MIXED SIGNAL TESTING SYSTEM TECHNIQUE WITH ALGEBRAIC SIGNATURE ANALYZER WITHOUT CARRY PROPAGATION

by
Mohammed Faruque Ahmed
B.Sc, Ahsanullah University of Science & Technology, Bangladesh, Dhaka, 2007

A project

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, 2017

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A PROJECT

I hereby declare that I am the sole author of this Project. This is a true copy of the Project, including any required final revisions.

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

I further authorize Ryerson University to reproduce this Project 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.

I understand that my Project may be made electronically available to the public.

MIXED SIGNAL TESTING SYSTEM TECHNIQUE WITH ALGEBRAIC SIGNATURE ANALYZER WITHOUT CARRY PROPAGATION

Mohammed Faruque Ahmed
Master of Engineering
Electrical and Computer Engineering
Ryerson University, Toronto, 2017

Abstract

Signature Analyzer is an analyzer which is widely used for mixed-signal system testing. But its hardware has high complexity in implementation as the application technique is a system with rules of an arithmetic finite field with arbitrary radix. It's a challenging task. To avoid this complexity here the project is made based on Algebraic Signature Analyzer that can be used for mixed signal testing and the analyzer doesn't contain carry propagation circuitry. It improves performance and fault tolerance. This technique is simple and applicable to systems of any size or radix. The hardware complexity is very low compared to the conventional one and can be used in arithmetic/ algebraic cryptography as well as coding.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

INTRODUCTION

Mixed signal system consists of both analog and digital circuit but the analysis method is only applicable to the subset of these systems that have digital outputs. Signature analysis can be employed to embedded into the system under test solution or can be used for the external test.

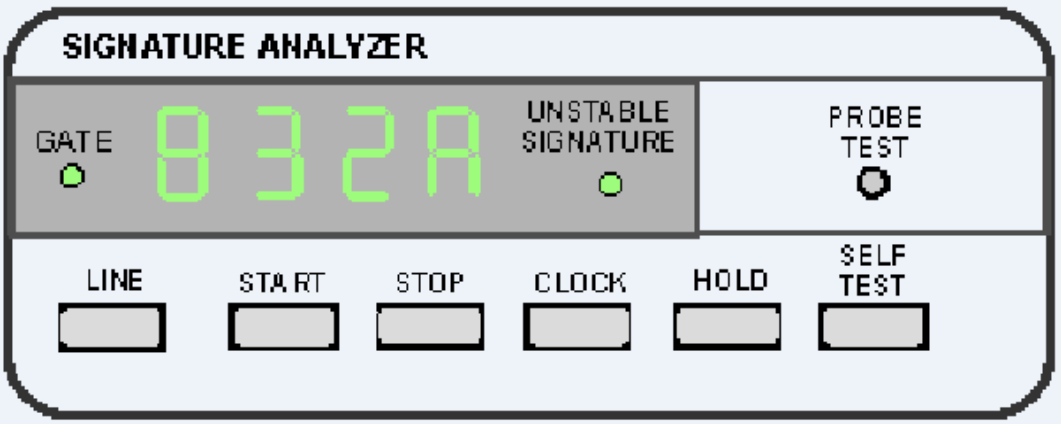

Figure 1: Signature Analyzer

In the case of implementation, a reference signature will be used which nothing but a fault-free circuit. On the other hand, a circuit under test (CUT) of mixed signal nature will be fed by test stimuli and output will be compacted by the algebraic signature analyzer (ASA).

The signature analyzer formulates an ideal form of a test instrument for analyzing digital or logic patterns in a circuit in some conditions. It is often ideal for field repair and applications where it can perceive logic patterns in a circuit under given or fix conditions, in so doing enabling detection of correct or incorrect operation of a circuit or board.

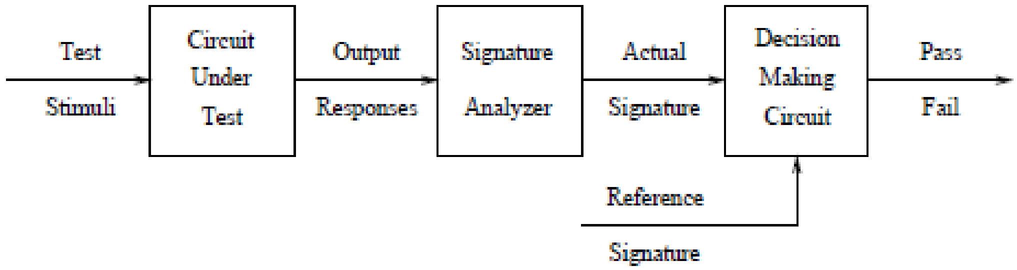

Figure 2: Built-in signature analysis of a circuit under test

The signature analyzer which is algebraic is designed on the basic concept of a polynomial division circuit. It's which is shown in figure 3 This circuit divides the incoming sequence of non-binary symbols, $a_{m-1}, \ldots, a_1, a_0$ and this non-binary symbols are treated as a polynomial:

$$a(y)=a_{m-1}y^{m-1}+\ldots\ldots +a_1y+ a_0 \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \; (1)$$

by the polynomial

$$p(y)= P_t\, y^t +\ldots\ldots +P_1y+P_0 \qquad t<<m \; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \; (2)$$

the reminder

$$s(y)=s_{t-1}y^{t-1}+ \ldots. +s_1y+s_0 \; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \; (3)$$

Normally A microprocessor board is use for

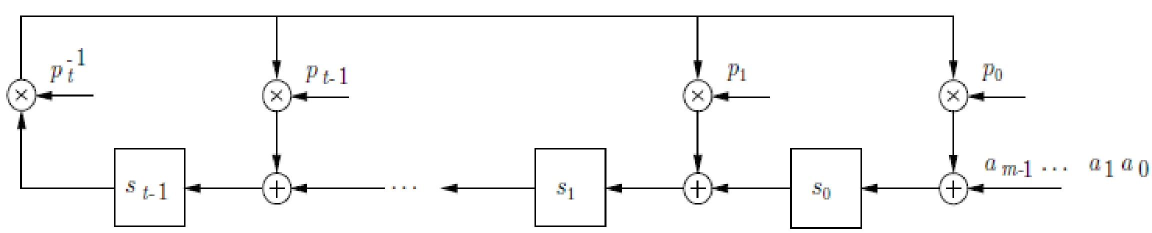

Figure: 3 A t-Stage polynomial division Circuit

checking data in signature analyzer on given nodes within a logic system testing. An operational scenario is set up, e.g. a test mode and the data on various nodes are monitored. The signature analyzer transforms the serial data into a hexadecimal data pattern - this is the equivalent signature. Typically, this signature has digits depending on different signature analyzer's different lengths.

Chapter 2

Conventional Signature Analyzer

The basic signature analyzer takes in the input from the node under test and uses a clock from the system for synchronization. Start and stop pulses are seized to start and end the sample.

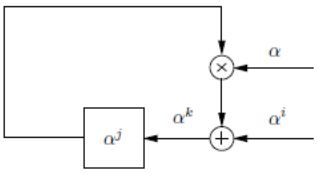

Figure 4: A Symbolic Presentation of a one-stage arithmetic.

The pulses from the node under test are then passed into a shift register to provide the hexadecimal equivalent of the waveform. The multiple input signature register compression (MISR) is the prime technique used in the signature analysis. The outputs of the circuit under test (CUT) are connected to the inputs of the MISR while the test patterns are applied to the CUT. The final contents of the MISR are compared to that expected for a fault-free circuit to determine whether the CUT is faulty.

Before starting the implementation and design, some theoretical knowledge should be Marge on the process to handle some typical factor. Aliasing Probability is one of it. More input to the analyzer is formulated an expression for estimating the aliasing probability. Multiple input use provides a more accurate error model by relating the analysis q- ary code where q= 2^m ; m = number of output for the circuit under test (CUT).

Let *C* be an n-tuple $(C_{n-1} * C_{n-2} * \ldots C_0)$ where $C_i \in GF(q)$.

Let $C(x) = C_{n-1} X^{n-1} + \ldots + . . + C_1 X + C_0$. be the polynomial representation of the n-tuple.

The vector and polynomial representations shall be used interchangeably. All polynomial representations and operations will be assumed to be over ***GF(q)*** where $\boldsymbol{q} = 2^m$.

Thus, all additions and multiplications in this piece will be assumed to be over $GF(2^m)$. The terms of the polynomials can be characterized as only positive terms.

Definition I: The generator polynomial ***g(x)*** of a code ***C*** is that polynomial ***g(x)*** which divides every code word polynomial in ***C.*** The degree of ***g(x)*** is equal to ***n*** - ***k*** where ***n*** is the length of the code and *k* is the number of information symbols. [2]

Two key observations should be made here. First, when ***g(x)*** divides $\mathbf{x^n}$ - 1, only then does the code become a cyclic code of length ***n.*** On the other hand, when ***g(x)*** does not divide $\boldsymbol{x^n}$ - 1, then the code is not cyclic. The results derived here are applicable to cyclic and noncyclic codes. [2]

In the following, the Galois field elements **0** = (0, 0) and **1** = (0, **1)** are denoted by boldface to distinguish from the binary 0, 1.

The uniqueness of our formulation is that it not only allows a uniform model for analysis of both LFSR and MISR techniques but also provides for the development of new signature techniques. Using this a new compression scheme for multiple output circuits are developed. This new scheme, referred to here as multi-input LFSR (MLFSR), has the potential to achieve lower aliasing than other existing schemes with analogous hardware complexity.

New error models are discussed for multioutput circuits. It is shown how these can utilize circuit-specific information to obtain realistic error models. This paper presents several new aliasing probability results, using the coding theory framework. Specifically, exact closed-form expressions of aliasing probability for both LFSR and MISR are presented for certain test lengths. To the best of our knowledge, an exact closed-form countenance for MISR aliasing probability

under independent error model had not been previously reported. Also presented are algorithms to compute aliasing probabilities of LFSR's and MISR's for any arbitrary test length.

The aliasing probability of MLFSR, the new compression scheme proposed here, is studied. It is shown that the MLFSR achieves lower aliasing compared to other schemes of comparable hardware complexity such as multiple MISR.

Finally, a theoretical question is that of significant importance is whether not zero aliasing compression is possible. We show that it is not only possible but there exist design techniques which achieve aliasing-free compression with compression efficiency (1 - the length of the signature length of test response) no less than half. Next, we present a result which states that any desired compression efficiency can also be attained, asymptotically.

This result is of theoretical significance because previously, it was commonly believed that zero aliasing is impossible.

There has been a significant volume of research on the problem of test data compaction for digital circuits using special hardware that is usually implemented in a built-in self-test (BIST) environment [6], [8]. Several compaction schemes have been developed which are based on transition counting [9], checksums [10], syndrome testing [11] and single- and multiple-input linear feedback shift registers (LFSR's) [7]. Recently, Rajski and Tyszer [12] have analyzed the properties of digital integrators for test response compaction for digital circuits.

As opposed to LFSR's, the scheme using integrators introduces very small hardware overhead. Although there has been a considerable amount of progress in test response compaction for digital circuits, there has not been any past work in this direction for analog/mixed-signal circuits, to the best of our knowledge.

The integrator for computing the signature is shown in Fig. 4. The input to the integrator consists of a sequence x of sampled data words of length N: An integral (summation) of x over N samples

is given by S(x) which consists of an integer part I(x); and a fractional part R(x) represented by m and n bits, respectively.

For simplicity, we assume that the register in Fig. 4 represents numerical data normalized to lie between 0 and 1 (including 0 but not including 1).

The data is represented as fixed-point

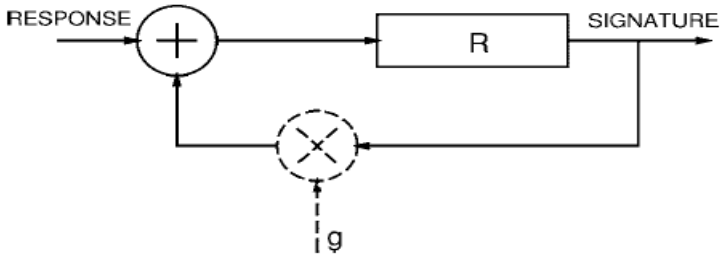

Figure 5: Digital Integrator

fractions. An overflow occurs if the value in the register exceeds its full-scale value (greater than or equal to 1). The remaining value in the register, R(x) constitutes the signature. If desired, the signature analyzer can be designed to handle numbers with integer and fractional parts, rather than just the latter as considered in this analysis. Let the WORDSIZE be given by n: As an example, the data word .111 000 00 representing the number 0.875 is specified by WORDSIZE = 8: The signature is given by the bit values stored in the register and consists of the decimal fraction of the integral.

