DESIGNING OF AN ALGEBRAIC SIGNATURE ANALYZER FOR MIXED-SIGNAL SYSTEMS AND TESTING

By

Muhammad Mohsin Babar

Bachelor of Science in Electrical and Electrical Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh, 2005

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

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.

Designing of an Algebraic Signature Analyzer for Mixed-Signal Systems and Testing

Muhammad Mohsin Babar

Master of Engineering

Electrical and Computer Engineering

Ryerson University, Toronto, 2017

ABSTRACT

While the design of signature analyzers for digital circuits has been well researched in the past, the common design technique of a signature analyzer for mixed-signal systems is based on the rules of an arithmetic finite field. The analyzer does not contain carry propagating circuitry, which improves its performance as well as fault tolerance. The signatures possess the interesting property that if the input analog signal is imprecise within certain bounds (an inherent property of analog signals), then the generated signature is also imprecise within certain bounds. We offer a method to designing an algebraic signature analyzer that can be used for mixed-signal systems testing. The application of this technique to the systems with an arbitrary radix is a challenging task and the devices designed possess high hardware complexity. The proposed technique is simple and applicable to systems of any size and radix. The hardware complexity is low. The technique can also be used in algebraic coding and cryptography.

CONTENTS

FIGURES

TABLES

CHAPTER 1: INTRODUCTION

A number of mixed-signal testing approaches have been proposed to detect faults in digital circuitry [1]-[2]. Early approaches were primarily aimed at defect oriented testing of the digital circuitry for manufacturing and system-level testing [2]-[3]. More recent approaches are target digital functional test and measurement, or specification oriented testing [4]-[6]. Signature analysis has been widely used for digital and mixed-signal systems testing [1]–[12]. Mixed-signal systems consist of both digital and analog circuits; however, the signature analysis method is only applicable to the subset of these systems that have digital outputs (such as analog-to digital converters, measurement instruments etc.). Signature analysis can be employed as an external test solution or can be embedded into the system under test. In the built-in implementation, a circuit under test (CUT) of digital or mixed signal nature is fed by test stimuli, while the output responses are compacted by a signature analyzer (SA), as illustrated in Figure 1. The actual signature is compared against the fault free circuit's signature and a $\boldsymbol{pass/fail}$ decision is made. A signature of a fault free circuit is referred to as a reference signature. If the CUT is of a digital nature, the SA essentially constitutes a circuit that computes an algebraic remainder. The reference signature has only one, punctual value, and the decision-making circuit consists of a simple digital comparator. If the CUT is of a mixed-signal

nature, the SA computes an arithmetic residue. In this case, the reference signature becomes an interval value and the decision-making circuit uses a window comparator.

Design methods for an algebraic signature analyzer have been well developed in error-control coding [13]. A remainder calculating circuit for an arbitrary base (binary or non-binary) can be readily designed for a digital CUT of any size. In contrast, it is much harder to design a residue calculating circuit, specifically for a non-binary base [14]. Furthermore, due to the presence of carry propagating circuitry, the implementation complexity and error vulnerability of the residue calculating circuit is higher compared to the remainder calculating circuit.

We propose an approach to designing an algebraic signature analyzer that can be used for mixed-signal systems testing. Due to an algebraic nature, the analyzer does not contain carry propagating circuitry. This helps to improve its error immunity, as well as performance.

CHAPTER 2: SCOPE OF THIS PROJECT

Designed of an algebraic signature analyzer on the basis of a polynomial division circuit, as shown in Figure 2 [3], [13], [15]. This circuit divides the incoming sequence of non-binary symbols (digits), $\boldsymbol{a_{m-1}, \ldots, a_1, a_0,}$

treated as a polynomial:

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

Eq. 1

by the polynomial:

$$\boldsymbol{p(y) = p_t y^t + \cdots + p_1 y + p_0, t \ll m}$$

Eq. 2

The remainder:

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

Eq. 3

constitutes a CUT signature.

Each digit, $\boldsymbol{a_i, 0 \leq i \leq m - 1}$ consists of $\boldsymbol{n}$ bits and is considered to be an element of the field $\boldsymbol{GF(2^n)}$. The degree of the polynomial (2), or the number of stages, $\boldsymbol{t}$, in Figure 2, depends on the desired probability of undetected error in the sequence of incoming digits. For long sequences with independent errors, this probability is estimated as $\boldsymbol{P_{nd} \approx 2^{-tn}}$. In practice, $\boldsymbol{n \geq 8}$ and even for the one-stage circuit, $\boldsymbol{P_{nd} \leq 2^{-(1\times 8)} = 0.0039}$, which is quite low. Therefore, a

multiple-input signature analyzer normally contains only one stage. Such an analyzer is presented in Figure 3 [14], where α is a primitive element of the field $\boldsymbol{GF(2^n)}$, i.e. a root of a primitive polynomial

$$\boldsymbol{g(x) = g_{n-1}x^{n-1} + \ldots + g_1x + g_0}$$

Eq. 4

The field of each element can be represented by a power of $\boldsymbol{\alpha}$. Let 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 expression:

$$\boldsymbol{\alpha^j\alpha \oplus \alpha^i = \alpha^k}$$

Eq. 5

Without a loss of generality, we will consider a $\boldsymbol{3}$-bit signature register ($\boldsymbol{n = 3}$), with $\boldsymbol{\alpha}$ being a primitive element of $\boldsymbol{GF(2^3)}$, in particular, a root of a primitive polynomial $\boldsymbol{g(x) = x^3 + x + 1}$. Then, a symbolic scheme of Figure 3 will transfer to the logic level circuit of Figure 4, where

$$\boldsymbol{\alpha^l = \alpha_2^{(l)}x^2 + \alpha_1^{(l)}x + \alpha_0^{(l)}, \alpha_i^{(l)} \in \{0,1\}, 0 \le i \le 2, 0 \le l \le 6}$$

Eq. 6

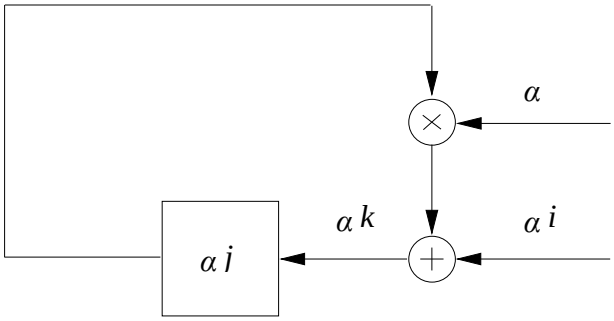


Figure 1 Built-in signature analysis of a circuit under test

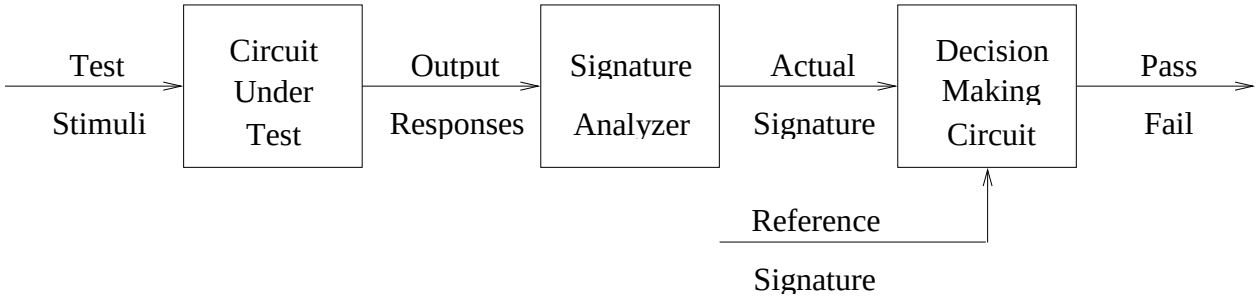


Figure 2 A t-stage polynomial division circuit

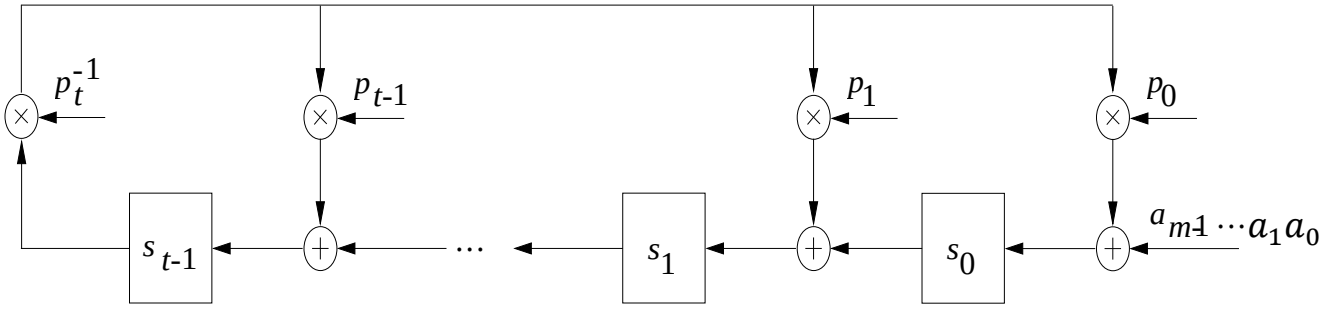


Figure 3 A symbolic presentation of a one-stage algebraic signature analyzer

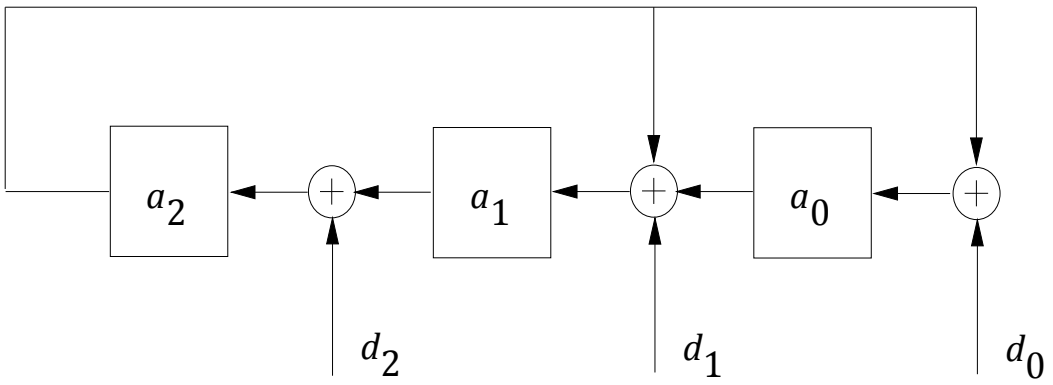


Figure 4 A logic level presentation of the algebraic 3-input signature analyzer

This expression indicates the relationship between the power and vector representations of a field element, as reflected in Table I ($\boldsymbol{where\ x\ =\ \alpha}$).

If the preliminary "cleared" analyzer receives, for example, the following sequence of **3**-bit output responses from a digital CUT, $\boldsymbol{\alpha^5, \alpha^6, \alpha^4, \alpha^2, \alpha^1, \alpha^0}$,then after the $\boldsymbol{6-th}$ shift its content will become:

$$(((((\mathbf{0}\cdot\boldsymbol{\alpha}+\boldsymbol{\alpha^5})\boldsymbol{\alpha}+\boldsymbol{\alpha^6})\boldsymbol{\alpha}\ +\ \boldsymbol{\alpha^4})\boldsymbol{\alpha}\ +\ \boldsymbol{\alpha^2})\boldsymbol{\alpha}+\boldsymbol{\alpha^1})\boldsymbol{\alpha}+\boldsymbol{\alpha^0}=\boldsymbol{\alpha}$$

Eq. 7

The power representation of the field element, $\boldsymbol{\alpha}$, corresponds to the vector representation, 010, which is the actual signature of the CUT.

