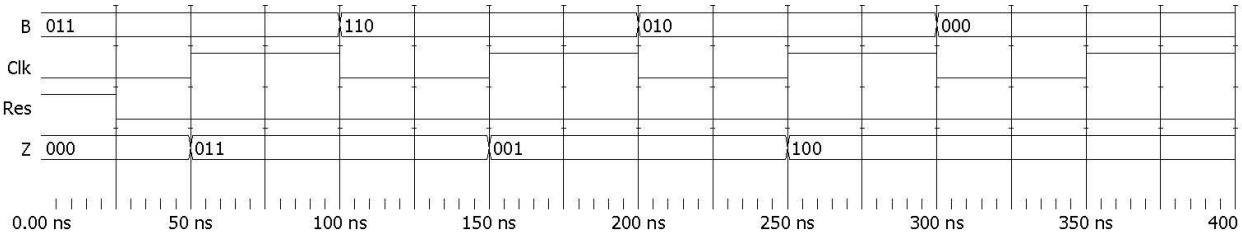


Figure 5-4 Testing an ADC with Offset

Figure 5.5, 5.6, and 5.7 show the RTL form of the simulated circuit of 5.1, 5.2 and 5.3 respectively. As it is clearer in the RTL form of the circuits, the input signal goes into the combinational unit which feeds into the adder. In each clock cycle the register gets updated and the final residue will be saved as the signature of the circuit.