Let the maximum tolerance of the response signal at any given time-point be denoted by β and let N be the total number of samples over which the signal is integrated. Then the tolerance for the good signature (for the nominal response and those within tolerance) is bounded by € = N X β; As discussed in the previous section, β must lie within the range of values that can be represented by the register. This condition is equivalent to €<1: However, if the nominal signature is such that by

adding the tolerance margin €; the register overflows, the remaining signature would no longer be good. This will cause a response within tolerance to be incorrectly rejected as bad (false reject). On the other hand, if a faulty response maps to a signature within a tolerance of the nominal (good signature) it results in incorrectly passing a faulty response (*aliasing*). Both phenomena are undesirable and the signature analyzer should be designed such that it minimizes the probability of occurrence of these incorrect judgments. [5] a new framework is presented for shift register-based test response compressors.

Next, consider the signature analyzer at the output of the CUT. Aliasing occurs when the error

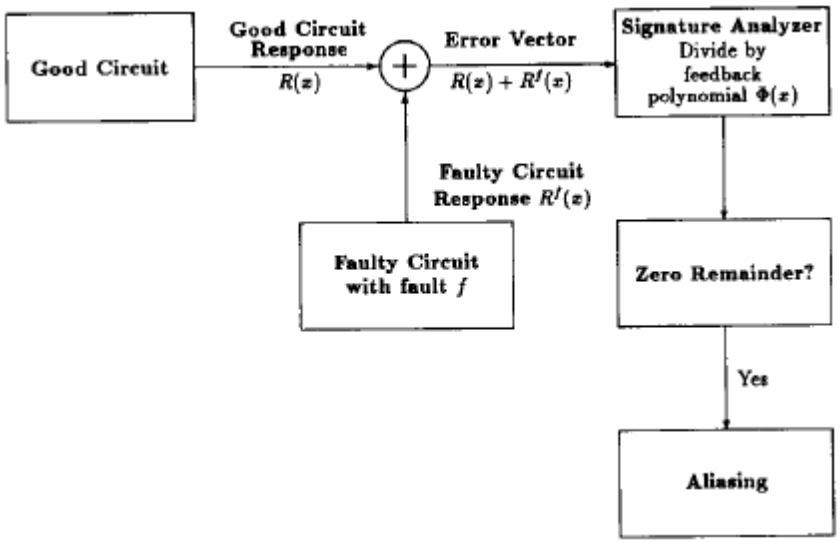

Figure 6: Aliasing in Signature Analysis

vector, which is defined as the sum of the faulty circuit response and good circuit response, is divisible by the feedback polynomial, as shown in Fig. 5.

Using the communication channel analogy, one can state that aliasing occurs precisely when the error vector corresponds to a code vector in the code generated by the feedback polynomial of the signature analyzer.

Hence, the probability of aliasing in GLFSR is precisely equal to the probability of undetected error in the following equivalent communication scheme. Let the good circuit response be transmitted over a noisy channel where the characteristics of this noisy channel are defined by the error model used for estimating the aliasing probability. (For example, the well-known independent error model [9], [10] will correspond to the binary symmetric channel model [12] used frequently in communication theory.) The receiver then divides this received vector by g(x), where g(x) = ø(x), the feedback polynomial. If the resulting remainder (syndrome) equal to the remainder (syndrome) obtained by dividing the good circuit response with ø(x) then no error is detected. This happens only when the added noise in the channel corresponds to a code vector in the code generated by g(x) = Ø(x).

The following is a direct consequence of the above observations.

Let AC (aliasing code) represent the code *C* generated by g(x) = Ø(x) (the feedback polynomial of the GLFSR) of length *n* = 1, where I is the length of the test response compressed by the GLFSR (β, *m).*

An error polynomial E(z) causes aliasing in GLFSR (β, m) if E(x) belongs to the code AC defined above. The following examples motivate the results subsequently presented.

Let the test response from a single output circuit be compressed into a **3-**bit signature, using a GLFSR (1, 3) shown in Fig. 6, with primitive feedback polynomial over GF (2):

$$g(x) = x^3 + x + 1.$$

This corresponds to a three-stage simple LFSR.

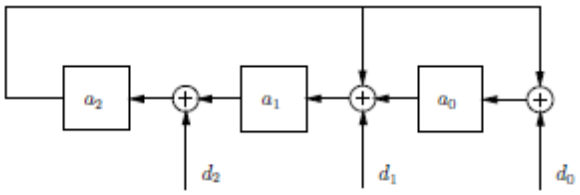

Figure 7: A Logic Level Presentation of the algebraic 3-input Signature Analyzer

In this case, we assume there are L=7 tests applied to the CUT. Since m =3 one has the degree of Ø(x), L = (2^m - 1). Thus, the code words in the code AC generated by Ø(x) of length **7** constitute the cyclic Hamming code from Corollary 3**.** These are given

AC ={0, x^3+ x+ l, x^4+ x^2+x, x^4+ x^3+x^2 +1, x^5+ x^2+x+1, x^5+ x^3+x^2, x^5+ x^4+1, x^5+ x^4+x^3+x, x^6+ x^2+1, x^6+ x^3+x^2+x, x^6+ x^4+x+1, x^6+ x^4+x^3, x^6+ x^5+x, x^6+ x^5+x^3+1, x^6+ x^5+x^3+x^2+ x+1}.

In Table I(a), (b), and (c), we illustrate the states of the LFSR in Fig. 6, in response to input sequences 0101110, 0100101, and 0100100, respectively. Let the first sequence, 0101110, corresponding to the good circuit response, and the other two correspond to faulty circuit responses. It may be seen that the first faulty circuit response 0100101 will cause aliasing since it produces the same signature, 010, as the good circuit response. This is because 0001011, the bit by bit EXOR sum of the good circuit response and this faulty circuit response, is a code word in the code AC.

Now consider the second faulty response 0100100. The signature in response to this sequence is 110 which is different from 010, the good circuit signature. This is because 0001010, the bit-by-bit EXOR sum of 0101110 and 0100100, is not a code word in the code AC. Thus, the faulty response (c) will be detectable, whereas the response (b) will cause aliasing.

Therefore, a fault f can cause aliasing if and only if the tests for f are such that the error polynomial is a code word in the above code. The number of 1's in the code words has important implications. For example, consider a test sequence T = { t_5,t_4,t_3,t_2,t_1,t_0}. It may be noted that if t; is a test for the fault f, then the error vector (error polynomial) corresponding to f will have a 1 in the i-th position (the term x^i).

For example, if a fault f is tested by three tests t_3, t_1, and t_o, then f will be aliased because the corresponding error vector $x^3+ x+ 1$ is a code word in the above code. However, any fault that has either two or five test will not cause any aliasing, since there is no code word with two or five 1's in it. It may also be noted that all the nonzero codewords in the above code have at least three 1's. Therefore, any fault that is detected by only one or two tests will not cause aliasing. A fault to cause aliasing should be detectable by at least three tests.

Table: 1 a) Good Circuit Response

Test	Shift	R(X)	Register value	
			Stage 0	Stage 1
			$0 = \binom{0}{0}$	$0= \binom{0}{0}$
t_4	1	$\alpha = \binom{1}{0}$	$\alpha = \binom{1}{0}$	$0 = \binom{0}{0}$
t_3	2	$\beta = \binom{1}{1}$	$\beta = \binom{1}{0}$	$\alpha = \binom{1}{0}$
t_2	3	$0 = \binom{0}{0}$	$\beta = \binom{1}{0}$	$1 = \binom{0}{1}$
t_1	4	$1 = \binom{0}{1}$	$\beta = \binom{1}{0}$	$\alpha = \binom{1}{0}$
t_0	5	$\alpha = \binom{1}{0}$	$1 = \binom{1}{0}$	$1 = \binom{0}{1}$

Table: 1 b) Faulty Circuit Response

Test	Shift	R(X)	Register value	
			Stage 0	Stage 1
			$0 = \binom{0}{0}$	$0= \binom{0}{0}$
t_4	1	$\alpha = \binom{1}{0}$	$\alpha = \binom{1}{0}$	$0 = \binom{0}{0}$
t_3	2	$\alpha = \binom{1}{0}$	$\alpha = \binom{1}{0}$	$\alpha = \binom{1}{0}$
t_2	3	$1 = \binom{0}{1}$	$\alpha = \binom{1}{0}$	$0 = \binom{0}{0}$
t_1	4	$\beta = \binom{1}{1}$	$\beta = \binom{1}{1}$	$\alpha = \binom{1}{0}$
t_0	5	$\alpha = \binom{1}{0}$	$1 = \binom{0}{1}$	$1 = \binom{0}{1}$

Table: 1 c) Faulty Circuit Response

Test	Shift	R(X)	Register value	
			Stage 0	Stage 1
			$0 = \binom{0}{0}$	$0= \binom{0}{0}$
t_4	1	$\alpha = \binom{1}{0}$	$\alpha = \binom{1}{0}$	$1 = \binom{0}{1}$
t_3	2	$\alpha = \binom{1}{0}$	$\alpha= \binom{1}{0}$	$\alpha = \binom{1}{0}$
t_2	3	$0 = \binom{0}{0}$	$\beta = \binom{1}{1}$	$0 = \binom{0}{0}$
t_1	4	$0 = \binom{0}{0}$	$0 = \binom{0}{0}$	$\beta = \binom{1}{1}$
t_0	5	$\beta = \binom{1}{1}$	$\alpha = \binom{1}{0}$	$\beta = \binom{1}{1}$

A multiple-input signature analyzer normally contains only one stage. It is presented in Figure 8 where α is a primitive element of the field $GF(2^n)$, i.e. a root of a primitive polynomial

$$g(x) = g^{n-1}x^{n-1} + \ldots + g^1x + g^0 \ldots\ldots\ldots\ldots (4)$$

All elements of the field can be represented by a power of α. Assume α^i be the incoming digit and α^j be the content of the analyzer. Then, each operational cycle of the analyzer is described by the following expression:

$$\alpha^j \alpha \oplus \alpha^i = \alpha^k \ldots\ldots\ldots\ldots (5)$$

Without a loss of generality, we will consider a 3-bit signature register (n = 3), with α being a primitive element of GF (2^3), in particular, a root of a primitive polynomial is

$$g(x) = x^3 + x + 1.$$

Then, a symbolic scheme of Figure 4 will transfer to the logic level circuit of Figure 8, where

$$\alpha^l = a_2^{(l)}x^2 + a_1^{(l)}x + a_0^{(l)}, \quad a_i^{(l)} \in \{0,1\}, \ldots\ldots\ldots\ldots (6)$$

$$0 \le i \le 2, \qquad 0 \le l \le 6$$

The above expression indicates the relationship between the power and vector representations of a field element.

For example, If the preliminary “cleared” analyzer receives, the following sequence of 3-bit output responses from a digital CUT, α^5 , α^6 ,α^4 ,α^4 ,α^2 ,α^1 ,α^0 then after the 6-th shift its content will become:

$$(((((0.\alpha + \alpha^5)\alpha + \alpha^6)\alpha + \alpha^4)\alpha + \alpha^2)\alpha + \alpha^1)\alpha + \alpha^0 = \alpha \ldots\ldots\ldots\ldots (7)$$

If in the above sequence of output responses, the least significant bit in the first response changes from 1 to 0 (i.e. the vector 111 changes to 110, or power α^5 changes to α^4, then the actual signature will change from 010 to 101or from α to α^6 in power form.

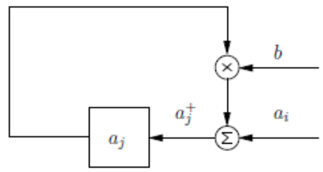

Figure 8: A symbolic presentation of a one-stage arithmetic excluding adder

In the known methods, output responses of mixed-signal circuits are compacted by a circuit referred to as a modulo adder. It should be noted that a modulo adder is a special case of a residue computing circuit. A residue computing circuit is represented in Figure 9. Here a_j is the current content of the register, a_i is the incoming (arithmetic) symbol and b are the bases of the system. This circuit divides the incoming sequence of symbols, a_{m-1} ……. a_1, a_0, treated as a number:

$$a = a_{m-1}b^{m-1} + \ldots\ldots + a_1b + a_0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (8)$$

by the modulus

$$p = p_{t-1}\, b^{m-1} + \ldots\ldots + p_1b + p_0 \quad ; t << m \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (9)$$

we consider a single stage device, i.e. t = 1; $p = p_0 < b = 2^n$, where n is the number of bits occupied by the symbol. The residue, s_0, constitutes a signature.

An operational cycle of the circuit in Figure 9 can be described by the expression:

$$a_jb + a_i = a^+_j \ (\mathrm{mod}\ p) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (10)$$

Although the circuits of Figures 4 and look similar, their implementation is quite different. In general case, the designing procedure for the arithmetic circuits is more complicated and their hardware complexity is greater.