In contrast to a digital CUT, the output responses of a mixed-signal CUT are distorted even in a fault-free case. Small permissible variations in the responses cause a significant deviation of the final signature. For example, if in the above

Table 1 THREE REPRESENTATIONS FOR THE ELEMENTS OF $GF(2^3)$ GENERATED BY $\boldsymbol{g(x) = x3 + x + 1}$. HERE $\boldsymbol{g(\alpha) = 0}$.

Power Representation α^l	Polynomial Representation $\alpha_2^{(l)}\alpha^2 + \alpha_1^{(l)}\alpha + \alpha_0^{(l)}\alpha$	Vector Representation $\alpha_2^{(l)}$	$\alpha_1^{(l)}$	$\alpha_0^{(l)}$
0	0	0	0	0
α^0	α^0	0	0	1
α^1	α^1	0	1	0
α^2	α^2	1	0	0
α^3	$\alpha^1 + \alpha^0$	0	1	1
α^4	$\alpha^2 + \alpha^1$	0	1	1
α^5	$\alpha^2 + \alpha^1 + \alpha^0$	1	1	1
α^6	$\alpha^2 + \alpha^0$	1	0	1

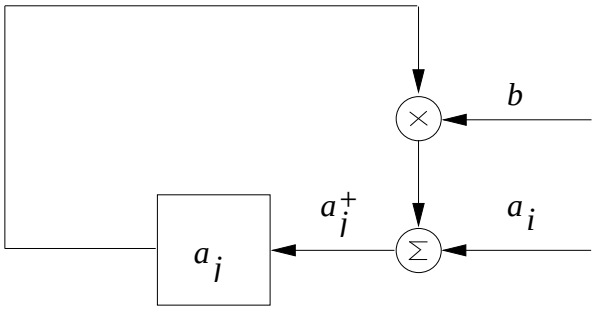


Figure 5 A symbolic presentation of a one-stage arithmetic signature analyzer

sequence of output responses the least significant bit in the first response changes from **1 *to* 0** (i.e. the vector $\mathbf{111}$ changes to $\mathbf{110}$, or power $\boldsymbol{\alpha^5}$ changes to $\boldsymbol{\alpha^4}$), then the actual signature will change from **010 *to* 101** (or from $\boldsymbol{\alpha}$ to $\boldsymbol{\alpha^6}$ in power form).

Apparently, the conventional SA represented in Figures 3 and 4 cannot be employed for mixed-signal circuits testing.

The known methods, output responses of mixed-signal circuits are compacted by a circuit referred to as a modulo adder (or accumulator, or digital integrator) [4]–[8]. It should be noted that a modulo adder is a special case of a ***residue computing circuit*** [14]. A residue computing circuit is represented in Figure 5. Here $\boldsymbol{a^j}$ is the current content of the register, $\boldsymbol{a^i}$ is the incoming (arithmetic) symbol and $\boldsymbol{b}$ is the base of the system. This circuit divides the incoming data sequence of symbols, $\boldsymbol{a_{m-1}, \ldots, a_1, a_0}$, treated as a number:

$$\boldsymbol{a = a_{m-1}b^{m-1} + \ldots + a_1 b + a_0}$$

Eq. 8

by the modulus

$$\boldsymbol{p = p_{t-1}b^{t-1} + \cdots + p_1 b + p_0, t \ll m}$$

Eq. 9

As in the case with the algebraic SA, we consider a singlestage device, i.e.

$$\boldsymbol{t = 1, p = p_0 < b = 2^n}$$

Eq. 10

where $\boldsymbol{n}$ is the number of bits occupied by the symbol. The residue, $\boldsymbol{s_0}$, constitutes a signature.

An operational cycle of the circuit in Figure 5 can be described by the expression:

$$\boldsymbol{a_j b + a_i = a_j^+ \ (modp)}$$

Eq. 11

Although the circuits of Figures 3 and 5 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.

As an example, Figure 6 represents the circuit that computes a modulo 5 residue of the incoming sequence of 3-bit symbols treated as an octal number [14]. Here $\boldsymbol{a_i}$ is the incoming octal digit and $\boldsymbol{C}$ is a combinational circuit which generates the following next state signals:

$$\boldsymbol{c_2 = \alpha_0^j \overline{\alpha_1^j} \alpha_0^i \alpha_1^i \alpha_2^i + \alpha_1^j \left(\alpha_0^j \alpha_1^i \oplus \alpha_2^i + \alpha_0^j \alpha_0^i \overline{\alpha_2^i} \right)}$$

Eq. 12

$$\boldsymbol{c_1 = \alpha_2^j \overline{\alpha_2^i} \left(\overline{\alpha_0^i} + \overline{\alpha_1^i} \right) + \overline{\alpha_2^j} \alpha_2^i \left(\overline{\alpha_0^j} \alpha_0^i + \overline{\alpha_1^j} \alpha_1^i \right) + \alpha_1^j \left(\overline{\alpha_0^j} + \overline{\alpha_2^i} \right) + \alpha_0^j (\alpha_1^i \oplus \alpha_2^i)}$$

Eq. 13

$$\boldsymbol{c_0 = \alpha_0^j \alpha_1^i \left(\overline{\alpha_1^j \oplus \alpha_2^i} \right) + \alpha_1^j \overline{\alpha_2^i} \left(\overline{\alpha_0^j} + \overline{\alpha_0^i \alpha_1^i} \right) + \alpha_2^j + \overline{\alpha_1^j} \alpha_2^i \left(\overline{\alpha_0^j} \alpha_1^i + \alpha_0^i \overline{\alpha_1^i} + \alpha_0^j \overline{\alpha_0^i} \right)}$$

Eq. 14

Each shift of this circuit implements the operation $\boldsymbol{a_j \times 8 + a_i (mod 5)}$.

In addition to high hardware complexity, the arithmetic compactor contains carry propagating circuitry (shown in red color in Figure 6) that delays the operation and aggravates the effect of a single fault.

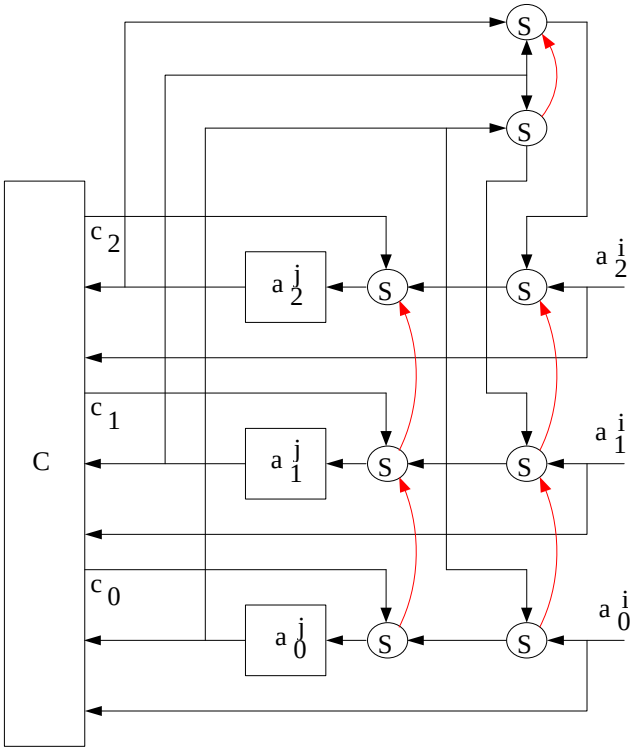


Figure 6 A 3-input arithmetic compactor

Below, we design an algebraic circuit that can be employed for mixed-signal data compaction. It does not contain carry propagating circuitry.

CHAPTER 3: A NOVEL METHOD

The polynomial (1) in conjunction with the reference signature can be considered as a code word of the code whose minimal distance is defined by the $\boldsymbol{g(x)}$. The distance here is the Hamming distance. This distance characterizes algebraic error-detecting properties of the code and is not convenient for arithmetic errors that occur in mixed-signal systems. Indeed, a small permissible deviation of the data to be compacted causes the reference signature to span the entire space. Under these conditions, the decision-making circuit in Figure 1 must be able to compare the actual signature with the entire set of possible reference signatures. This increases the analyzer complexity.

To decrease the complexity, an arithmetic SA treats the sequence of output responses from a mixed-signal circuit as a number (4). In conjunction with the reference residue, this is considered as a code word of an arithmetic error-control code. The properties of this code depend on the arithmetic minimal distance which in turn depends on the modulus $\boldsymbol{p}$. The arithmetic residue calculating analyzer does not search the entire space, since the space of arithmetic reference signatures is now contiguous. To make a decision, it employs a window comparator. This simplifies the circuitry. However, the hardware complexity of the arithmetic SA can still be quite high, as it was illustrated above.

In the rest of this paper, we will show how to design an algebraic SA, which generates a contiguous space of algebraic reference signatures.

In order to be contiguous, the space of signatures must be ordered. A signature can be represented in the vector or power forms. We will use the power exponent as the criterion for ordering the signature set. The distance between two vectors (signatures) will be evaluated as the arithmetic difference between the corresponding exponents. For example, the distance between the signatures **010** and **101** will be 5, because the exponents of powers $\boldsymbol{\alpha^6}$ and $\boldsymbol{\alpha}$ differ by **5**. We can interpret these

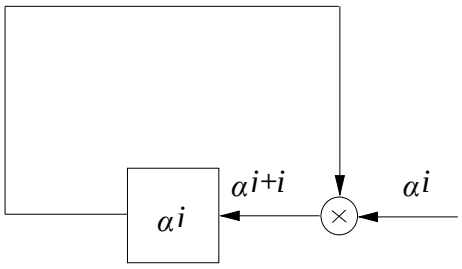


Figure 7 A symbolic form of an algebraic SA for a mixed-signal CUT

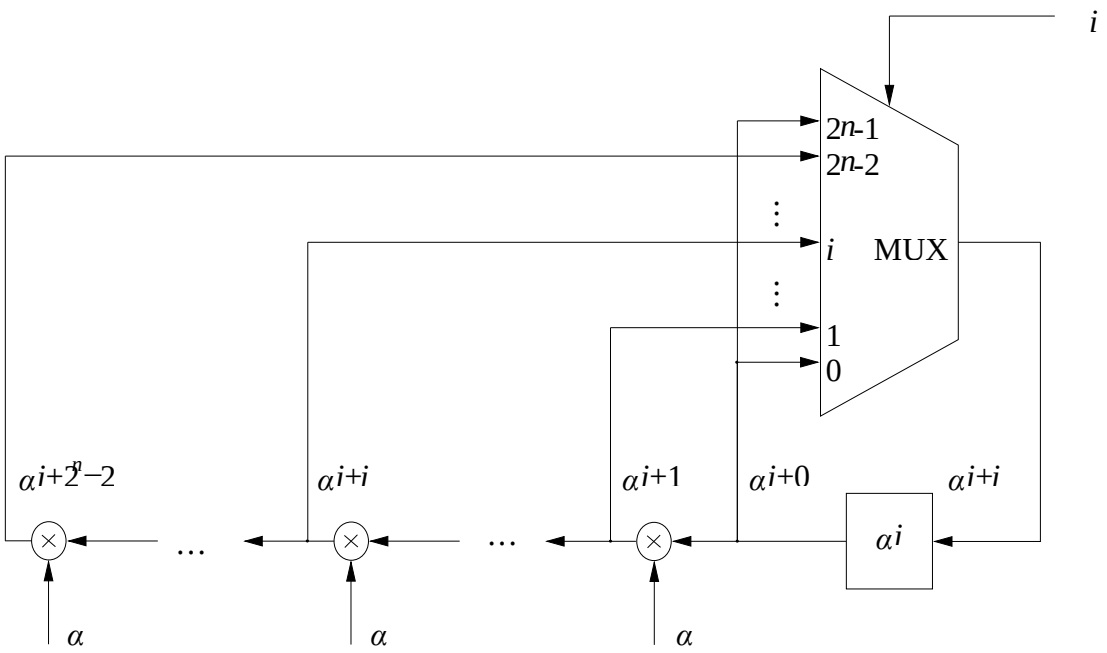


Figure 8 A more detailed symbolic form of the SA

exponents as output responses of a mixed-signal CUT, since they possess arithmetic properties. At the same time, the corresponding vectors (signatures) possess algebraic properties. Therefore, an arithmetic data is mapped into an algebraic data. Figure 7 represents the circuit which performs the mapping and computes an algebraic signature.