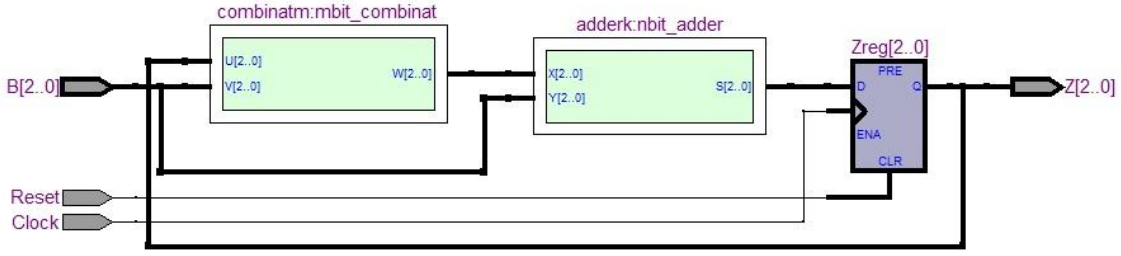


Figure 5-5 RTL form of modulo adder with Combinational Unit

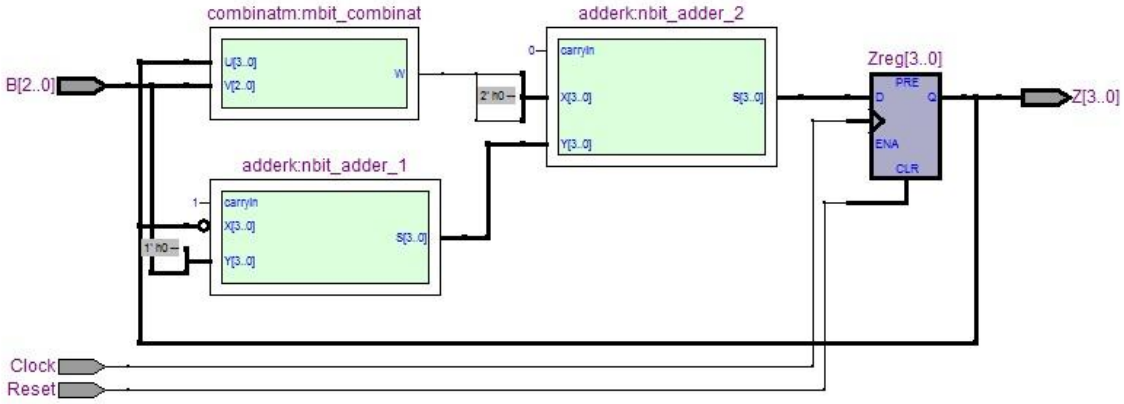


Figure 5-6 Compaction Circuit calculating modulo adder of a ternary number

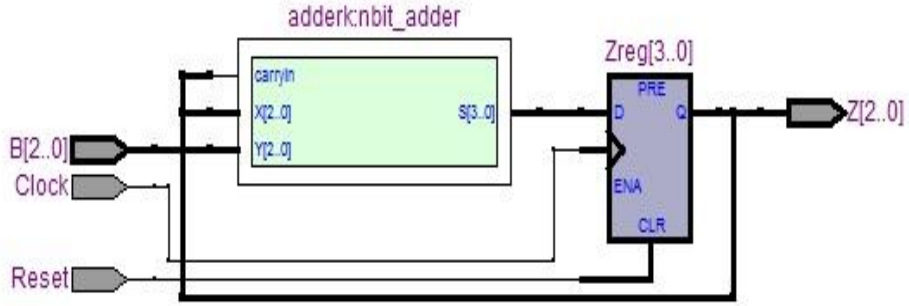


Figure 5-7 An alternative modulo adder

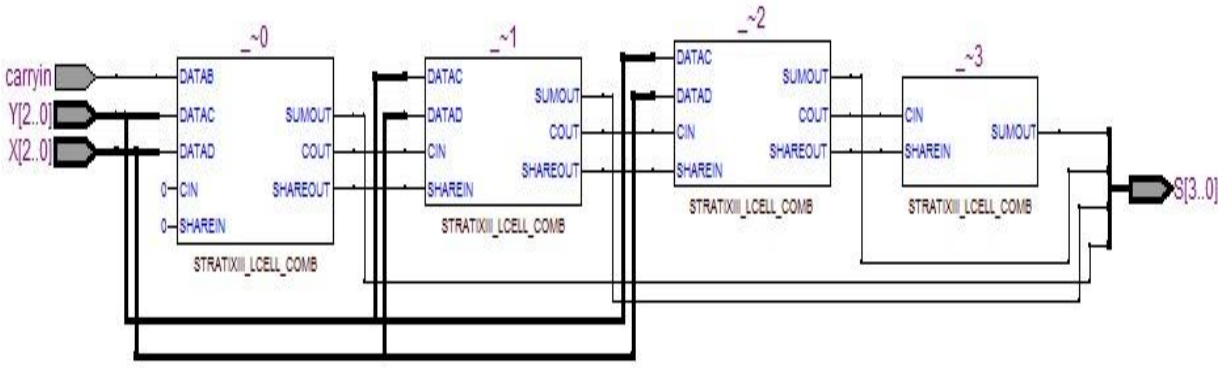


Figure 5-8 Implementation of the Modulo adder

A compaction process was examined for an ADC which is a mixed-signal system. Several compaction schemes were verified with simulation which proved the validity of the theoretical results. Tolerance bounds for a fault-free ADC signature were analytically evaluated in the previous chapter. Modulo 9 and 7 which are in the form of $2^3 \pm 1$ for a 3-bit resolution ADC would detect all single errors [16] with a low aliasing rate.

6. CHAPTER 6
FUTURE WORK AND CONCLUSION

Residue codes which are a class of separable arithmetic codes, are finding more application in digital testing specially self-checking designs. Other types of codes including, parity codes, and Berger codes have been also utilized in previous literature for testing arithmetic operations. High implementation cost related to residue codes has been always a concern. However, because of the simplicity and separability of this type of codes, and the fact that they are closed under operations such as addition and multiplication they are still popular. In particular it has been shown that in case of multipliers it is better to use residue codes for error protection. This project has developed the theoretical knowledge needed to achieve cost efficient, reliable and fault secure circuits based on residue codes. More importantly this work intends to be the platform for studying the effectiveness of arithmetic and algebraic compaction method for testing mixed signal devices such as analog-to-digital converters.

Application of residue testing method, for on-line error detecting in an ADC was introduced in this project. Different compaction schemes that can be used for testing digital as well as mixed-signal system were presented. According to this testing method, a faulty circuit can be identified if the result of the output signature does not exist in the predefined permissible interval. Compactor schemes with different modulo adders where analyzed and simulated to find an arithmetic compacting circuit with the minimal uncertainty. The circuits were simulated for a 3 bit data in ModelSIM and the scheme of the RTL form of the circuits were presented in the report. The tolerance bounds for the signature and the aliasing rate of the circuit were obtained. It was shown that the aliasing rate decreases if the resolution of the ADC increases. If the ADC can

perform a direct-conversion, the binary counter of this ADC can be used as a signature compactor. Such an ADC in the testing mode is going to reset after a series of conversions is performed for the entire sequence of test stimuli.