Figure 10 represents the circuit that computes a modulo 5 residues of the incoming sequence of 3-bit symbols treated as an octal number.

$$
\begin{aligned}
c_2 &= a_0^j \overline{a_1^j} a_0^i a_1^i a_2^i + a_1^j (a_0^j a_1^i \oplus a_2^i + a_0^j a_0^i \overline{a_2^i}) \\
c_1 &= a_2^j \overline{a_2^i} (\overline{a_0^i} + \overline{a_1^i}) + \overline{a_2^j} a_2^i (\overline{a_0^j} a_0^i + \overline{a_1^j} a_1^i) + \\
&\quad a_1^j (\overline{a_0^j} + \overline{a_2^i}) + a_0^j (a_1^i \oplus a_2^i) \\
c_0 &= a_0^j a_1^i \overline{(a_1^j \oplus a_2^i)} + a_1^j \overline{a_2^i} (\overline{a_0^j} + \overline{a_0^i a_1^i}) + \\
&\quad a_2^j + \overline{a_1^j} a_2^i (\overline{a_0^j} a_1^i + a_0^i \overline{a_1^i} + a_0^j \overline{a_0^i})
\end{aligned}
$$

Here a_i is the incoming octal digit and C is a combinational circuit which generates the following next state signals:

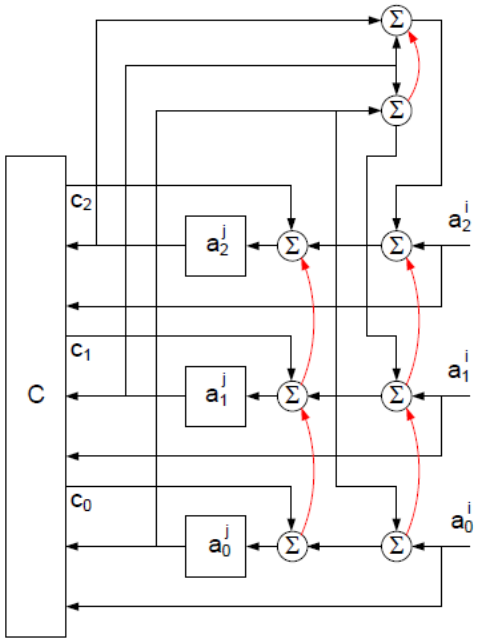

Figure 9: A 3-input arithmetic compactor

The shift of this circuit implements the operation $a_j x^8 + a_i \pmod 5$. For high hardware complexity, the arithmetic compactor contains carry propagating circuitry. It's shown in red color in Figure 9. This circuitry delays the operation and aggravates the effect of a single fault.

In figure 9, it designed an algebraic circuit that can be employed for mixed-signal data compaction and it does not contain carry propagating circuitry.

Chapter 3

NOVEL APPROACH

When the polynomial connected with the reference signature then it can be considered as a code word of the code whose minimal distance is defined by the g(x). Here this distance is called the Hamming distance. This distance characterizes algebraic error-detecting properties of the code and It is not convenient for arithmetic errors that occur in mixed-signal systems.

A small permissible deviation of the data to be compacted causes the reference signature to span the entire space. The circuit which can be called decision making is in Figure 2. This circuit must be able to compare the actual signature with the entire set of possible reference signatures, under these conditions. Analyzer's complexity increases for this. If try to decrease the complexity, an arithmetic SA treats the sequence of output responses from a mixed-signal circuit as a number.

In conjunction with the reference residue, this is considered as a code word of an arithmetic error-control code. The properties of this code depending on the arithmetic minimal distance. The arithmetic minimal distance depends on the modulus p. The arithmetic residue calculating analyzer does not search the entire space. For taking a decision, it employs a window comparator. This simplifies the circuitry but the hardware complexity of the arithmetic SA can still be quite high. The distance between two vectors will be calculated as the arithmetic difference between the corresponding exponents. The distance between the signatures 010 and 101 will be 5 because the exponents of powers α^6 and α differ by 5.

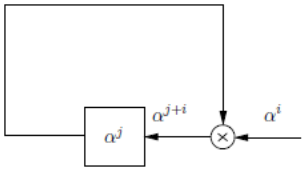

Fig. 10. A symbolic form of an algebraic SA for a mixed-signal CUT

We can interpret these exponents as output responses of a mixed-signal CUT since they possess arithmetic properties and at the same time, the corresponding vectors or signatures possess algebraic properties so an arithmetic data is mapped into an algebraic data. Figure 10 shows the circuit which performs the mapping and computes an algebraic signature.

The circuit of Figure 10 can be obtained from the circuit of Figure 4 if we do the following transform:

$$\alpha^j\alpha^i = (\alpha^j\alpha)\alpha^{i-1} = (\alpha^j\alpha)\overbrace{(1+\alpha^k)}^{\alpha^{i-1}} =$$

$$\alpha^j\alpha + \alpha^{j+1+k} = \alpha^j\alpha + \alpha^l$$

……. (11)

This mapping will not change the probability of undetected error since the finite field GF(2^n) is closed and errors are independent.

In Figure 10, the logic level implementation of the circuit is more complex compared to the circuit of Figure 4, but it is less complex than that of the circuit of Figure 9. Before designing the circuit, we have to make a few observations.

The first observation is that

$$\alpha^j \alpha^i = ((\cdots(\alpha^j \underbrace{\alpha)\alpha)\cdots\alpha}_{i} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(12)}$$

Assume an output response from a mixed-signal CUT as i. The second observation is that the response i can be considered as an exponent of the power, i.e. α^i, it means that the arithmetic values i are mapped into algebraic values α^i.

we can design a signature analyzer in the way shown in Figure 11, based on these observations. Here α is a primitive element of a finite field $GF(2^n)$ and n is the bit length of the output responses.

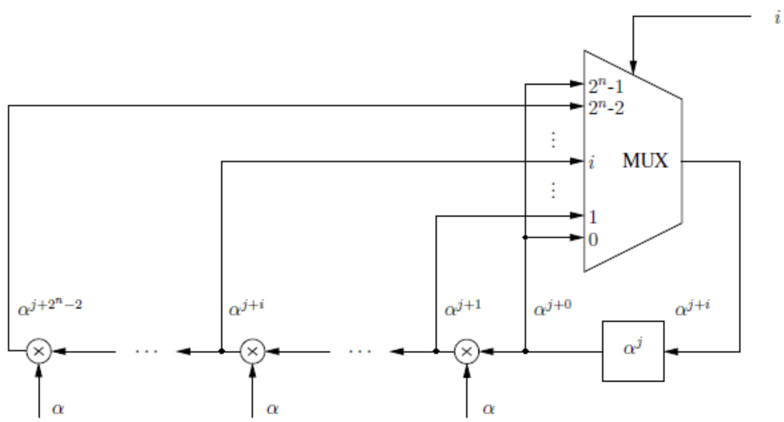

Figure 11: A more detailed symbolic form of the SA

If we consider the case when the analyzer is fed by 3-bit data, its more detailed implementation will have the form of Figure 12.

In figure12, the buses consist of 3 lines, as indicated by the appropriate number. The initial content of the SA before the shift is α^j , or $a_2x^2 + a_1x + a_0$ in the polynomial form. The notations a_k and a_k^+, where index k can be one of the 0, 1, 2, indicate the present and next states, respectively

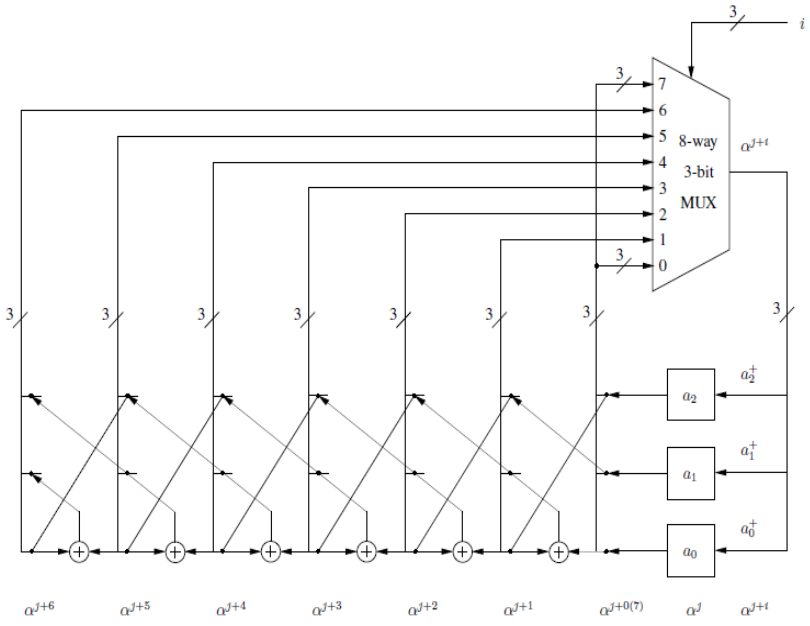

Fig. 12. A register transfer level implementation of the SA

A multiplier by α in GF (2^3) is realized bearing in mind that g(x) = x3 + x + 1, α corresponds to x, and

$$(a_2x^2 + a_1x + a_0)x \bmod g(x) =$$

$$(a_2x^3 + a_1x^2 + a_0x) \bmod g(x) =$$

$$a_2(x + 1) + a_1x^2 + a_0x =$$

$$a_1x^2 + (a_2 + a_0)x + a_2$$

This operation is shown by cross-lines in Figure 10. The multiplexer inputs “0” and “7” are tied together, because $\alpha^7 = \alpha^0$ in the field GF (2^3).

To demonstrate how to use this analyzer, we will have to consider that it receives only two values from a CUT, j and i. Since the CUT is of a mixed-signal nature, there is an unavoidable deviation

of these values by ± 1. The analyzer will map the received data into $\alpha^{j\pm1}$, α^{i+1}, respectively. If the initial content of the SA is 001, then after the first shift the content becomes $\alpha^0\ \alpha^{j+1} = \alpha^{j+1}$.

After the second shift, it changes to $\alpha^{j+1}\ \alpha^{i+1} = \alpha^{j+i+2}$.

It states that for the fault-free CUT the actual result must match one of the values from the interval α^{j+i-2} , α^{j+i+2}. that is one of the following:

$$\alpha^{j+i-2}\ ,\ \alpha^{j+i-1}\ \alpha^{j+i},\ \alpha^{j+i+1}\ ,\ \alpha^{j+i+2} \qquad (13)$$

In order to simplify the SA operation, we will assume that instead of α^0 the initial SA content is $\alpha^{-(j+i)}$. We will refer to this value as the seed value. Then, by the same reasoning, the SA content after two shifts will match one of the following powers:

$$\alpha^{-2}\ ,\ \alpha^{-1}\ ,\alpha^0\ ,\alpha^1\ \alpha^2 \qquad (14)$$

For the closure property of the field GF (2^3), this power set is equivalent to:

$$\alpha^5\ ,\ \alpha^6\ ,\alpha^0\ ,\alpha^1\ \alpha^2 \qquad (15)$$

Since these values are ordered in the decision-making circuit can employ a comparator, reducing the hardware complexity of the SA.

As in any signature analyzer, some errors in the CUT output responses may escape detection. The aliasing rate can be estimated and will coincide with the aliasing rate of the conventional analyzer.

Example: Here we consider a 3-bit CUT, which is fed by two input stimuli. Under the fault-free operation, the CUT produces the output responses j = 101$\pm$1 and i = 110 $\pm$1. The seed value will be

$$\alpha^{-(j+i)} = \alpha^{-(5+6)}) = \alpha^{-11} = \alpha^3,$$

or 011 in the vector form. If the CUT is fault-free, then after 2 shifts the SA content must match one of the elements in the set (6). If the actual responses are 101+1=110 (or α^6) and 110+1=111 (or α^7), the signature will be $\alpha^3\alpha^6\alpha^7=\alpha^2$ which belong to the set (6). And the decision-making circuit will generate a pass signal. The validity of such a decision is determined by the aliasing rate.

Now assume that a fault in the CUT has made the following changes in the output responses: 110 $\mapsto$ 011 ($\alpha^6 \mapsto \alpha^3$) and 111 $\mapsto$ 100 ($\alpha^7 \mapsto \alpha^4$). Then the actual signature will become $\alpha^3\ \alpha^3\ \alpha^4 = \alpha^3$. This element does not belong to the set (6), so the fault is detected. There are two distinct ways of designing the decision-making circuit depending on the optimization criteria.

Hardware overhead: The following approach can be employed if performance is paramount and time overhead is not desirable. Let m be the number of output responses. All of the 2m+1 α-multiplier outputs that belong to the set (6), are connected to the first inputs of the 2m+1comparators of a similar type. The second inputs of these comparators are shared and fed by the vector 0…01. If the CUT is fault-free, one of the comparators will produce a logic "1" signal. The logic OR of the comparator outputs will constitute a pass / fail signal.

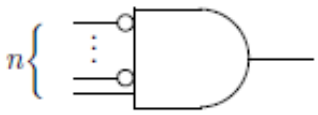

Fig 13: An n-bit comparator

The logic diagram of the n-bit comparator is shown in Figure 11.

This procedure is based on the fact that the fault-free CUT produces one of the signatures from the set (6). If the actual signature is α^0, the comparator connected directly to the signature register produces a logic "1", thus indicating that the CUT is fault free. If the actual signature is α^6, then the product $\alpha^6\alpha$ generated at the output of the first α-multiplier equals to 1, which is detected by the next comparator. The same n reasoning applies to the rest of signatures from the set (6).

Time overhead: The hardware complexity can be further reduced if time overhead is allowed, for implementation use the following seed value:

$\alpha^{-(j+i+m+1)}$, where m is the number of output responses.