The circuit of Figure 7 can be obtained from the circuit of Figure 3 by the following transform:

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

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

Eq. 15

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

The logic level implementation of the circuit of Figure 7 is more complex compared to the circuit of Figure 3, but it is less complex than that of the circuit of Figure 5.

Prior to designing the circuit, we have to make a few observations.

The ***first*** observation is that

$$\alpha^j \alpha^i = \underbrace{\left(\ldots\left(\alpha^j \alpha\right)\alpha\right)\ldots\alpha}_{i}$$

Eq. 16

Let us denote 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.

$\boldsymbol{\alpha^i}$. Essentially, this means that the arithmetic values $\boldsymbol{i}$ are mapped into algebraic values $\boldsymbol{\alpha^i}$.

Based on these observations, we can design a signature analyzer in the way shown in Figure 8. Here $\boldsymbol{\alpha}$ is a primitive element of a finite field $\boldsymbol{GF(2^n)}$; $\boldsymbol{n}$ coincides with the bitlength of the output responses. The lower and upper inputs of the multiplexer in Figure 8 are connected together, since $\boldsymbol{\alpha^{2^n-1} = \alpha^0\ in\ GF(2^n)}$.

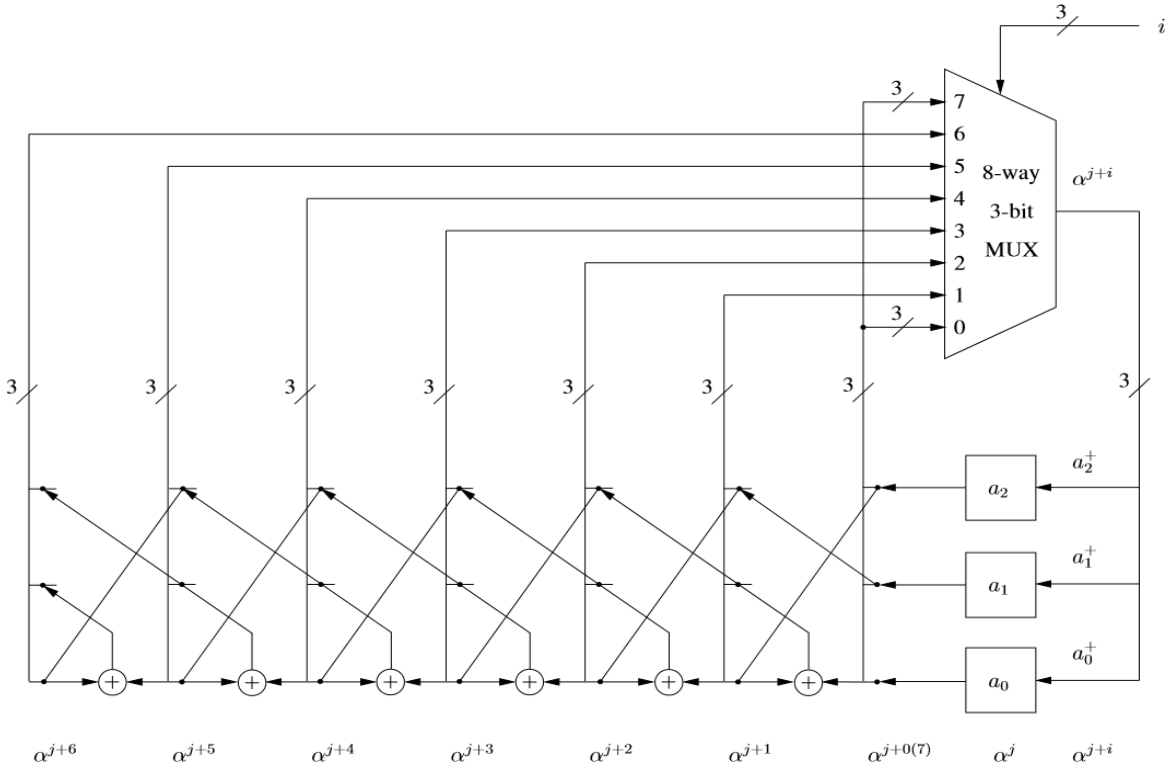


Figure 9 A register transfer level implementation of the SA

Considering the case when the analyzer is fed by $\mathbf{3}$-bit data, its more detailed implementation will have the form of Figure 9.

Here the buses consist of 3 lines, as indicated by the appropriate number. The initial content of the SA before the shift is $\boldsymbol{\alpha^j}$, or $\boldsymbol{a_2x^2 + a_1x + a_0}$ in the polynomial form (we have omitted the superscripts for the sake of simplicity). The notations $\boldsymbol{a_k}$ and $\boldsymbol{a_k^+}$, where $\boldsymbol{index\ k}$ can be one of the $\mathbf{0}, \mathbf{1}, \mathbf{2}$, indicate the present and next states, respectively.

A multiplier by $\boldsymbol{\alpha}$ in $\boldsymbol{GF(2^3)}$ is realized bearing in mind that

$\boldsymbol{g(x) = x^3 + x + 1}$, α corresponds to $\boldsymbol{x}$, and

$$(\boldsymbol{a_2x^2 + a_1x + a_0})\boldsymbol{x\ mod\ g(x)} =$$

$$(\boldsymbol{a_2x^3 + a_1x^2 + a_0x})\boldsymbol{mod\ g(x)} =$$

$$\boldsymbol{a_2(x + 1) + a_1x^2 + a_0x} =$$

$$\boldsymbol{a_1x^2 + (a_2 + a_0)x + a_2}$$

Eq. 17

This operation is shown by cross-lines in Figure 9. The multiplexer inputs "0" and "7" are tied together, because $\boldsymbol{\alpha^7 = \alpha^0}$ in the field $\boldsymbol{GF(2^3)}$.

In order to demonstrate how to use this analyzer, we will assume that it receives only two values from a CUT, in particular $\boldsymbol{j}$ *and* $\boldsymbol{i}$. Since the CUT is of a mixed-signal nature, there is an unavoidable (and thereby permitted) deviation of these values by $\pm\mathbf{1}$ (the greater tolerances can also be considered). The analyzer will map the received data into $\boldsymbol{\alpha^{j\pm 1}}$ and $\boldsymbol{\alpha^{i\pm 1}}$, respectively. If we assume that the initial content of the SA is 001 (versus 000 for a conventional SA), then after the first shift the content becomes $\boldsymbol{\alpha^{0}\alpha^{j\pm 1} = \alpha^{j\pm 1}}$. After the second shift, it changes to $\boldsymbol{\alpha^{j\pm 1}\alpha^{i\pm 1} = \alpha^{j+i\pm 2}}$. This expression is derived using the interval arithmetic rules.

It states that for the fault-free CUT the actual result must match one of the values from the interval $[\boldsymbol{\alpha^{j+i-2}, \alpha^{j+i+2}}]$, that is one of the following:

$$\boldsymbol{\alpha^{j+i-2}, \alpha^{j+i-1}, \alpha^{j+i}, \alpha^{j+i+1}, \alpha^{j+i+2}}$$

Eq. 18

To further simplify the SA operation, we will assume that instead of $\boldsymbol{\alpha^{0}}$ $(\boldsymbol{i.e.\,001})$ the initial SA content is $\boldsymbol{\alpha^{-(j+i)}}$. We will refer to this value as the $\boldsymbol{seed}$ value. Then, by the same reasoning, the SA content after two shifts will match one of the following powers:

$$\boldsymbol{\alpha^{-2}, \alpha^{-1}, \alpha^{0}, \alpha^{1}, \alpha^{2}}$$

Eq. 19

Due to the closure property of the field $\boldsymbol{GF(2^3)}$, this power set is equivalent to:

$$\boldsymbol{\alpha^5, \alpha^6, \alpha^0, \alpha^1, \alpha^2}$$

Eq. 20

Consequently, the decision-making circuit in Figure 3 will work as follows. If the actual signature does not match any value from the set (20), the CUT is considered to be faulty. Since these values are ordered (and surround the power $\boldsymbol{\alpha^0}$), the decision-making circuit can employ a comparator, thereby 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 as described in [16] and will coincide with the aliasing rate of the conventional analyzer.

CHAPTER 4: APPLICATION

Let us assume a $\mathbf{3}$-bit CUT, which is fed by two input stimuli. Under the fault-free operation, the CUT produces the output responses $\boldsymbol{j} = \mathbf{101} \pm \mathbf{1}$ and $\boldsymbol{i} = \mathbf{110} \pm \mathbf{1}$.Therefore, the seed value will be $\boldsymbol{\alpha^{-(j+i)} = \alpha^{-(5+6)} = \alpha^{-11} = \alpha^{3}, or\ 011}$ in the vector form. If the CUT is fault-free, then after $\mathbf{2}$ shifts the SA content must match one of the elements in the set (20). For example, if the actual responses are $\mathbf{101 + 1 = 110}$ $(\boldsymbol{or\ \alpha^{6}})$ $\boldsymbol{and}$ $\mathbf{110 + 1 = 111}$ $(\boldsymbol{or\ \alpha^{7}})$ (i.e. the variations are within the tolerance bounds), the signature will be $\boldsymbol{\alpha^{3}\alpha^{6}\alpha^{7} = \alpha^{2}}$ which belongs to the set (20). And the decision-making circuit will generate a pass signal. The validity of such a decision is determined by the aliasing rate.

Let us assume that a fault in the CUT has made the following changes in the output responses: $\mathbf{110 \rightarrow 011}(\boldsymbol{\alpha^{6} \rightarrow \alpha^{3}})\boldsymbol{and}$ $\mathbf{111 \rightarrow 100}$ $(\boldsymbol{\alpha^{7} \rightarrow \alpha^{4}})$. Then the actual signature will become $\boldsymbol{\alpha^{3}\alpha^{3}\alpha^{4} = \alpha^{3}}$. This element does not belong to the set (20), so the fault is detected.

There are two distinct ways of designing the decision-making circuit depending on the optimization criteria (time or hardware overhead).