Modulo of the form $2^n \pm 1$ are the most important ones for they are considered cheap and easy to implement. Similarly compaction modulo of the form $2^n - 1$ will detect all the single errors that would affect all bits except the least significant bits of the word [16].

Future work to complement this work would be to implement the compaction circuits to verify a more accurate evaluation of time delay, overhead, and fault secure property of the circuit. Ultimate goal would be to optimize RCC design so that it can be used for efficient modulo $2^n \pm 1$ multiplier and testing arithmetic operations.

7.APPENDIX

Part of the VHDL code that generated the combinational unit of figure 5.2, and 5.3 in chapter 5 are shown here. Code of the modulo adder circuit is also shown here.

```
ENTITY combinatm IS
   GENERIC ( m : INTEGER := 3 ) ;
   PORT ( U, V : IN STD_LOGIC_VECTOR(m-1 DOWNTO 0) ;
             W : OUT STD_LOGIC_VECTOR(m-1 DOWNTO 0) ) ;
END combinatm ;

ARCHITECTURE Behavior OF combinatm IS
BEGIN
   W(2) <= (not U(1) and U(0) and V(1) and not V(0)) or
           (U(0) and (V(1) xor V(2))) or
           (U(1) and ((not U(0) and V(2)) or (U(0) and not V(1)))) or
           (U(2) and ((V(1) and V(0)) or V(2))) ;
   W(1) <= (U(2) and not V(2) and (not V(1) or not V(0))) or
           (U(0) and ((U(1) and V(2)) or (V(1) and not V(0)) or (not U(1) and not V(1)) or (V(0) and not V(2)))) or
           (not U(2) and not U(1) and not U(0) and V(2) and (V(0) or V(1))) ;
   W(0) <= ((V(1) or V(0)) and ((U(1) and not V(2)) or (not U(1) and not U(0) and V(2)))) or
           (U(0) and not V(1) and not (U(1) xor V(2))) or
           (V(1) and V(0) and ((not U(1) and V(2)) or U(2))) or
           (U(1) and not U(0) and not V(2)) or
           (U(2) and V(2)) ;
END Behavior ;
```

```
ENTITY combinatm IS
   GENERIC ( m : INTEGER := 3 ) ;
   PORT    ( U : IN STD_LOGIC_VECTOR(m DOWNTO 0) ;
             V : IN STD_LOGIC_VECTOR(m-1 DOWNTO 0) ;
             W : OUT STD_LOGIC ) ;
END combinatm ;

ARCHITECTURE Behavior OF combinatm IS
BEGIN
   W <= ((not V(2) or U(2)) and ((not V(0) and U(0) and (not V(1) or U(1))) or (not V(1) and U(1)))) or
        (not V(2) and U(2)) or U(3) ;
END Behavior ;
```

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

-- Top-level entity
ENTITY RCC IS
   GENERIC ( n          : INTEGER := 3 ) ;
   PORT ( B             : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0) ;
          Clock, Reset  : IN STD_LOGIC ;
          Z             : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)) ;
END RCC ;

ARCHITECTURE Behavior OF RCC IS
   SIGNAL M, Zreg     : STD_LOGIC_VECTOR(n DOWNTO 0) ;
   COMPONENT adderk
      GENERIC ( k     : INTEGER := 3 ) ;
      PORT ( carryin : IN STD_LOGIC ;
                X, Y : IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
                   S : OUT STD_LOGIC_VECTOR(k DOWNTO 0) ) ;
   END COMPONENT ;
BEGIN
   PROCESS ( Reset, Clock )
   BEGIN
      IF Reset = '1' THEN
         Zreg <= (OTHERS => '0');
      ELSIF Clock'EVENT AND Clock = '1' THEN
         Zreg <= M;
      END IF ;
   END PROCESS ;
   nbit_adder: adderk
      GENERIC MAP ( k => n )
      PORT MAP ( Zreg(n), Zreg(n-1 DOWNTO 0), B, M ) ;
   Z <= Zreg(n-1 DOWNTO 0) ;
END Behavior;
```