For the above example, $\alpha^{-(11+3)} = \alpha^0$, and the set (6) will transform to:

$$\alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$$

After the last output response has been shifted in, the SA continues to shift its content 2m+1 more times, while the input i is forced to 1. This ensures that the SA content is multiplied by α with each shift. For the above example, 2m+1=5. If within this time, the match with an element of the set (7) has been determined, the CUT is considered to be fault-free. Otherwise, it is faulty.

If the CUT is fault free and its output responses have not exceeded their tolerances, then while cycling through the states during the extra 2m+1 shifts, the output of the multiplexer in Figure 9 will go through the power α^0 or vector 0… 01. The match with the vector 0… 01 is detected by

the comparator of Figure 13 connected to the multiplexor's output. The comparator output is producing a pass / fail signal.

In figure 11, the implementation complexity increases significantly with the growth of n. This circuit can only be implemented for the output responses with relatively low values of n. For greater values of n, we will modify the circuit of Figure 11 to the one shown in Figure 14.

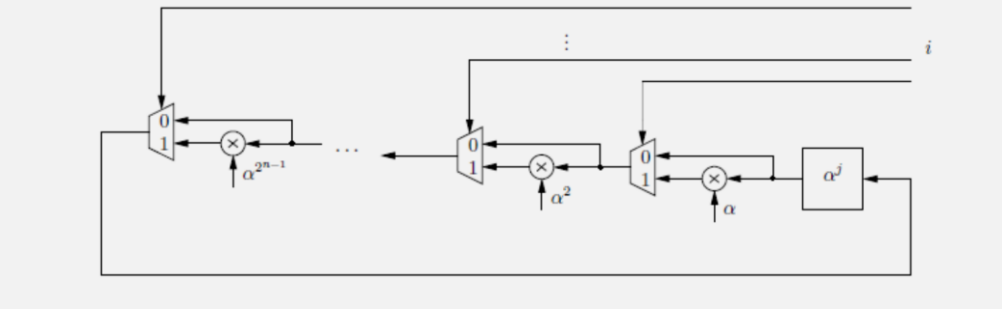

Fig. 14. A binary-weighted version of the SA

The modified circuit contains binary-weighted stages and is more economical in terms of hardware. The complexity of the multiplier $x\alpha^i$ is comparable with that of the multiplier $x\alpha$, whereas the number of multipliers drops from 2n to n. The economy increases with the growth of n.

For the case of 3-bit data, the circuit of Figure 14 transfers to the one shown in Figure 15. This circuit operates much in the same way. The α^i-multipliers structure is determined from the following expressions:

$$x(a_2x^2 + a_1x + a_0) \bmod g(x) = a_1x^2 + (a_2 + a_0)x + a_2,$$

$$x^2(a_2x^2 + a_1x + a_0) \bmod g(x) = (a_2 + a_0)x^2 + (a_2 + a_1)x + a_1$$

$$x^4(a_2x^2 + a_1x + a_0) \bmod g(x) = (a_2 + a_1 + a_0)x^2 + (a_1 + a_0)x + (a_2 + a_1)$$

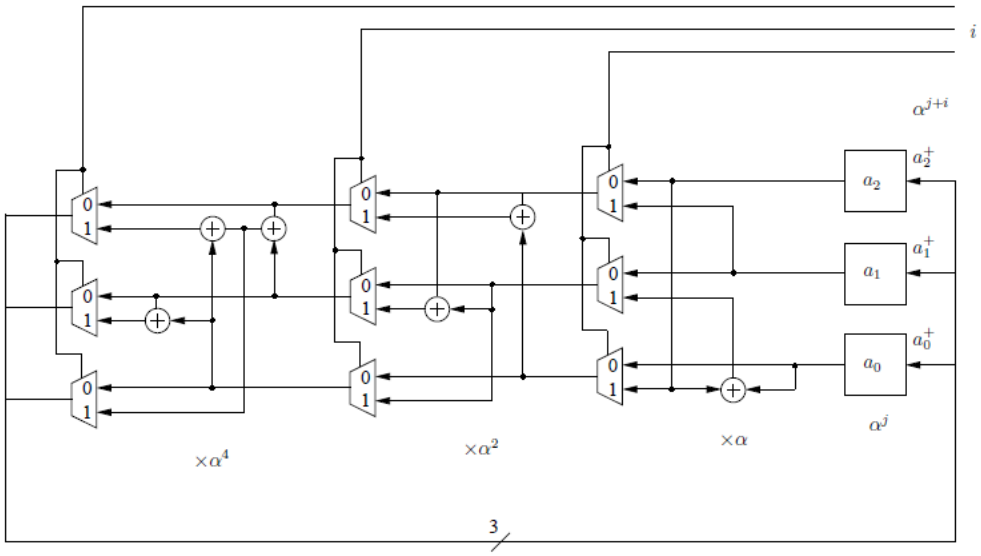

Fig. 15. A register transfer level implementation of the 3-bit SA

Conclusion to Chapter 3

The main idea is to remove the adder from the arithmetic signature analyzer and it becomes just a multiplying device. we supply *I* to the analyzer, but it interprets it as αi{α}^{i} and multiplies the current content, αj{α}^{j}, by αi{α}^{i}. The operation of this analyzer is equivalent to the normal analyzer, whose input data are shuffled (i.e., what is important, the probability of error detection is not changing). An outcome from the main idea is that the reference signatures (5 signatures that are discussed in the paper) are contagious. So, if we go through them, it will always include 001. If this code is not found, the circuit is faulty.

For a fault-free ADC, your actual signature drops into the range

α3{α}^{i+1}, α4{α}^{i+1}, α5{α}^{i+1},..., α12{α}^{i+1}, α13{α}^{i+1},α14{α}^{i+1}, ... , α21{α}^{i+1}, α22{α}^{i+1}, α23{α}^{i+1}. So, you take α23{α}^{i+1} and compare the actual signature with it. If they match, then the ADC is fault-free. If not, then you multiply this actual signature by α and see if the result of multiplication now matches α^{23}{α}^ {i+1, etc. If we clock it 20 times and it never matches α^{23}{α}^{i+1}, then the ADC is faulty. Multiplication of the register content by α can be easily done by the same circuit; you just keep i=1 and apply one clock (shift). For avoid the carry propagation to the main circuit, it is one of the most favorable conceptions for the future Signature analyzer. It's easy to use, operate and implementation.

Chapter 4

Experimental Results and Tests

In Figure 16, the experimental setup to test the proposed method of signature analysis is shown.

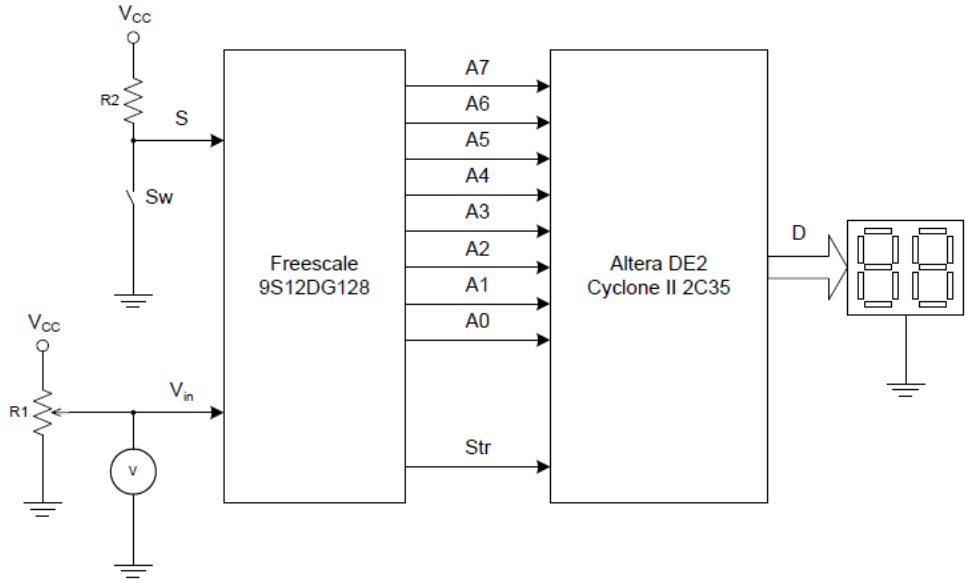

Fig. 16. The experimental setup

From the figure 16, we can see, the setup includes the microcontroller system board Adapt9S12D which is based on the Freescale's 9S12DG128 microcontroller and the Altera DE2 Development Board based on the Cyclone II EP2C35F672C6 FPGA device. 16 input test stimuli (voltages Vin,) equally distributed over the range (0 ~5.12) V and applied them to the analog-to-digital converter (ADC) of the 9S12 microcontroller which served as a mixed-signal system [1]. Input voltage was measured by a high-precision voltmeter and regarded as a nominal test input value.

The circuit in Figure 16 operates as follows. Every time the switch Sw is closed, the system performs 8 measurements of the same test signal and averages the result by accumulating the sum of the eight 8-bit measurements and shifting it right three times, which eliminates noise. Each conversion result for a properly operating device can deviate from the nominal value by ±1, For example, if Vin = 40mV, the conversion result can as the TABLE A.

Table: Input stimuli generation using different voltage level

Input voltage			output code		
mv	Min	Nom	Max	No fault	Fault
80	3	4	5	3	3
400	19	20	21	21	21
720	35	36	37	37	37
1040	51	52	53	53	53
1360	67	68	69	68	70
1680	83	84	85	85	85
2000	99	100	101	99	99
2320	115	116	117	117	117
2640	131	132	133	133	133
2960	147	148	149	148	150
3280	163	164	165	165	165
3600	179	180	181	179	179
3920	195	196	197	197	197
4240	211	212	213	212	240
4560	227	228	229	229	230
4880	243	244	245	244	244

Therefore, each of the thirty-two 8-bit average results contains an error of at most ±1 count. The test stimuli have been selected equal to the midpoints of the quantization bins, thereby increasing the uncertainty and worsening the probability of undetected error. If the test stimuli would have been selected at the transition points of the characteristic, the probability of undetected error (aliasing rate) would improve. This follows from the observation that each conversion would result in 2 possible values as opposed to 3 possible values in the previous case.

As soon as average values of the conversion results are computed by the microcontroller, they are transferred to the DE2 board. The transfer of each data is accompanied by a high-to-low transition of the strobe signal Str. The Str signal serves as a clock for the state machine that implements the signature analyzer (in its 8-bit configuration). The signature, D, is displayed on a two-digit 7-segment display in the hexadecimal form.

The first experiment was performed on the properly operating device. In the second experiment, the average results were corrupted digitally in the microcontroller (thereby simulating random faults in the ADC) and sent to the analyzer. The analyzer has correctly identified the faulty device. The relationship between input voltages and output codes is presented in Table II. Based on this Table and taking into consideration that $g(x) = x^8 + x^4 + x^3 + x^2 + 1$, the seed value is calculated as follows.

$$4 + 20 + \ldots + 244 = 1984 = 199 \bmod (2^8 - 1) = 199 \bmod;$$

$$\alpha^{-199} = \alpha^{56} = 01011101$$

$$\text{Seed Value} = \alpha^{56}\, \alpha^{-16} = \alpha^{40} = 01101010 = 106.$$

In addition to test experiments, the operation of the analyzer (the DE2 part of the test setup) was simulated using Altera Quartus II software. Based on the two experiments represented in Table II, the signatures that correspond to fault-free and faulty ADCs are respectively 233 and 201 (in decimal form). The process of calculation of these signatures is demonstrated in Figures 15 and 16. Figures 22 and 23 represent the fault detection process. The actual final signatures are shifted additionally 32 times. If the value 1 appears in the analyzer during these shifts, the system is fault free. Otherwise, it is faulty.

The simulation results matched the experimental results.

But in this project, I used Altera board switches as an input of stimuli in lieu of Microcontroller output. That's why it was changed manually for putting input to the FPGA Altera board. The Altera Board DE2 115 has 18 input PINs.