4.1: HARDWARE OVERHEAD

If performance is paramount and time overhead is not desirable, the following approach can be employed. Let $\boldsymbol{m}$ be the number of output responses. All of the $\boldsymbol{2m+1}$ $\boldsymbol{\alpha}$-multiplier outputs (see Figure 8) that belong to the set (20), are connected to the first inputs of the $\boldsymbol{2m+1}$ comparators of a similar type. The second inputs of these comparators are shared and fed by the vector $\boldsymbol{0\ldots01}$. If the CUT is fault-free, one of the comparators will produce a logic "$\boldsymbol{1}$" signal. The logic OR of the comparator outputs will constitute a $\boldsymbol{pass/fail}$ signal.

The above procedure is based on the fact that the fault-free CUT produces one of the signatures from the set (20). If the actual signature is $\boldsymbol{\alpha^0}$, the comparator connected directly to the signature register produces a logic "$\boldsymbol{1}$", thus indicating that the CUT is fault free. If the actual signature is $\boldsymbol{\alpha^6}$, then the product $\boldsymbol{\alpha^6\alpha}$, generated at the output of the first $\boldsymbol{\alpha}$-multiplier equals to $\boldsymbol{1}$, which is detected by the next comparator. The same

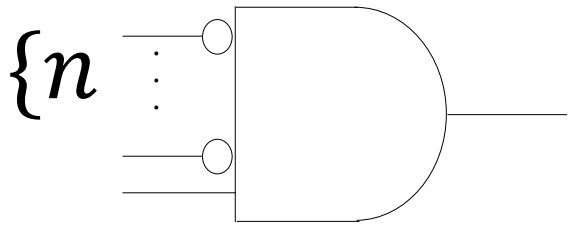


Figure 10 An n-bit comparator

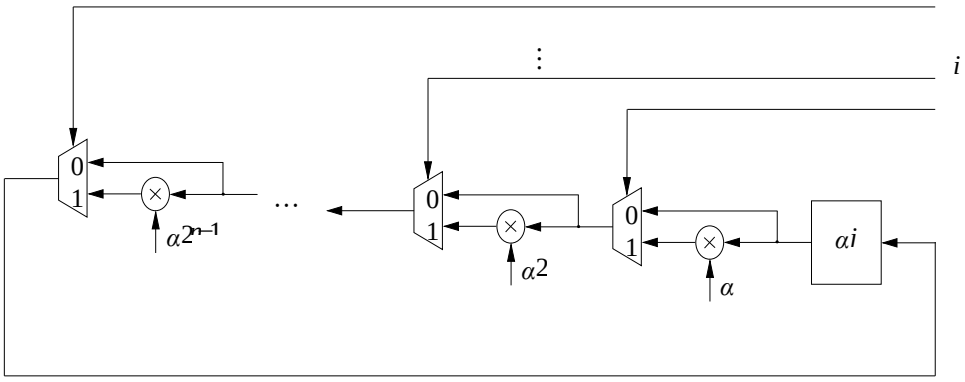


Figure 11 A binary-weighted version of the SA

reasoning applies to the rest of signatures from the set (20). The logic diagram of the $\boldsymbol{n}$-bit comparator is shown in Figure 10.

4.2: TIME OVERHEAD

If time overhead is allowed, the hardware complexity can be further reduced. In terms of implementation, it is more convenient to use the following seed value: $\boldsymbol{\alpha^{-(j+i+m+1)}}$ where $\boldsymbol{m}$ is the number of output responses. For the above example, $\boldsymbol{\alpha^{-(11+3)} = \alpha^{0}}$, and the set (20) will transform to:

$$\boldsymbol{\alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6}$$

Eq. 21

After the last output response, has been shifted in, the SA continues to shift its content $\boldsymbol{2m + 1}$ more times, while the input $\boldsymbol{i}$ is forced to $\boldsymbol{1}$. This ensures that the SA content is multiplied by $\boldsymbol{\alpha}$ with each shift. For the above example, $\boldsymbol{2m + 1 = 5}$. If within this time, the match with an element of the set (21) 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 $\boldsymbol{2m + 1}$ shifts, the output of the multiplexer in Figure 8 will go through the power $\boldsymbol{\alpha^0}$ or vector $\boldsymbol{0\ldots01}$. The match with the vector $\boldsymbol{0\ldots01}$ is detected by the comparator of Figure 10 connected to the multiplexor's output. The comparator output is actually producing a $\boldsymbol{pass/fail}$ signal.

The implementation complexity of the circuit of Figure 8 increases significantly with the growth of the data width, $\boldsymbol{n}$. Therefore, this circuit can only be implemented for the output responses with relatively low values of $\boldsymbol{n}$. For greater values of $\boldsymbol{n}$, we will modify the circuit of Figure 8 to the one shown in Figure 11. The modified circuit contains binary-weighted stages and is more economical in terms of hardware. The complexity of the multiplier $\times \boldsymbol{\alpha^i}$ is comparable with that of the multiplier $\times \boldsymbol{\alpha}$, whereas the number of multipliers drops from $\boldsymbol{2^n}$ to $\boldsymbol{n}$. The economy increases with the growth of $\boldsymbol{n}$.

For the case of $\mathbf{3}$-bit data, the circuit of Figure 11 transfers to the one shown in Figure 12. This circuit operates much in the same way. The $\boldsymbol{\alpha^i}$-multipliers structure is determined from the following expressions:

$$\mathbf{x(a_2x^2 + a_1x + a_0)\ mod\ g(x) = a_1x^2 + (a_2 + a_0)x + a_2}$$

$$\mathbf{x^2(a_2x^2 + a_1x + a_0)\ mod\ g(x)\ =}$$

$$\mathbf{(a_2 + a_0)x^2 + (a_2 + a_1)x + a_1}$$

$$\mathbf{x^4(a_2x^2 + a_1x + a_0)\ mod\ g(x) =}$$

$$\boldsymbol{(a_2 + a_1 + a_0)x^2 + (a_1 + a_0)x + (a_2 + a_1)}$$

Eq. 22

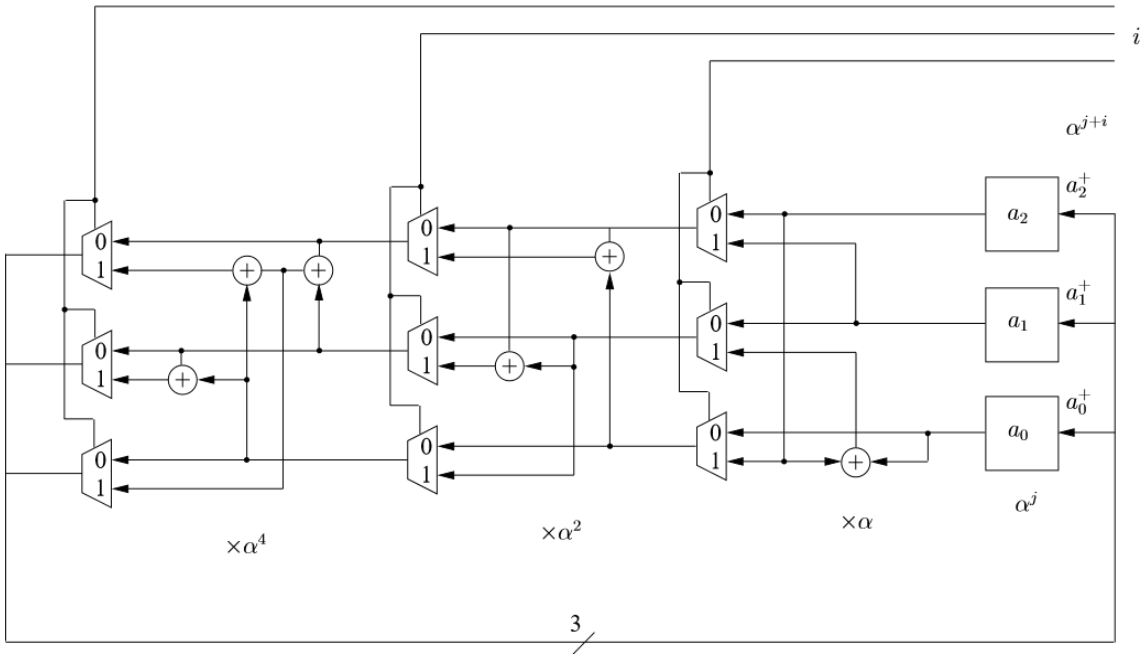


Figure 12 A register transfer level implementation of the 3-bit SA

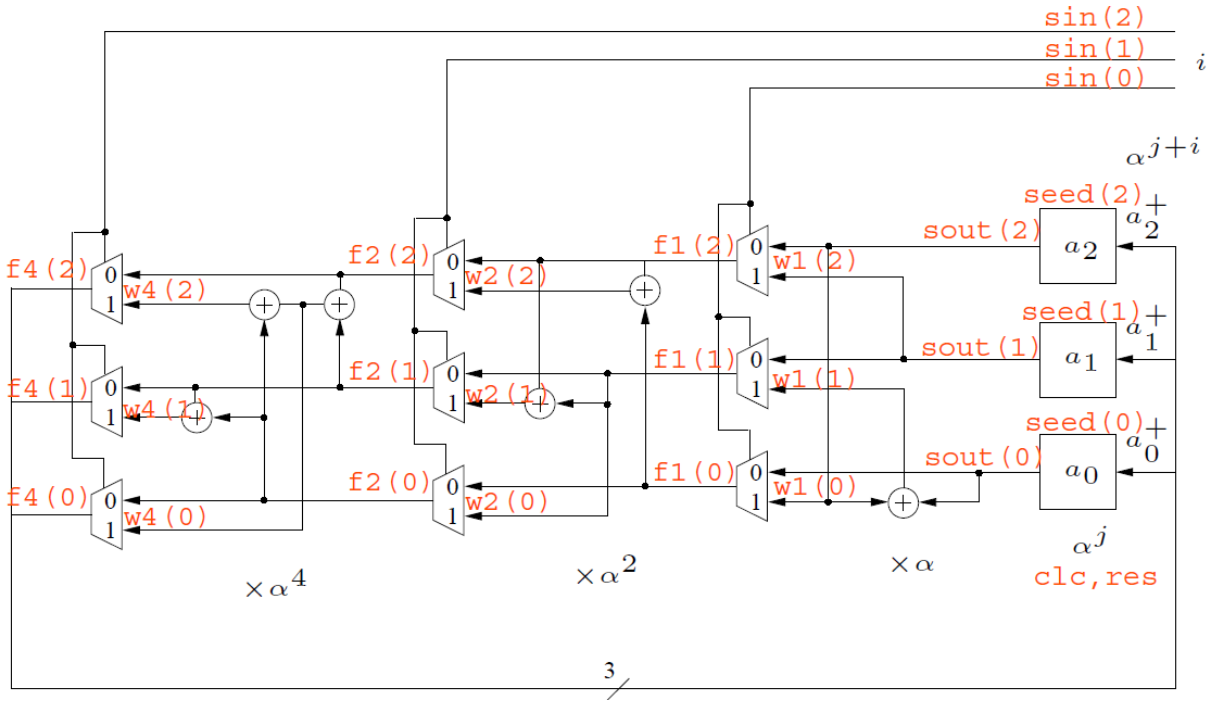


Figure 13 A 3-bit signature analyzer data flow

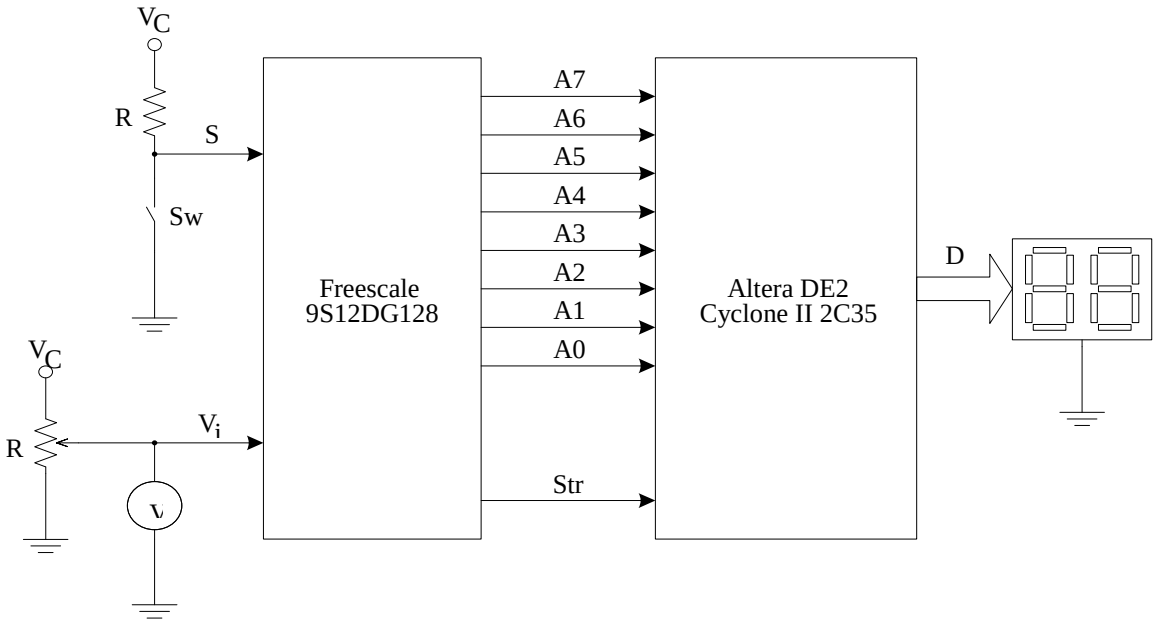


Figure 14 The experimental setup

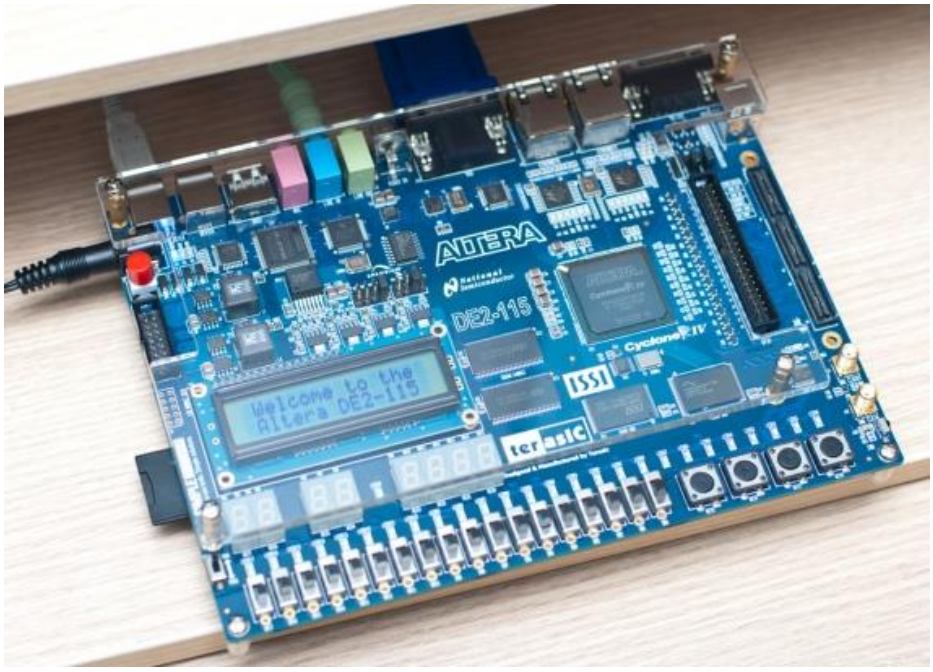


Figure 15 Altera DE2-115

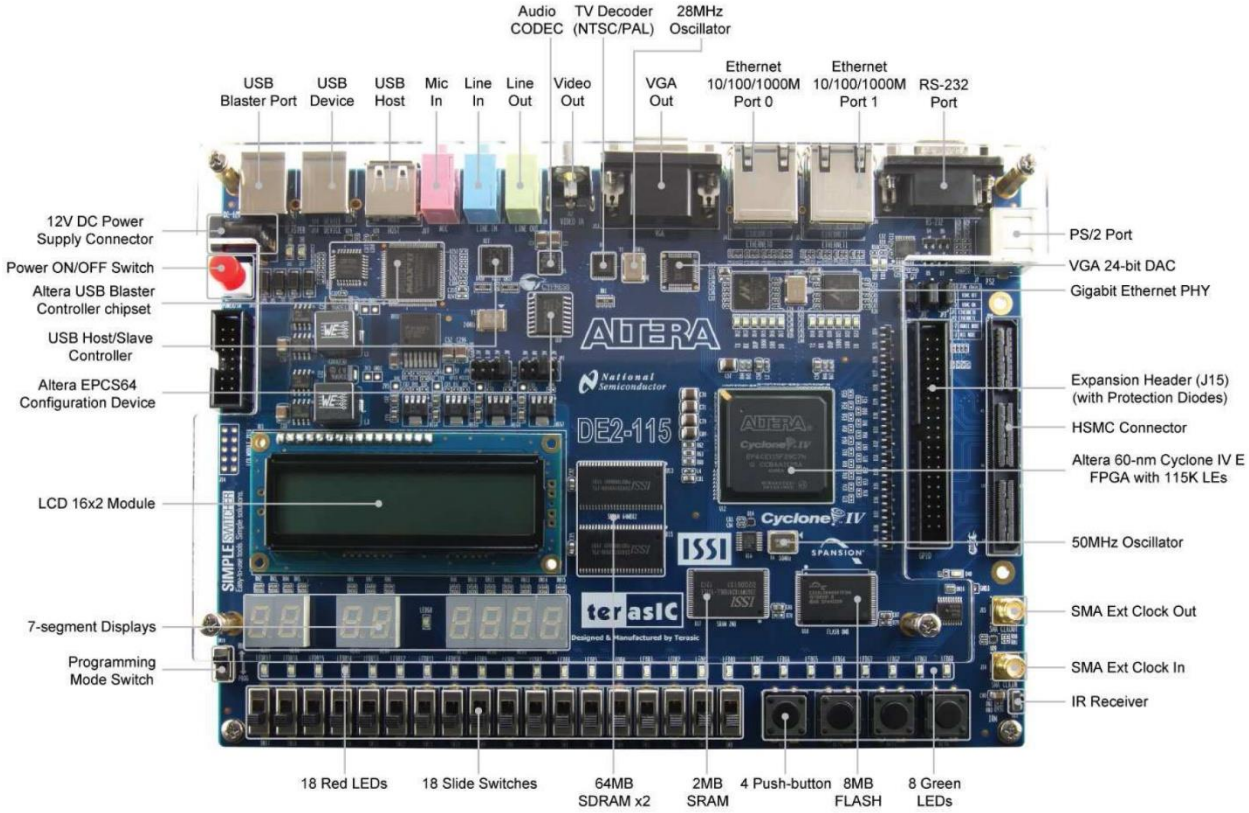


Figure 16 Altera DE2-115 Descriptions

CHAPTER 5: EXPERIMENTAL SETUP

The proposed method of signature analysis has set up for experimental setup to test the proposed method of signature analysis is shown in Figure 14. The setup includes the microcontroller system board Adapt9S12D (Technological Arts Inc.) based on the Freescale's 9S12DG128 microcontroller, and the Altera DE2 Development board based on the Cyclone II EP2C35F672C6 field-programmable gate-array (FPGA) device. We have selected 16 input test stimuli (voltages $\boldsymbol{V_{in}}$,) equally distributed over the range $(\mathbf{0} \sim \mathbf{5.12})\boldsymbol{V}$ and applied them to the analog-to-digital converter (ADC) of the 9S12 microcontroller (which served as a mixed-signal system). Each input voltage, $\boldsymbol{V_{in}}$, was measured by a high-precision voltmeter and regarded as a nominal test input value.