```
ENTITY adderk IS
   GENERIC ( k     : INTEGER := 3 ) ;
   PORT ( carryin : IN STD_LOGIC ;
             X, Y : IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
                S : OUT STD_LOGIC_VECTOR(k DOWNTO 0) ) ;
END adderk ;

ARCHITECTURE Behavior OF adderk IS
BEGIN
   S <= ( '0' & X) + ( '0' & Y) + carryin ;
END Behavior ;
```

8. REFERENCES

[1] I.L Sayers, D.J Kinniment, E.G Chester, "Design of a reliable and self-testing VLSI datapath using residue coding technique ," *IEE proceedings I Solid-State and Electron Devices,*Vol. 133, pp.129-140, 1986.

[2] U. Sparmann, S.M. Reddy, "On the effectiveness of residue code checking for parallel two's complement multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol. 4, pp.227-239, June 1996.

[3] M. Mahoney, *DSP-Based Testing of Analog and Mixed-Signal Circuits.* Los Alamitos: IEEE Computer Society Press, 1987.

[4] C. Stroud, J. Morton, T. Islam, H. Assaly, "A mixed-signal built-in self-test approch for analog circuits," *Soutwest Symposium on Mixed-Signal Design*, pp.196-201,2003.

[5] V. Geurkov, L. Kirischian, "A Concurrent Testing Technique for Analog-to-Digital Converter ," *IEEE Mixed-Signals, Sensors and Systems Test Workshop*,pp.133-136,2011.

[6] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., second edition, 1996.

[7] M. Abramovici, M. Breuer, A. Friedman. *Digital Systems Tesing and Testable Design* . Wiley-IEEE Press, NewYork, 1994.

[8] W. Hong, R. Modugu, M. Choi, "Efficient Online Self-Checking Modulo 2^n-1 Multiplier Design," *IEEE Transactions on Computer,* vol. 60, pp.1354-1365,2011

[9] S. Bayat-Sarmadi, "*Concurrent Error Detection in Finitr Field Arithmetic Operations,"* University of Waterloo, 2007

[10] A. I. Noufal and M. Nicolaidis, "A CAD framework for generating self-checking multipliers based on residue codes," *IEEE Conference on Design, Automation and Test*, pp. 122-129, 1999

[11] P.E. Beckmann and B.R. Musicus, "Fast Fault-Tolerant Digital Convolution Using a Polynomial Residue Number Systems," *IEEE Transaction on Signal Processing*, vol. 41, no. 7, pp. 2300-2313,1993

[12] M.B. Sullivan, "Aplication of Residue Codes for Error Detection in Modern Computers," Texas, 2010

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

[14] J. Cavanagh, *Computer Arithmetic and Verilog HDL Fundamental*, CRC Press, Taylor & Francis Group, 2010

[15] U. Sparmann, S.M. Reddy, "On the effectiveness of residue code checking for parallel two's complement multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 4, pp.227-239, June 1996

[16] V. Geurkov, L. Kirischian, "A Concurrent Testing Technique for Analog-to-Digital Converter ," *IEEE Mixed-Signals, Sensors and Systems Test Workshop*, 2011

[17] S. Das, M. Sudarma, M. Assaf, W. Jone, K. Chakrabarty, and M. Sahinoglu, "Parity Bit Signature in Response Data compaction and Built-In Self-Testing of VLSI Circuits With Nonexhaustive Test Sets*,"IEEE Transaction on Instrumentation and Measurment, Vol. 52*,pp.1363-1380, October 2003

[18] W. Peterson and E.Weldon, *Error Correcting Codes*, Cambridge, MA: The MIT Press, 1972

[19] S. Z. Hassan, D.J. Lu, and E.J. McCluskey, "Parallel Signature Analyzers," *26th IEEE Computer Society Intn'l. Conf.,* COMPCON, Spring 1983, pp.440-445, 1983

[20] M.Mahony, *DSB-Based Testing of Analog and Mixed-Signal Circuits.* Los Alamitos: IEEE Computer Society Press, 1987