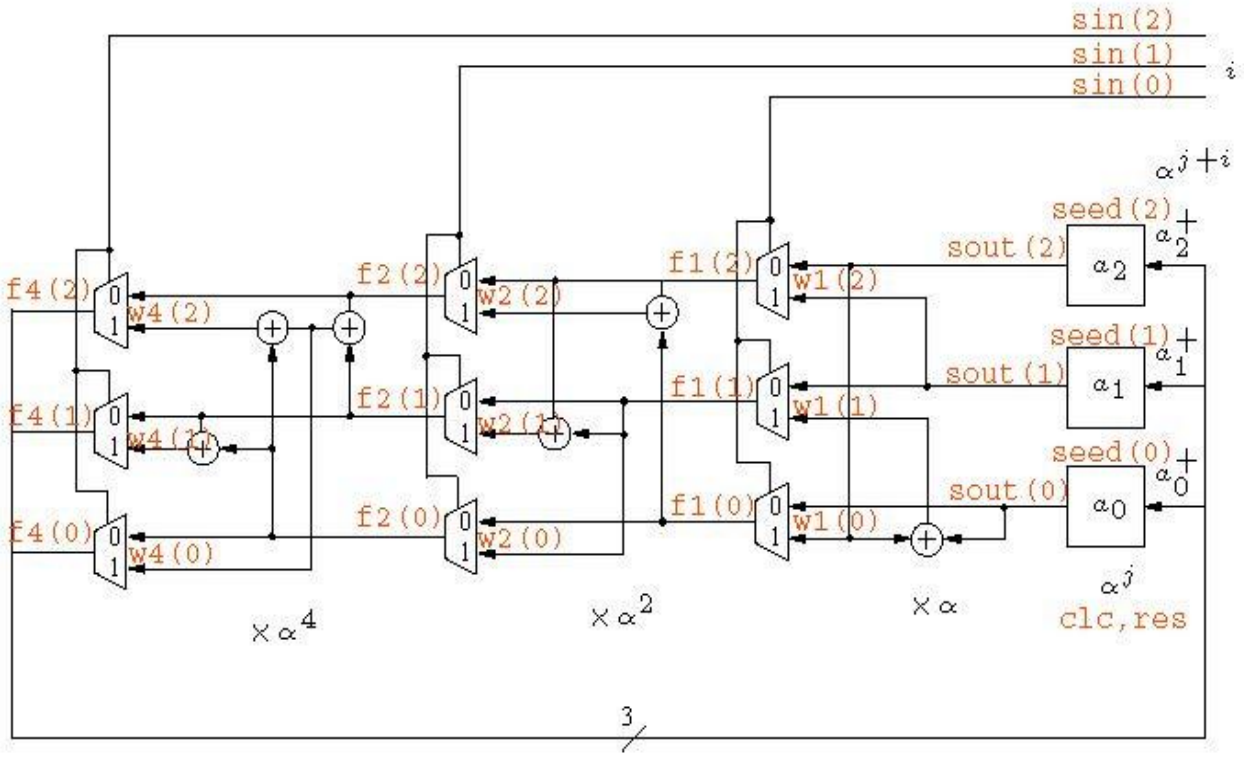

Figure 17: Circuit design for 3 bit SA for a resister transfer level

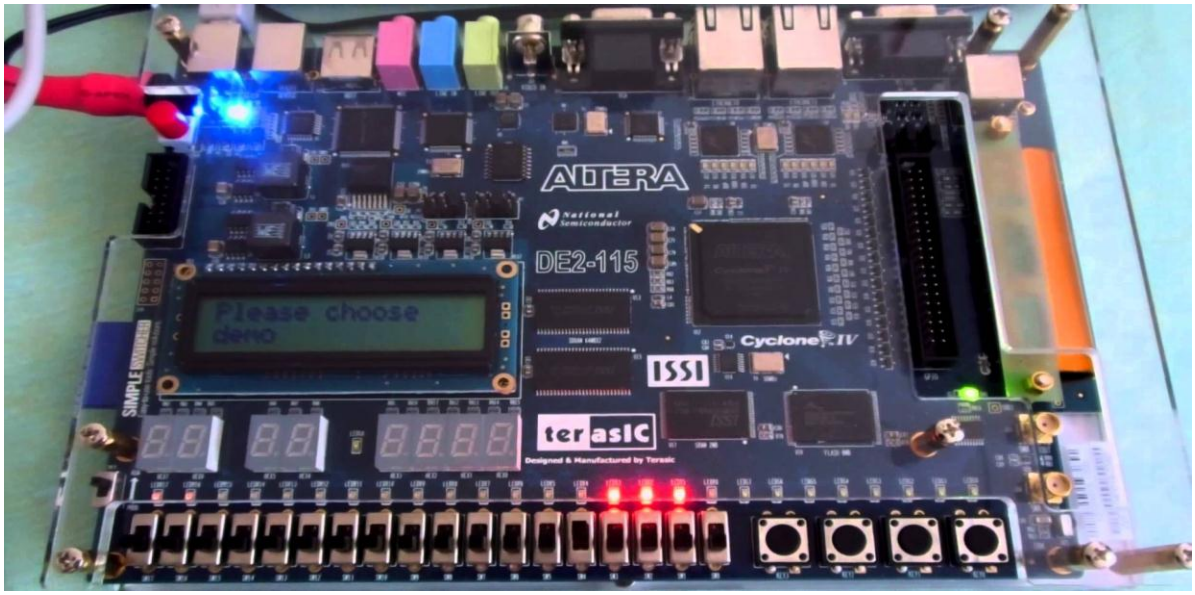

Figure 18: Altera DE2 115 FPGA Board

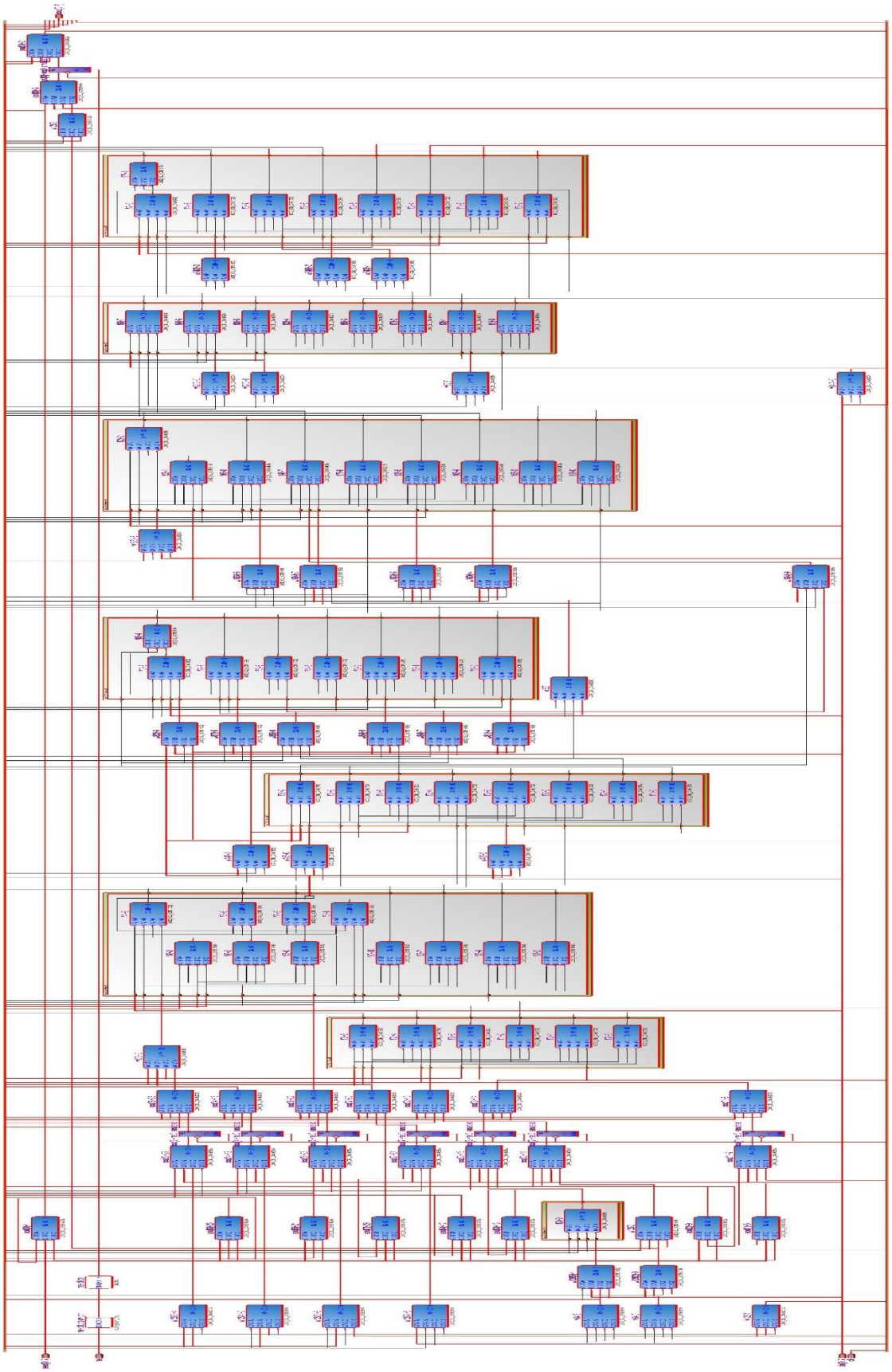

FIG:19 A resister transfer level implementation of the 8-Bit Signature Analyzer

Coding Compilation result in Altera.