The circuit in Figure 14 operates as follows. Every time the switch S_w is closed, the system performs 8 measurements of the same test signal and averages the result by accumulating the sum of the eight $\mathbf{8}$-bit measurements and shifting it right three times, which eliminates noise. The ADC transfer characteristic is presented in Figure 16 [17]. According to this characteristic, each conversion result for a properly operating device can deviate from the nominal value by $\pm\mathbf{1}$, which is an implication of the fact that the permissible differential nonlinearity can range from $-\mathbf{0.5}\ \boldsymbol{to}\ +\ \mathbf{0.5}$ LSB (see shadowed boxes in Figure 16). For example, if $\boldsymbol{V_{in}} = \mathbf{40}\boldsymbol{mV}$, the conversion result can be $\mathbf{01}, \$\mathbf{02}\ \boldsymbol{or}\ \$\mathbf{03}$ (in the worst case,

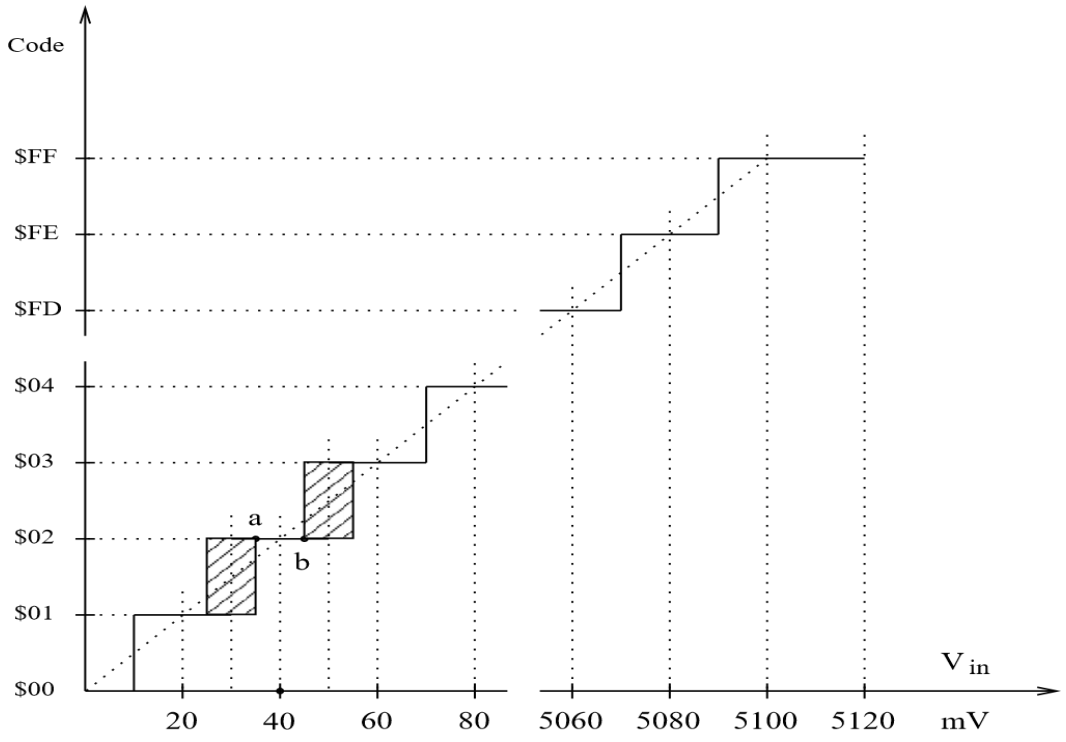


Figure 17 9S12 ADC Transer Function

the points $\boldsymbol{a}$ and $\boldsymbol{b}$ coincide). Therefore, each of the thirty-two $\mathbf{8}$-bit average results contain an error of at most $\pm\mathbf{1}$ 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 $\mathbf{2}$ possible values as opposed to $\mathbf{3}$ 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 s_{tr}. The s_{tr} signal serves as a $\boldsymbol{clock}$ for the state machine that implements the signature analyzer (in its $\mathbf{8}$-bit configuration). The signature, $\boldsymbol{D}$, is displayed on a two-digit $\mathbf{7}$-segment display in 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

$$\boldsymbol{g(x) = x^8 + x^4 + x^3 + x^2 + 1} \qquad \textbf{Eq. 23}$$

the seed value is calculated as follows.