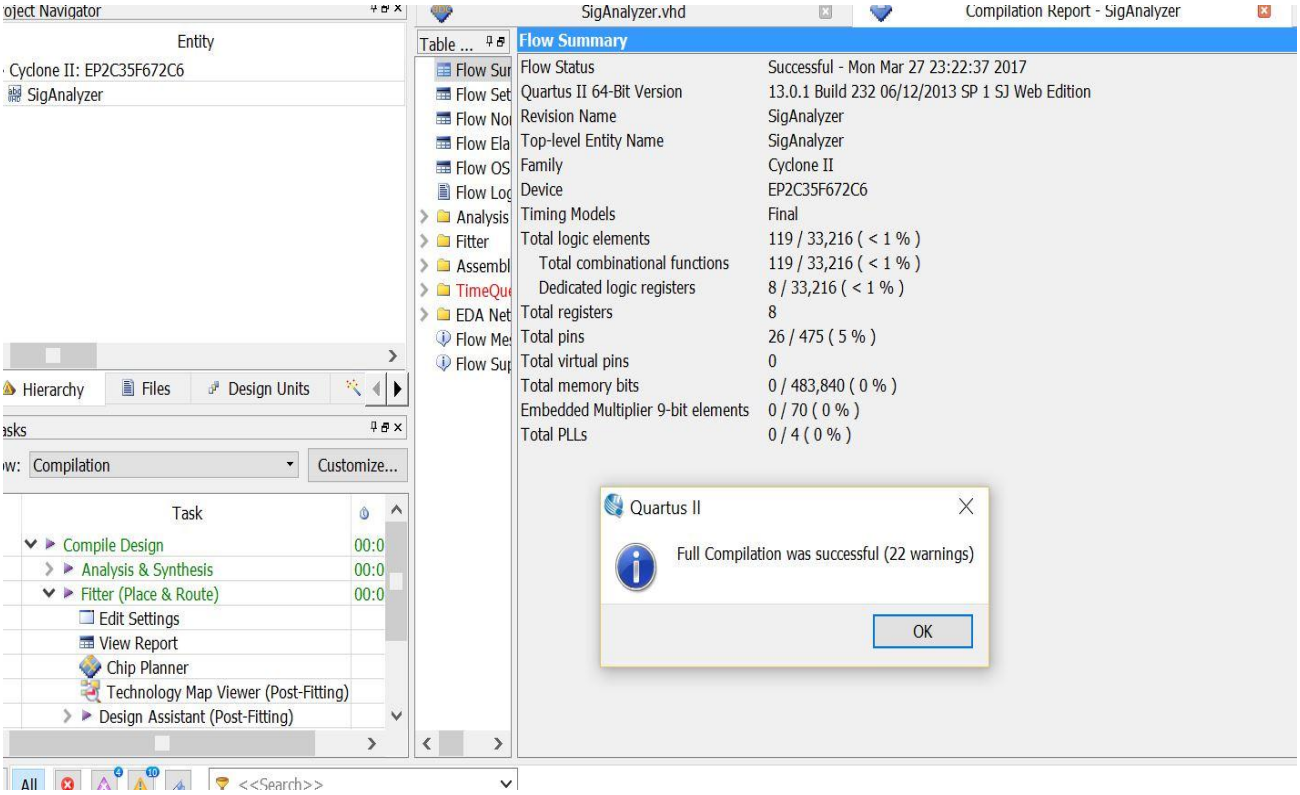

Figure20: Compilation result of coding in Altera DE2 115

Block Diagram of designed algorithm

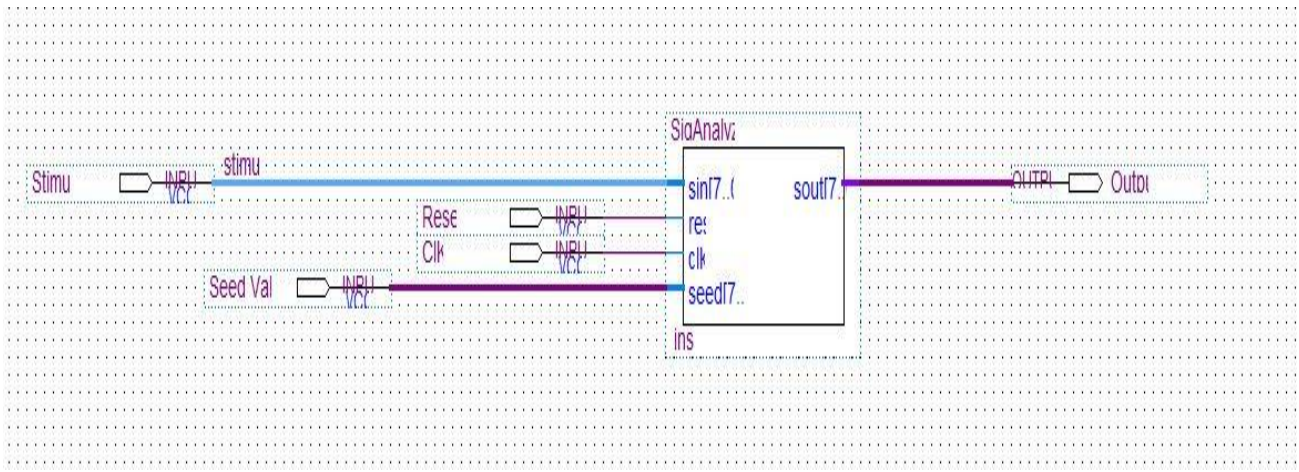

Figure 21: Designed Block diagram from Altera DE2 115

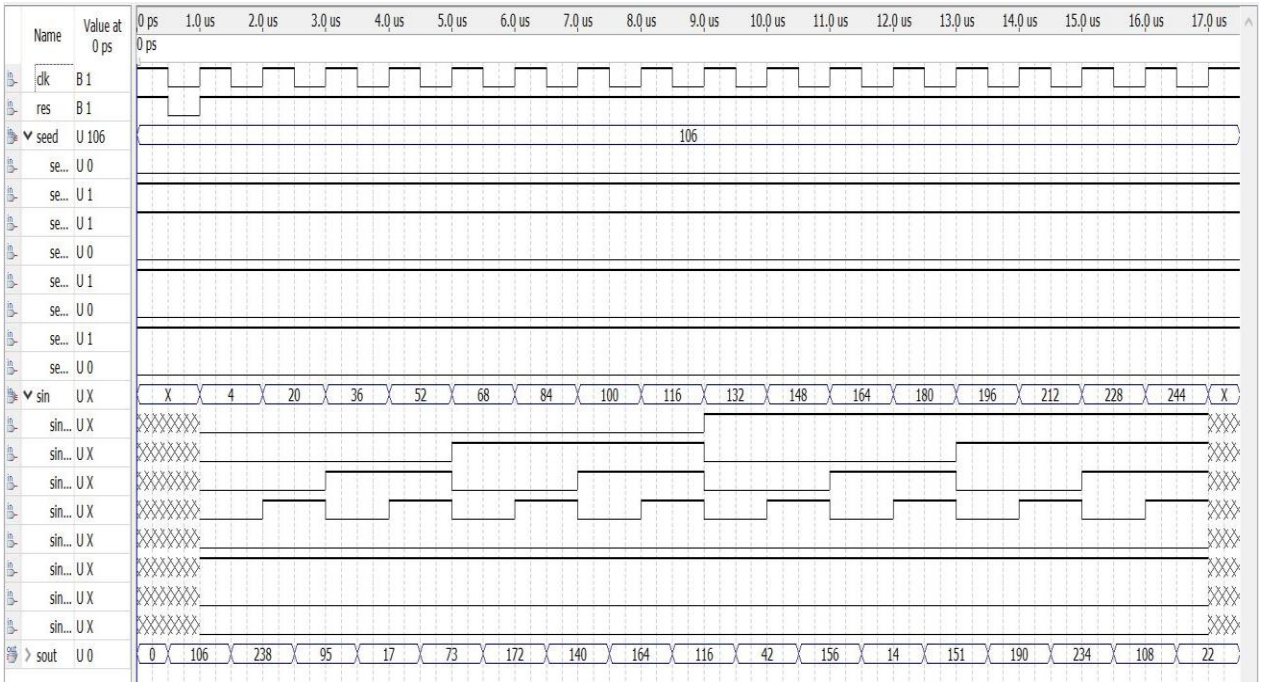

Figure 22: Experiment for nominal Value in input stimuli of the Designed device

Figure 22 & 23 is a graphical representation of input stimulus in CUT for Nominal and Maximum value. Figure 24 is the minimum value of input stimulus. In this case, seed value was always 106.

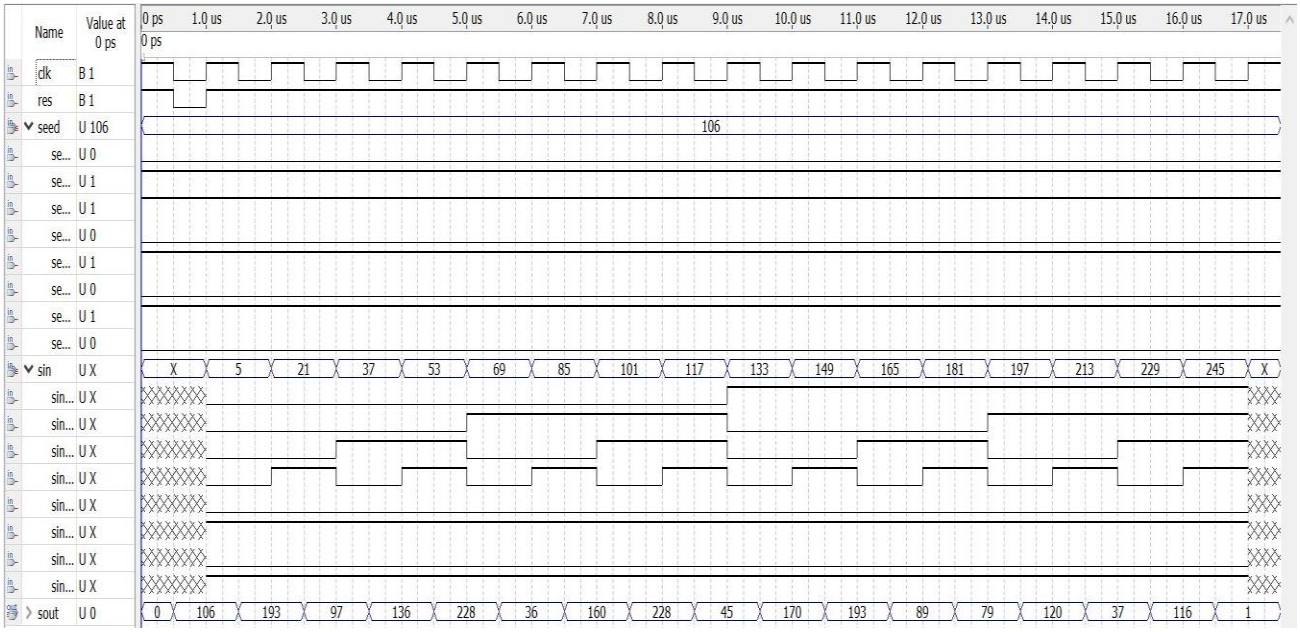

Figure 23: Experiment for Maximum Value in input stimuli of the Designed device

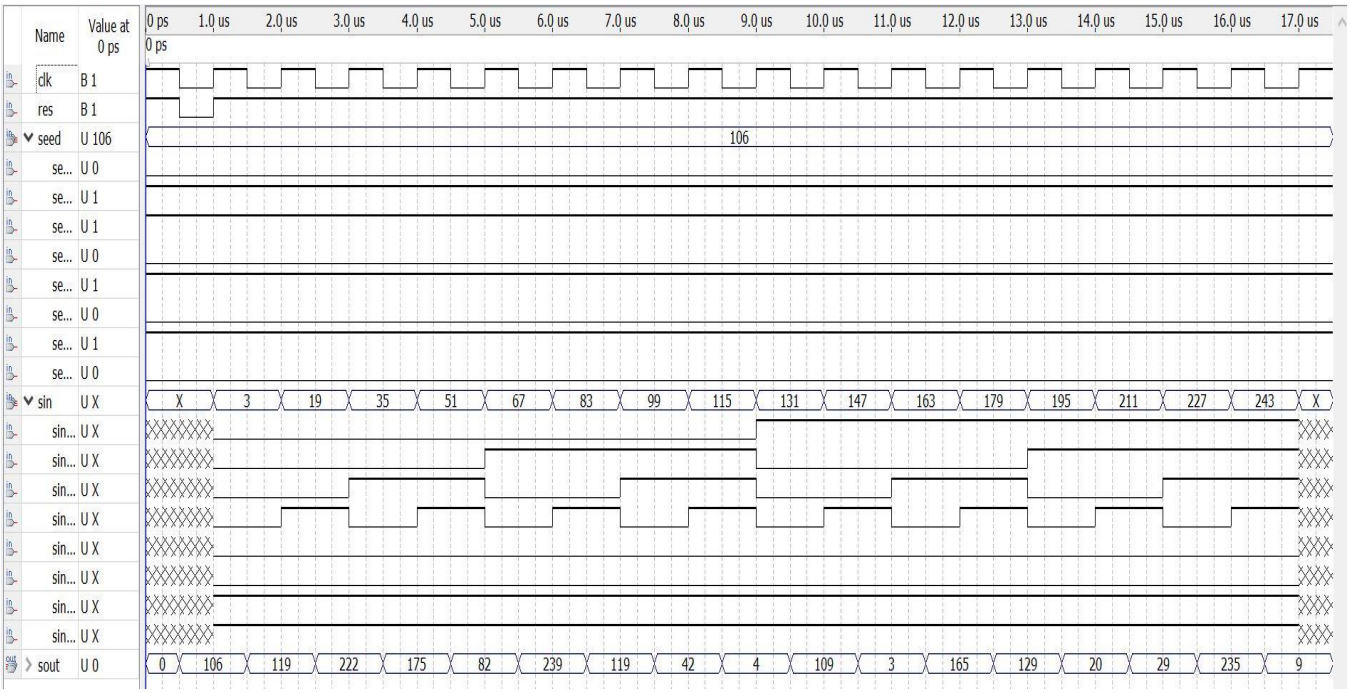

Figure 24: Experiment for Minimum Value in input stimuli of the Designed device

In Figure 25, we are getting combination "1" is detected in "sout" as input stimuli in CUT are 1. So the CUT/ ADC is ok. This time seed value was 233. But in Figure 26, the input "1" is not detected

For seed value 201. Here ADC should be Faulty.

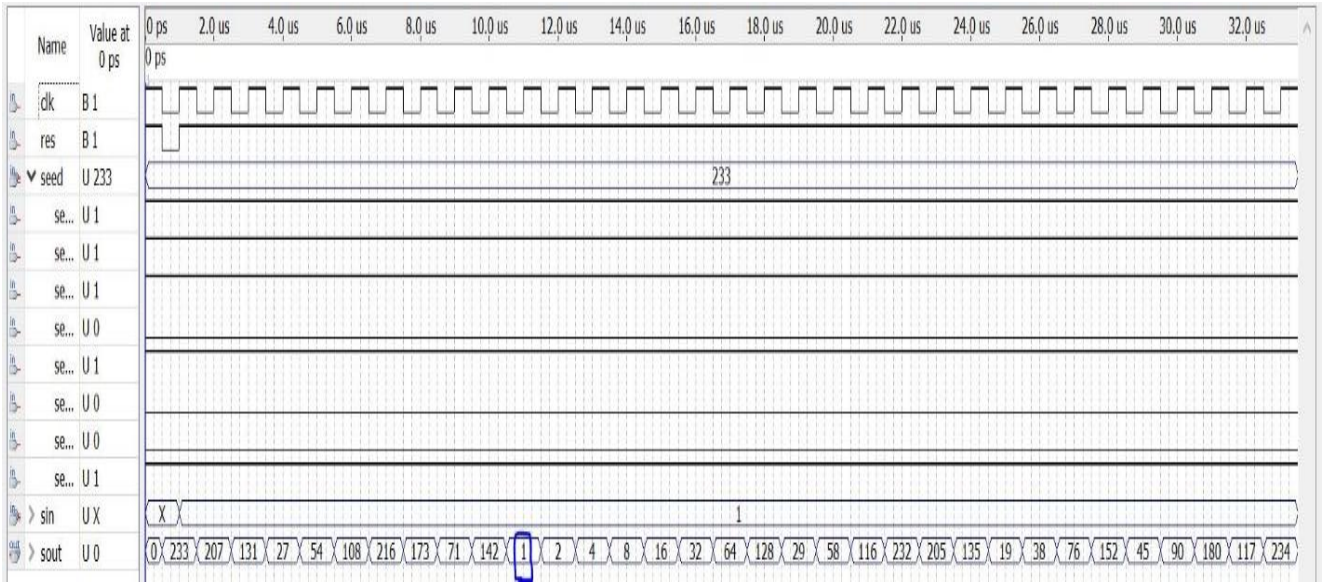

Figure 25: The combination “1” is detected: ADC is operating Properly

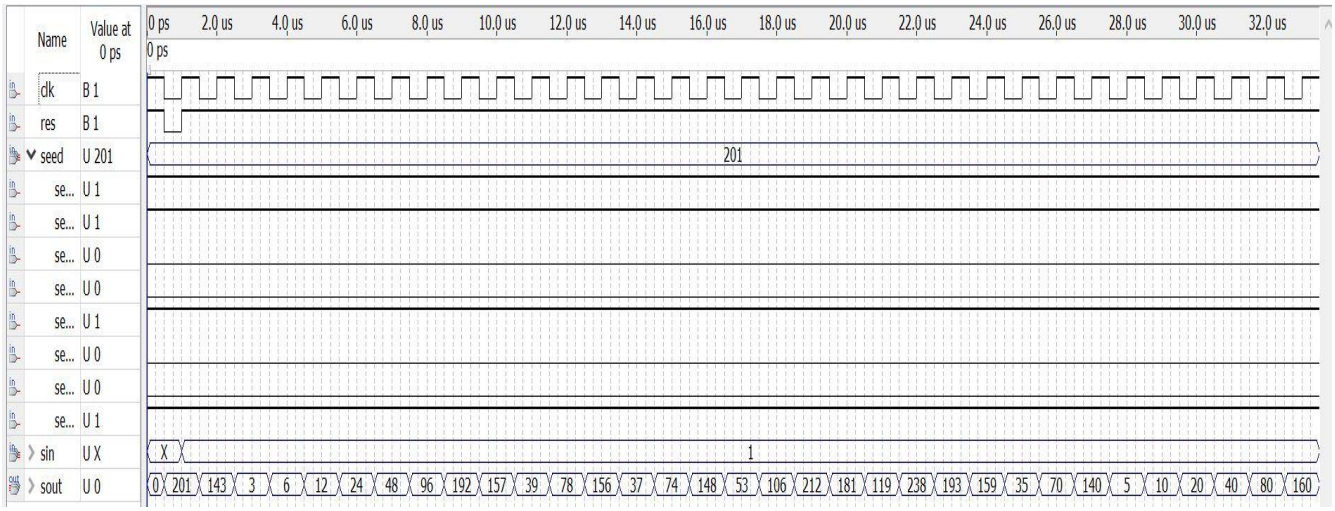

Figure 26: The combination “1” is not detected: ADC should be faulty

Again, when seed value is 251in Figure 27, we are getting the input in output. So, for the seed value 251, the operation is ok and we are getting the perfect output. So we can see that actual seed value is important for getting a perfect result.

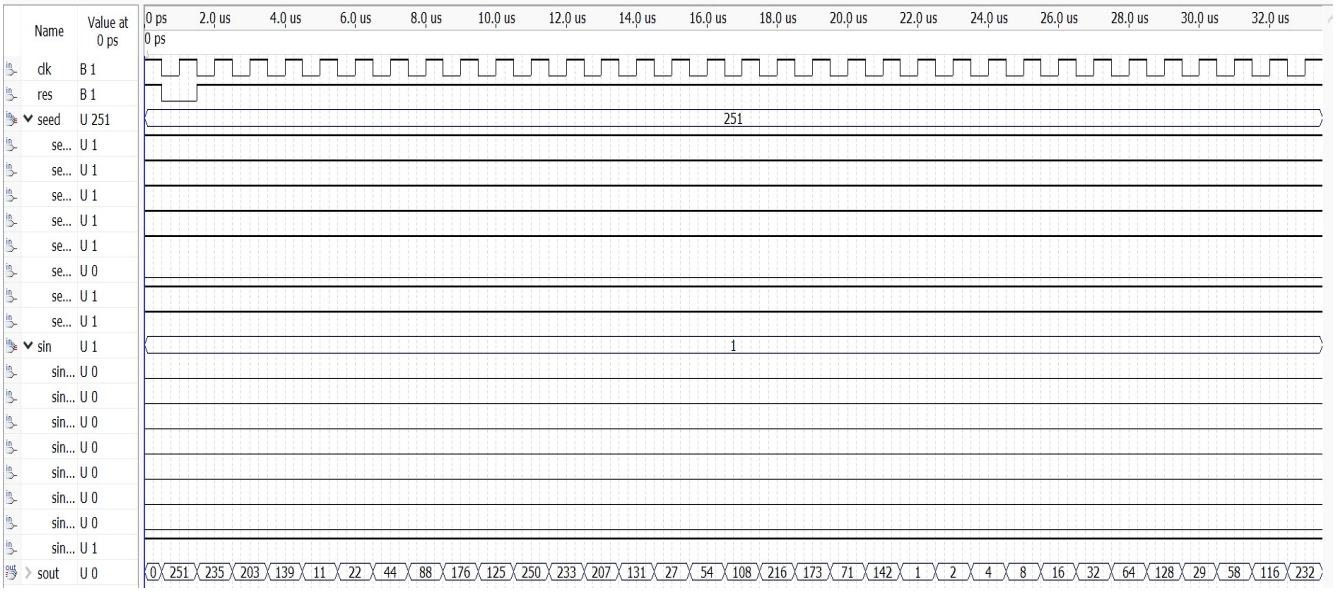

Figure 27: The combination “1” detected, so ADC properly operating

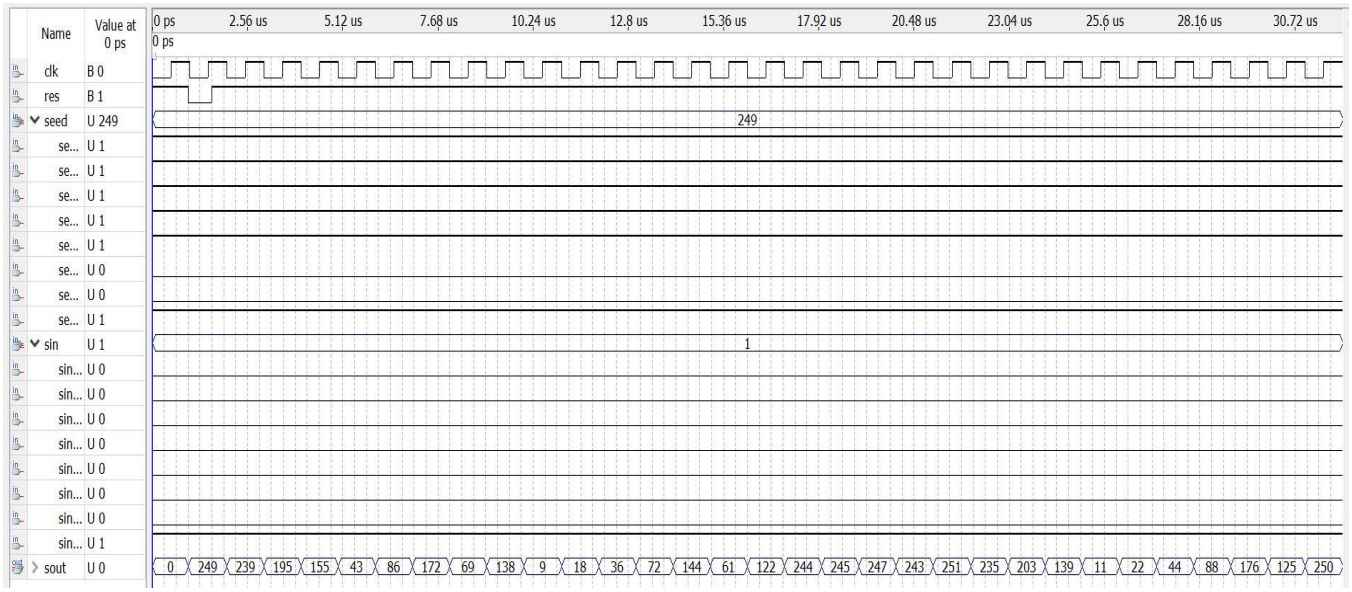

Figure 28: The combination “1” not detected, so ADC faulty.

Another example for wrong seed value ADC showing as a faulty. On the contrary, Figure 29 with seed value 250, showing the CUT/ ADC as a good device. Input stimuli’s decimal is in output as

an unsigned Decimal form. We can get those value as a Binary also just changing the setting in Altera software.

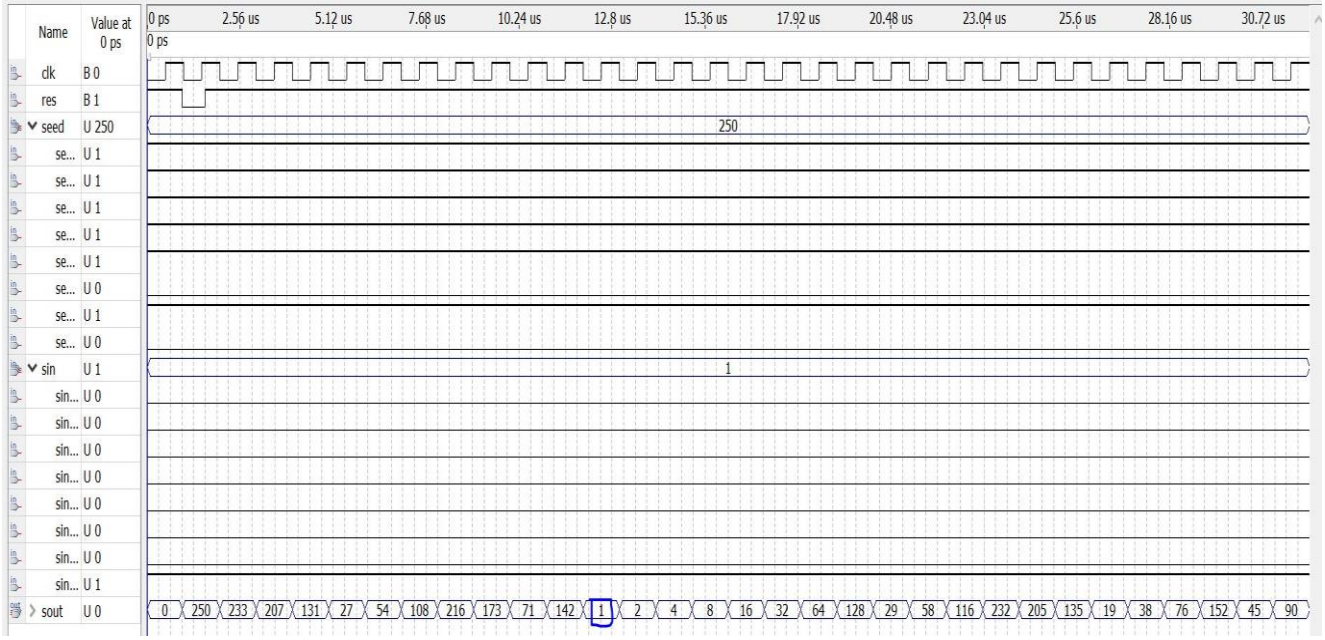

Figure 29: The combination “1” detected, so ADC properly operating

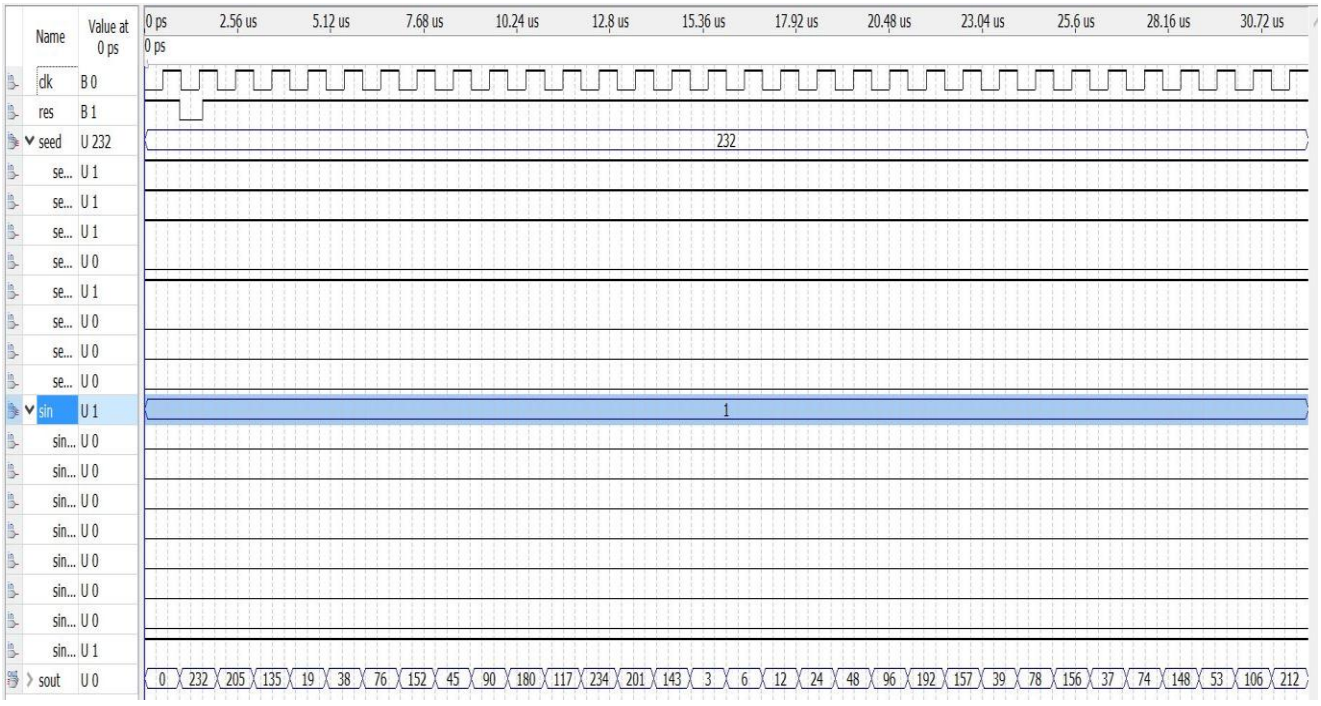

Figure 30: The combination “1” not detected, so ADC faulty

Conclusion:

So, we examined and testing an algebraic signature Analyzer (ASA) technique that can be employed for mixed-signal circuit testing. Here we just tried to demonstrate the strategy of appropriate device design. This device is not carrying arithmetic carry and here is less susceptible to errors. This obstacle of carry propagation providing a better performance to the device.