$$\boldsymbol{4 + 20 + \ldots + 244 = 1984 = 199\ mod\ (2^8 - 1) = 199}$$
$$\boldsymbol{\alpha^{-199} = \alpha^{56} = 01011101}$$
$$\boldsymbol{SeedV\ alue = \alpha^{56}\alpha^{-16} = \alpha^{40} = 01101010 = 106}$$

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 $\mathbf{233}, \mathbf{250}, \mathbf{251}$ and $\mathbf{201}, \mathbf{234}, \mathbf{252}$ (in decimal form).

Table 2 RELATIONSHIP BETWEEN INPUT TEST STIMULI AND OUTPUT RESPONSES

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

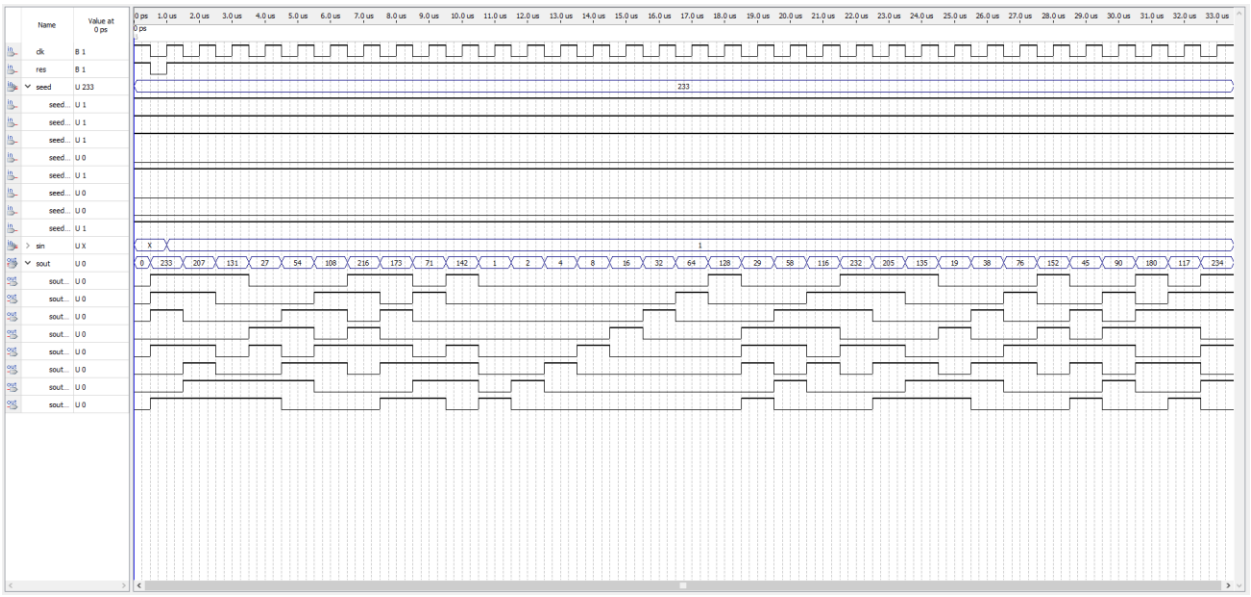


Figure 18 The combination ”1” is detected: ADC is operating properly on Seed 233

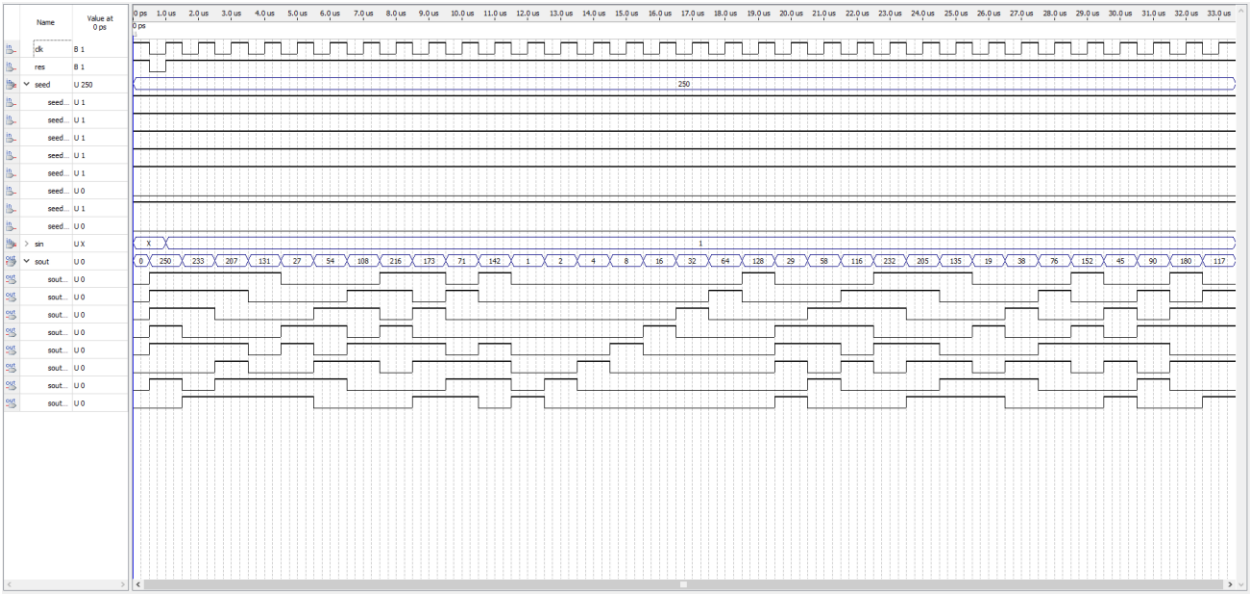


Figure 19 The combination ”1” is detected: ADC is operating properly on Seed 250

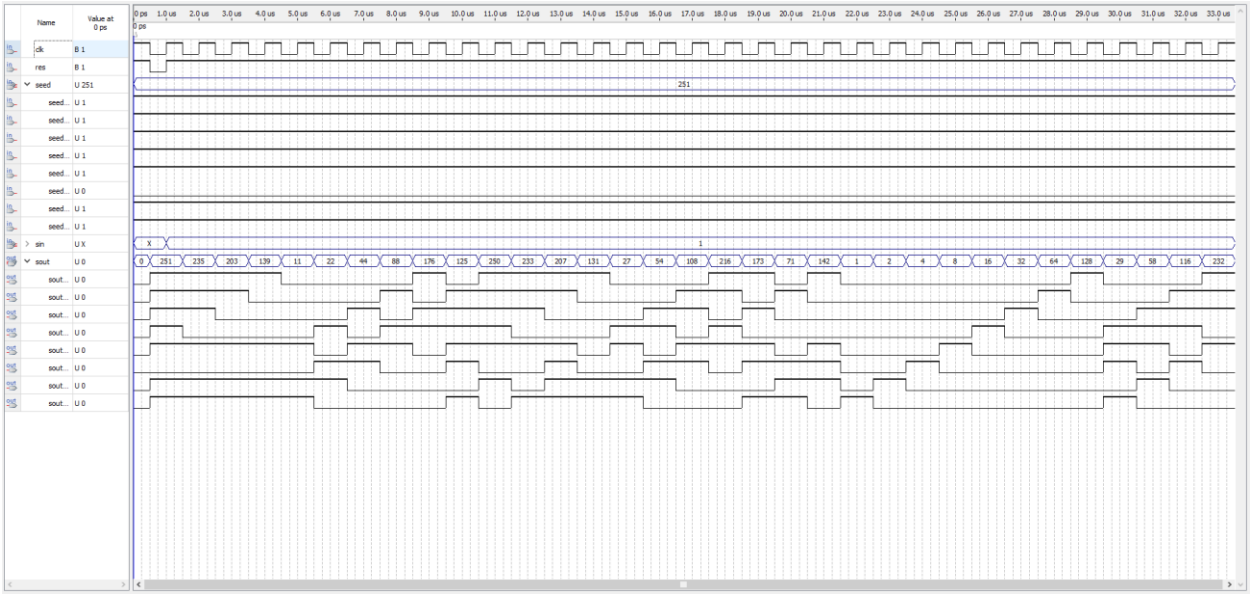


Figure 20 The combination ”1” is detected: ADC is operating properly on Seed 251

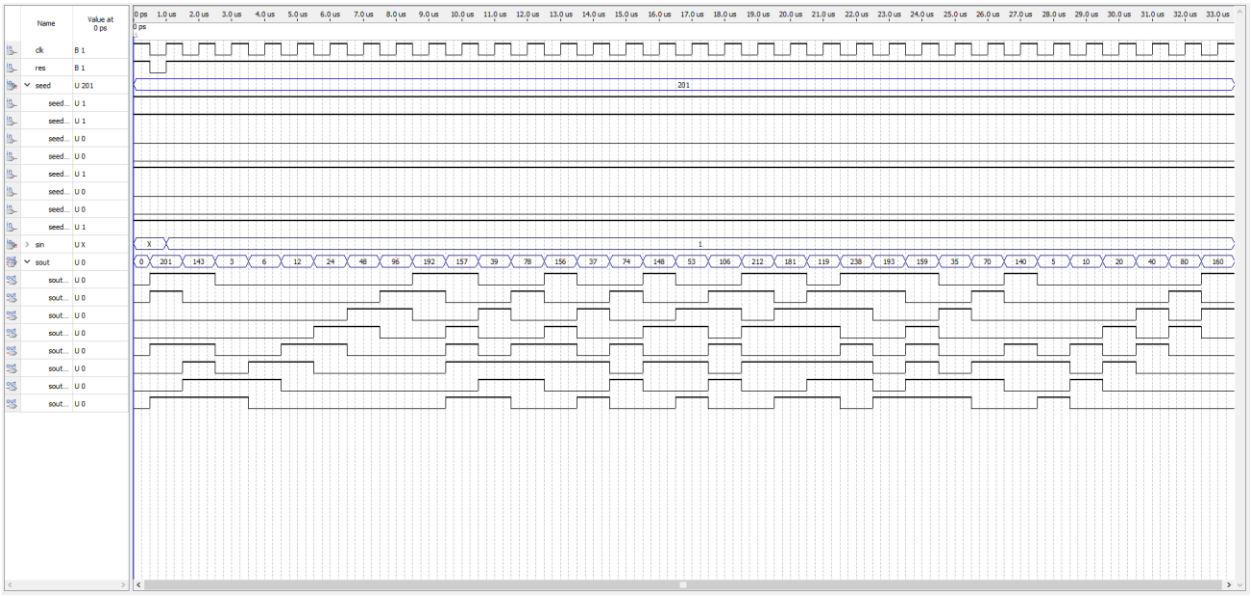


Figure 21 The combination ”1” is not detected: ADC is faulty on Seed 201

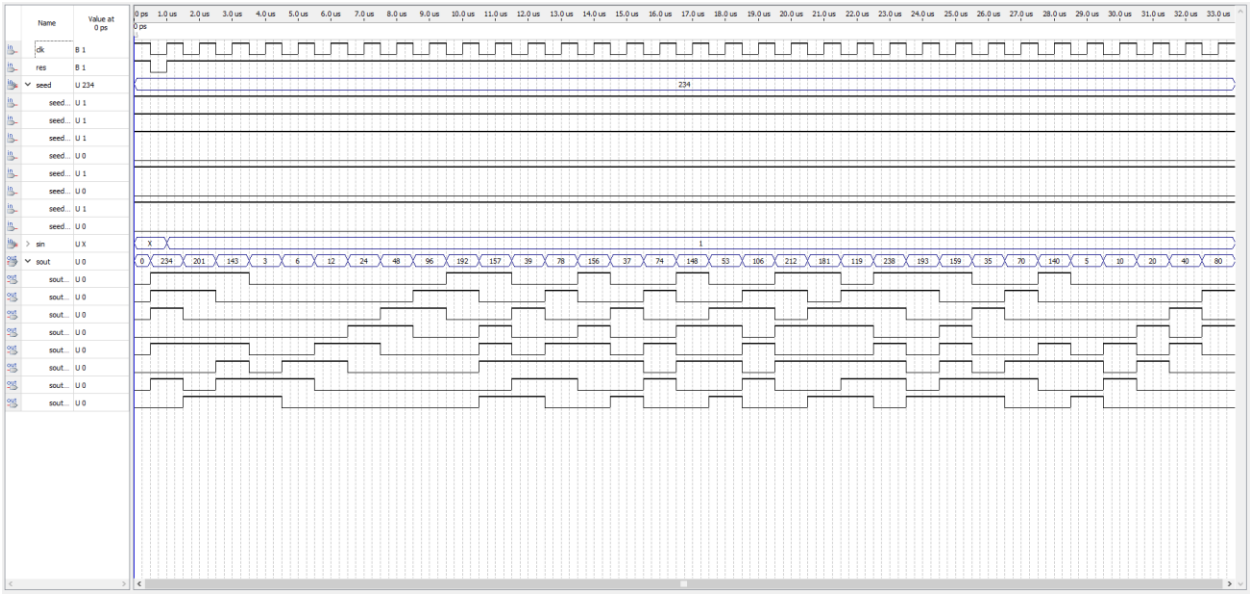


Figure 22 The combination ”1” is not detected: ADC is faulty on Seed 234

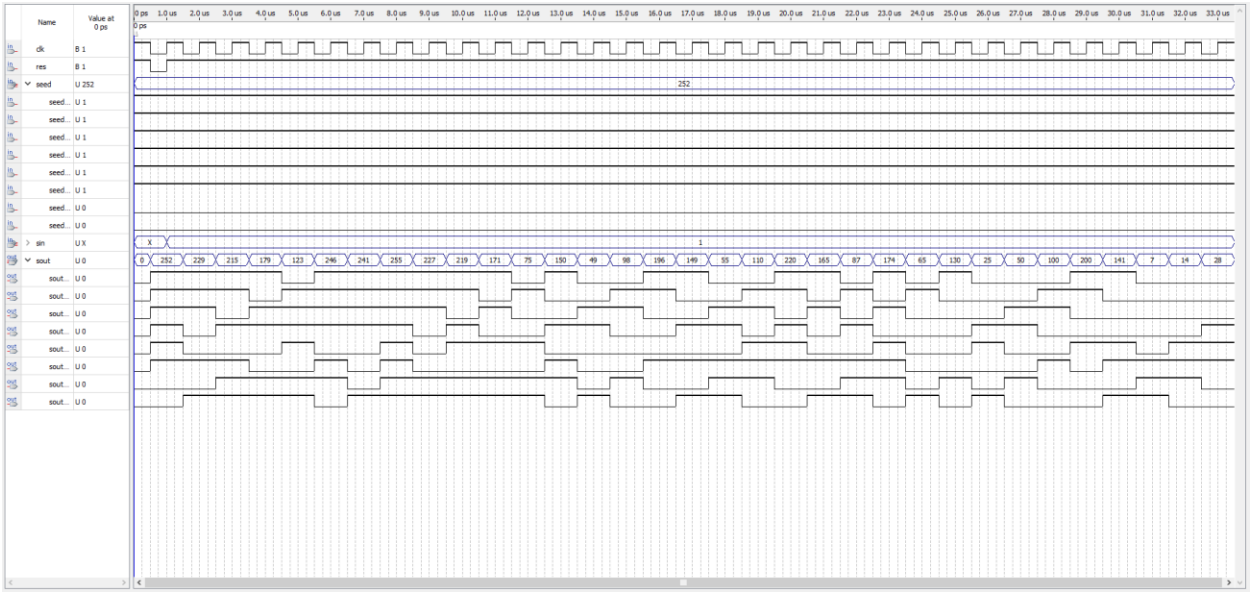


Figure 23 The combination ”1” is not detected: ADC is faulty on Seed 252

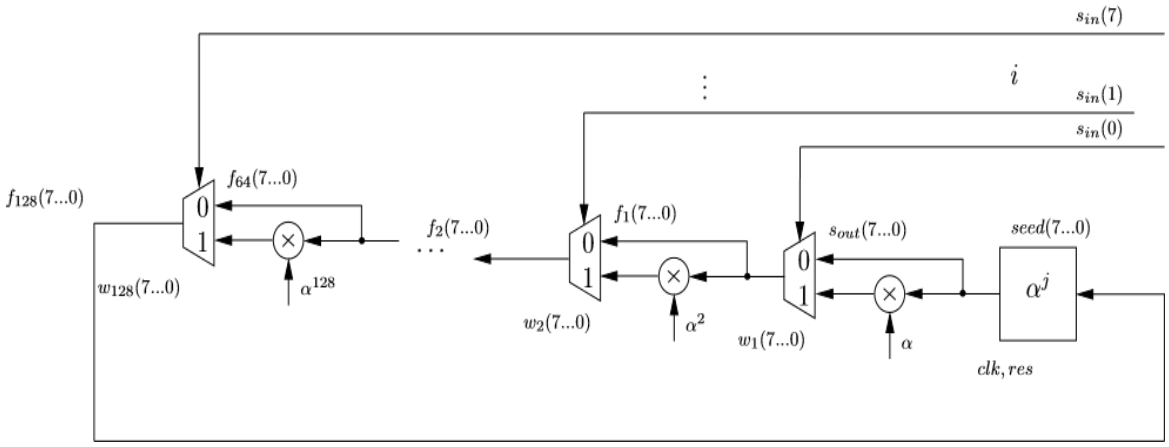


Figure 24 An 8-input signature analyzer

The process of calculation of these signatures is demonstrated in Figures 28 and 29. Figures 18, 19, 20 and 21, 22, 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.

CHAPTER 6: FUTURE WORK AND CONCLUSION

Mixed signal systems test and measurement using Signal Analyzer is complex and can require considerable knowledge and effort by designer for successful implementation. We examined an algebraic signature analysis method that can be employed for mixed-signal circuits testing. We demonstrated how to design the appropriate device. To simplify the hardware description model, hardware elements can be conditionally generated using simulation software. This device does not produce arithmetic carries and is therefore less prone to errors. The absence of carry propagating circuitry also contributes to the higher performance of the device.

Stimuli output for respective Seed for the fault-free or faulty system's signature were evaluated analytically.

The proposed scheme can also be used in arithmetic and algebraic error-control coding, as well as cryptography.

Future work to complement this work would be to implement the signature analyzer circuits in arithmetic error-control coding to verify a more accurate evaluation of time delay, overhead, and fault secure property of the circuit.

APPENDICES

A. VHDL CODE

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

ENTITY SigAnalyzer 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 SigAnalyzer;