In future, this schemed can be used as an arithmetic and algebraic error-control coding. So it can be suggested as a future work on arithmetic compactor design.

For this test, we found seed value in 250, 251 and 233. Rest of the value are not seed value and that's why our simulation showing ADC is faulty. So there factor here Proper seed value, the range of seed and productive value of ADC.

APPENDIX

Part of the VHDL code that generated the combinational unit

VHDL Coding:

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY ASigAnalyzer IS
        PORT (sin               : IN            STD_LOGIC_VECTOR(7 DOWNTO 0);
                        res, clk : IN           STD_LOGIC ;
                        seed            : IN            STD_LOGIC_VECTOR(7 DOWNTO 0);
                        sout            : BUFFER        STD_LOGIC_VECTOR(7 DOWNTO 0));
END ASigAnalyzer;

ARCHITECTURE Behavior OF ASigAnalyzer IS

        SIGNAL w128, w64, w32, w16, w8, w4, w2, w1      : STD_LOGIC_VECTOR(7
DOWNTO 0);
        SIGNAL f128, f64, f32, f16, f8, f4, f2, f1      : STD_LOGIC_VECTOR(7 DOWNTO 0);
        COMPONENT mux2to1
        PORT (w0,w1 : IN      STD_LOGIC_VECTOR(7 DOWNTO 0);
                        s               : IN      STD_LOGIC;
                        f               : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0));
        END COMPONENT;
BEGIN
        PROCESS (res, clk)
        BEGIN
                IF res = '0' THEN
                        sout <= seed;
                ELSIF Clk'EVENT AND Clk = '0' THEN
                        sout <= f128;
                END IF;
        END PROCESS;

        stage128: mux2to1 PORT MAP (f64,  w128, sin(7), f128);
        stage64:        mux2to1 PORT MAP (f32,  w64,  sin(6), f64);
        stage32:        mux2to1 PORT MAP (f16,  w32,  sin(5), f32);
        stage16:        mux2to1 PORT MAP (f8,   w16,  sin(4), f16);
        stage8:  mux2to1 PORT MAP (f4,   w8,   sin(3), f8);
        stage4:  mux2to1 PORT MAP (f2,   w4,   sin(2), f4);
        stage2:  mux2to1 PORT MAP (f1,   w2,   sin(1), f2);
        stage1:  mux2to1 PORT MAP (sout, w1,   sin(0), f1);
```

```
--
        w128(7) <= f64(7) XOR f64(6) XOR f64(4) XOR f64(0);
        w128(6) <= f64(6) XOR f64(5) XOR f64(3);
        w128(5) <= f64(7) XOR f64(5) XOR f64(4) XOR f64(2);
        w128(4) <= f64(6) XOR f64(4) XOR f64(3) XOR f64(1);
        w128(3) <= f64(7) XOR f64(6) XOR f64(5) XOR f64(4) XOR f64(3) XOR f64(2);
        w128(2) <= f64(5) XOR f64(3) XOR f64(2) XOR f64(1) XOR f64(0);
        w128(1) <= f64(6) XOR f64(2) XOR f64(1);
        w128(0) <= f64(7) XOR f64(5) XOR f64(1) XOR f64(0);

--
        w64(7)  <= f32(7) XOR f32(4) XOR f32(3) XOR f32(1);
        w64(6)  <= f32(6) XOR f32(3) XOR f32(2) XOR f32(0);
        w64(5)  <= f32(7) XOR f32(5) XOR f32(2) XOR f32(1);
        w64(4)  <= f32(7) XOR f32(6) XOR f32(4) XOR f32(1) XOR f32(0);
        w64(3)  <= f32(7) XOR f32(6) XOR f32(5) XOR f32(4) XOR f32(1) XOR f32(0);
        w64(2)  <= f32(7) XOR f32(6) XOR f32(5) XOR f32(1) XOR f32(0);
        w64(1)  <= f32(6) XOR f32(5) XOR f32(3) XOR f32(1) XOR f32(0);
        w64(0)  <= f32(5) XOR f32(4) XOR f32(2) XOR f32(0);

--
        w32(7)  <= f16(6) XOR f16(3) XOR f16(0);
        w32(6)  <= f16(5) XOR f16(2);
        w32(5)  <= f16(7) XOR f16(4) XOR f16(1);
        w32(4)  <= f16(7) XOR f16(6) XOR f16(3) XOR f16(0);
        w32(3)  <= f16(5) XOR f16(3) XOR f16(2) XOR f16(0);
        w32(2)  <= f16(7) XOR f16(6) XOR f16(4) XOR f16(3) XOR f16(2) XOR f16(1) XOR
f16(0);

        w32(1)  <= f16(5) XOR f16(2) XOR f16(1);
        w32(0)  <= f16(7) XOR f16(4) XOR f16(1) XOR f16(0);

--
        w16(7)  <= f8(7)  XOR f8(6)  XOR f8(4)  XOR f8(1);
        w16(6)  <= f8(7)  XOR f8(6)  XOR f8(5)  XOR f8(3)  XOR f8(0);
        w16(5)  <= f8(6)  XOR f8(5)  XOR f8(4)  XOR f8(2);
        w16(4)  <= f8(5)  XOR f8(4)  XOR f8(3)  XOR f8(1);
        w16(3)  <= f8(7)  XOR f8(6)  XOR f8(3)  XOR f8(2)  XOR f8(1)  XOR f8(0);
        w16(2)  <= f8(5)  XOR f8(4)  XOR f8(2)  XOR f8(0);
        w16(1)  <= f8(6)  XOR f8(3);
        w16(0)  <= f8(7)  XOR f8(5)  XOR f8(2);

--
        w8(7)   <= f4(5)  XOR f4(4)  XOR f4(3);
        w8(6)   <= f4(4)  XOR f4(3)  XOR f4(2);
        w8(5)   <= f4(7)  XOR f4(3)  XOR f4(2)  XOR f4(1);
```

```
        w8(4)  <= f4(6)  XOR f4(2)  XOR f4(1)  XOR f4(0);
        w8(3)  <= f4(4)  XOR f4(3)  XOR f4(1)  XOR f4(0);
        w8(2)  <= f4(7)  XOR f4(5)  XOR f4(4)  XOR f4(2)  XOR f4(0);
        w8(1)  <= f4(7)  XOR f4(6)  XOR f4(5)  XOR f4(1);
        w8(0)  <= f4(6)  XOR f4(5)  XOR f4(4)  XOR f4(0);

--

        w4(7)  <= f2(7)  XOR f2(3);
        w4(6)  <= f2(7)  XOR f2(6)  XOR f2(2);
        w4(5)  <= f2(7)  XOR f2(6)  XOR f2(5)  XOR f2(1);
        w4(4)  <= f2(6)  XOR f2(5)  XOR f2(4)  XOR f2(0);
        w4(3)  <= f2(7)  XOR f2(5)  XOR f2(4);
        w4(2)  <= f2(6)  XOR f2(4);
        w4(1)  <= f2(5);
        w4(0)  <= f2(4);

--
        w2(7)  <= f1(5);
        w2(6)  <= f1(4);
        w2(5)  <= f1(7)  XOR f1(3);
        w2(4)  <= f1(7)  XOR f1(6)  XOR f1(2);
        w2(3)  <= f1(7)  XOR f1(6)  XOR f1(1);
        w2(2)  <= f1(6)  XOR f1(0);
        w2(1)  <= f1(7);
        w2(0)  <= f1(6);

--

        w1(7)  <= sout(6);
        w1(6)  <= sout(5);
        w1(5)  <= sout(4);
        w1(4)  <= sout(7)  XOR sout(3);
        w1(3)  <= sout(7)  XOR sout(2);
        w1(2)  <= sout(7)  XOR sout(1);
        w1(1)  <= sout(0);
        w1(0)  <= sout(7);


END Behavior;

--8-bit mux2to1 component


library ieee;
```

```
use ieee.std_logic_1164.all;

ENTITY mux2to1 IS
        PORT (w0,w1 : IN     STD_LOGIC_VECTOR(7 DOWNTO 0);
                     s              : IN     STD_LOGIC;
                     f              : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0));
END mux2to1;


ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
                     f <= w0 WHEN s='0' ELSE w1;
END Behavior;
```

References

1.Vadim Geurkov; Lev Kirischian

Designing of an algebraic signature analyzer for mixed-signal systems

2014 IEEE 12th International New Circuits and Systems Conference (NEWCAS) Year: 2014 Pages: 165 - 168, DOI: 10.1109/NEWCAS.2014.6934009

2. DHIRAJ K. PRADHAN, SANDEEP K. GUPTA, AND MARK G. KARPOVSKY

Aliasing Probability for Multiple Input Signature Analyzer

IEEE TRANSACTIONS ON COMPUTERS, VOL. 39, NO. 4, APRIL 1990

3. Dhiraj K. Pradhan, Fellow, IEEE, and Sandeep K. Gupta

A New Framework for Designing and Analyzing BIST Techniques and Zero Aliasing Compression, *IEEE TRANSACTIONS ON COMPUTERS*, VOL. 40, NO. 6, JUNE 1991

4. Janusz Rajski, Member, IEEE, and Jerzy Tyszer, Member, IEEE

The Analysis of Digital Integrators for Test Response Compaction

IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-11: ANALOG AND DIGITAL SIGNAL PROCESSING, VOL. 39, NO. 5, MAY 1992

5. Naveena Nagi, Abhijit Chatterjee, Heebyung Yoon, and Jacob A. Abraham

Signature Analysis for Analog and Mixed-Signal Circuit Test Response Compaction

IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 17, NO. 6, JUNE 1998

6. P. Bardell, W. McAnney, and J. Savir,

Built-in Test for VLSI: Pseudorandom Techniques. New York: Wiley, 1987.

7. D. K. Pradhan and S. Gupta,

"A new framework for designing and analyzing BIST techniques and zero aliasing compression," *IEEE Trans. Compute*., June 1991, pp. 743–763.

8. E. J. McCluskey, "Built-in self-test techniques," *IEEE Design & Test*

Mag., Apr. 1985, pp. 21–28. 90 J. P. Hayes, "Transition count testing of combinational logic circuits*," IEEE Trans. Compute*., June 1976, pp. 613–620.

10. J. P. Hayes, "Checksum test methods," in Proc. FTCS-6, 1976, pp. 114–120.

11. J. Savir, "Syndrome-testable design of combinational circuits," *IEEE Trans. Compute*., June

1980, pp. 442–451.

12. J. Rajski and J. Tyszer, "The analysis of digital integrators for test response compaction," *IEEE Trans. Circuits Syst. II,* May 1992, pp.293–301.
13. N. Nagi, A. Chatterjee, and J.A. Abraham, "Fault simulation of linear analog circuits," Analog Integrated Circuits and Signal Processing*: An International Journal*, vol. 4, pp. 245–260, 1993.
14. Ivanov and V.K. Agrawal. "An iterative technique for calculating aliasing probability of linear feedback shift registers," *in Proc. 18th Inr. Symp. Fault Tolerant Compute., Tokyo*, Japan, 1988.
15. T. W. Williams, A. Daehn, M. Gruetzner, and C. W. Starke, "Aliasing errors in signature analysis registers," *IEEE Design Test Compute*., vol. C-36, no. 4, pp. 39-45, Apr. 1987.
16. D. K. Pradhan, M. Y. Hsiao, A.M. Patel, and S.Y. Su, "Shift Register designed for on-line fault detection," in Proc. FTCS, 1978, pp. 173-17X.
17. S. Lin and D. J. Costello, Error Control Coding: Fundamentals and Applications.
18. Vadim Geurkov; Vladimir Dynkin; Reza Sedaghat
    An error-locating signature analyzer to identify faulty units in digital systems
    2008 Canadian Conference on Electrical and Computer Engineering; Year: 2008
    Pages: 001073 - 001076, DOI: 10.1109/CCECE.2008.4564702
    IEEE Conference Publications
19. I. Murashko; V. Yarmolik; M. Puczko
    The power consumption reducing technique of the pseudo-random test pattern generator and the signature analyzer for the built-in self-test
    The Experience of Designing and Application of CAD Systems in Microelectronics, 2003. CADSM 2003. Proceedings of the 7th International Conference.; Year: 2003
    Pages: 141 - 144, DOI: 10.1109/CADSM.2003.1255008; Cited by Papers (2)
    IEEE Conference Publications
20. Abu Khari bin A'ain; C. T. Lim; Kok Hong Ng; Sheng-Kwang Ng;
    A study on signature analyzer for design for test (DFT)
    2004 IEEE International Conference on Semiconductor Electronics
    Year: 2004; 5 pp., DOI: 10.1109/SMELEC.2004.1620855 IEEE Conference Publications