ARCHITECTURE Behavior OF SigAnalyzer 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;
```

B. BLOCK DIAGRAM

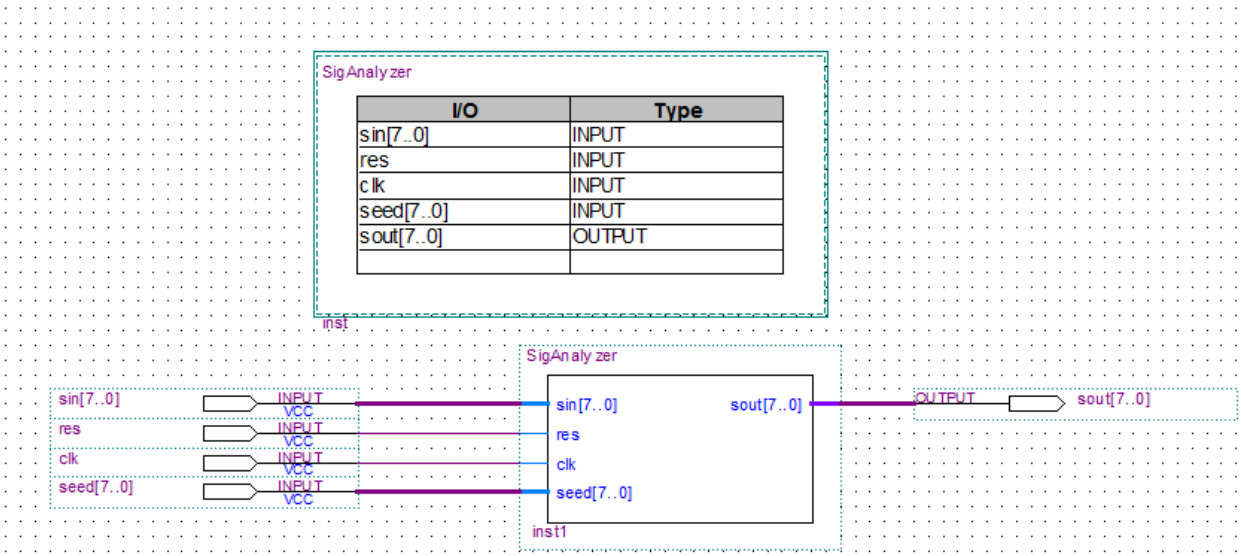


Figure 25 Block Diagram

C. PIN PLAN

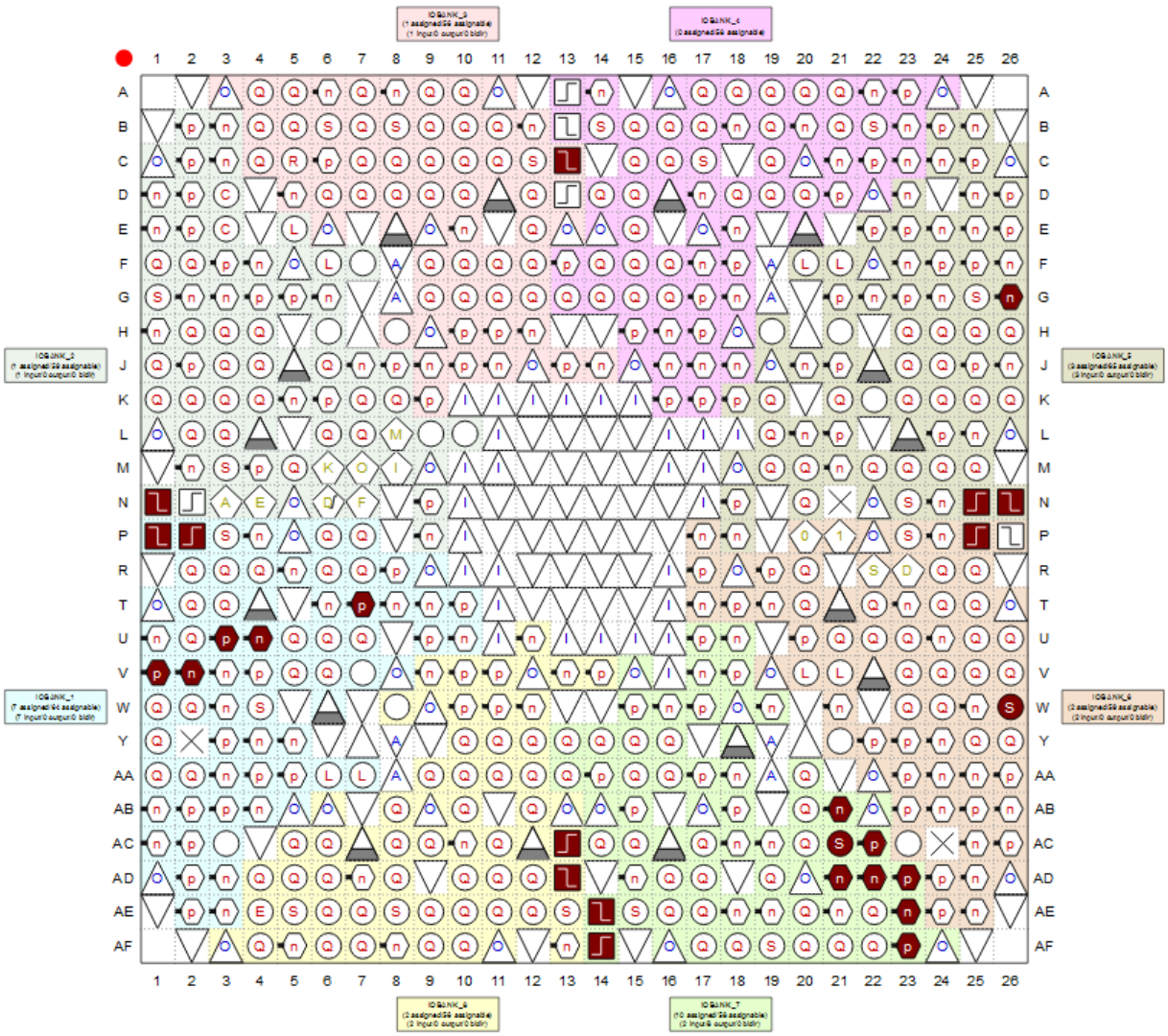
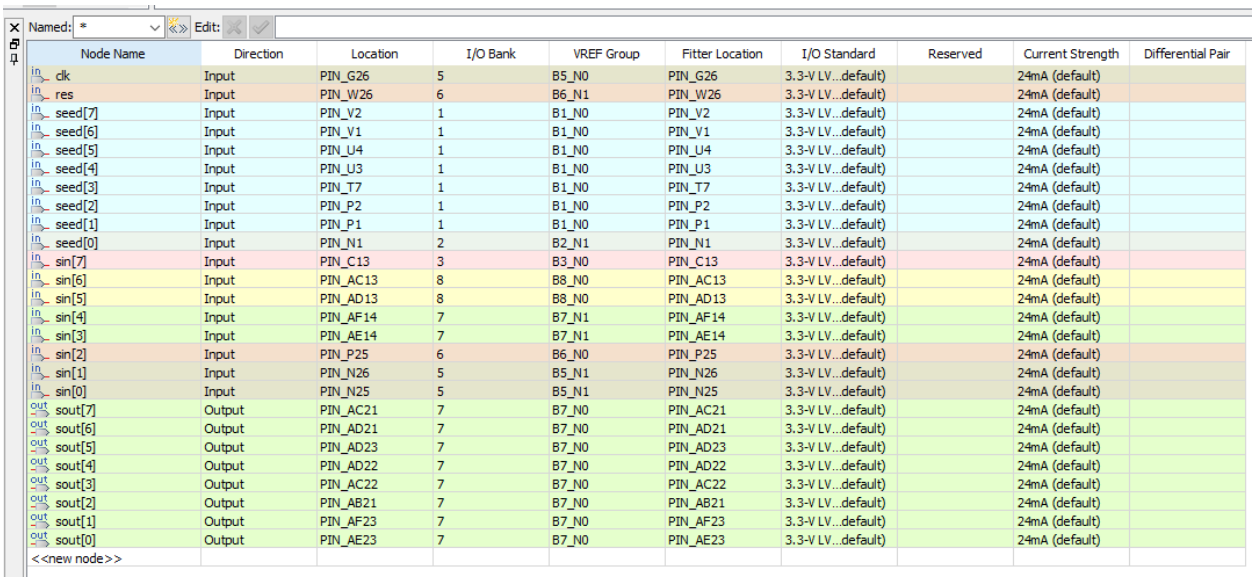


Figure 26 PIN Planner



Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_G26	5	B5_N0	PIN_G26	3.3-V LV...default)		24mA (default)	
res	Input	PIN_W26	6	B6_N1	PIN_W26	3.3-V LV...default)		24mA (default)	
seed[7]	Input	PIN_V2	1	B1_N0	PIN_V2	3.3-V LV...default)		24mA (default)	
seed[6]	Input	PIN_V1	1	B1_N0	PIN_V1	3.3-V LV...default)		24mA (default)	
seed[5]	Input	PIN_U4	1	B1_N0	PIN_U4	3.3-V LV...default)		24mA (default)	
seed[4]	Input	PIN_U3	1	B1_N0	PIN_U3	3.3-V LV...default)		24mA (default)	
seed[3]	Input	PIN_T7	1	B1_N0	PIN_T7	3.3-V LV...default)		24mA (default)	
seed[2]	Input	PIN_P2	1	B1_N0	PIN_P2	3.3-V LV...default)		24mA (default)	
seed[1]	Input	PIN_P1	1	B1_N0	PIN_P1	3.3-V LV...default)		24mA (default)	
seed[0]	Input	PIN_N1	2	B2_N1	PIN_N1	3.3-V LV...default)		24mA (default)	
sin[7]	Input	PIN_C13	3	B3_N0	PIN_C13	3.3-V LV...default)		24mA (default)	
sin[6]	Input	PIN_AC13	8	B8_N0	PIN_AC13	3.3-V LV...default)		24mA (default)	
sin[5]	Input	PIN_AD13	8	B8_N0	PIN_AD13	3.3-V LV...default)		24mA (default)	
sin[4]	Input	PIN_AF14	7	B7_N1	PIN_AF14	3.3-V LV...default)		24mA (default)	
sin[3]	Input	PIN_AE14	7	B7_N1	PIN_AE14	3.3-V LV...default)		24mA (default)	
sin[2]	Input	PIN_P25	6	B6_N0	PIN_P25	3.3-V LV...default)		24mA (default)	
sin[1]	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V LV...default)		24mA (default)	
sin[0]	Input	PIN_N25	5	B5_N1	PIN_N25	3.3-V LV...default)		24mA (default)	
sout[7]	Output	PIN_AC21	7	B7_N0	PIN_AC21	3.3-V LV...default)		24mA (default)	
sout[6]	Output	PIN_AD21	7	B7_N0	PIN_AD21	3.3-V LV...default)		24mA (default)	
sout[5]	Output	PIN_AD23	7	B7_N0	PIN_AD23	3.3-V LV...default)		24mA (default)	
sout[4]	Output	PIN_AD22	7	B7_N0	PIN_AD22	3.3-V LV...default)		24mA (default)	
sout[3]	Output	PIN_AC22	7	B7_N0	PIN_AC22	3.3-V LV...default)		24mA (default)	
sout[2]	Output	PIN_AB21	7	B7_N0	PIN_AB21	3.3-V LV...default)		24mA (default)	
sout[1]	Output	PIN_AF23	7	B7_N0	PIN_AF23	3.3-V LV...default)		24mA (default)	
sout[0]	Output	PIN_AE23	7	B7_N0	PIN_AE23	3.3-V LV...default)		24mA (default)	
<<new node>>									

Figure 27 PIN Identifications

D. WAVE FORMS

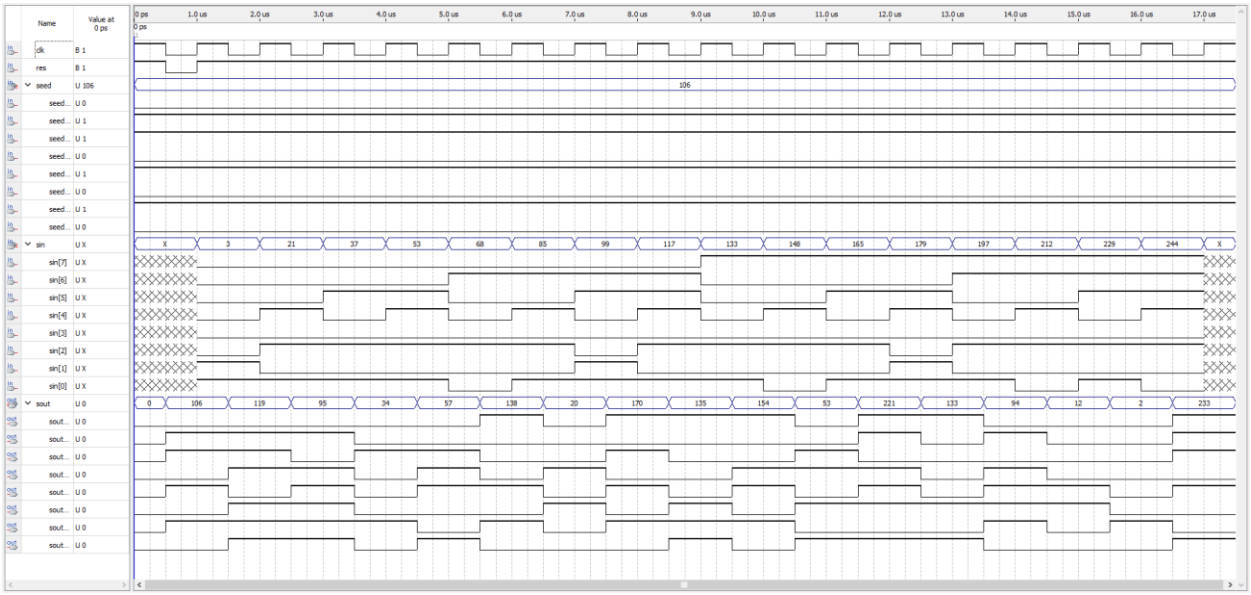


Figure 28 All output code deviations are within the tolerance bounds

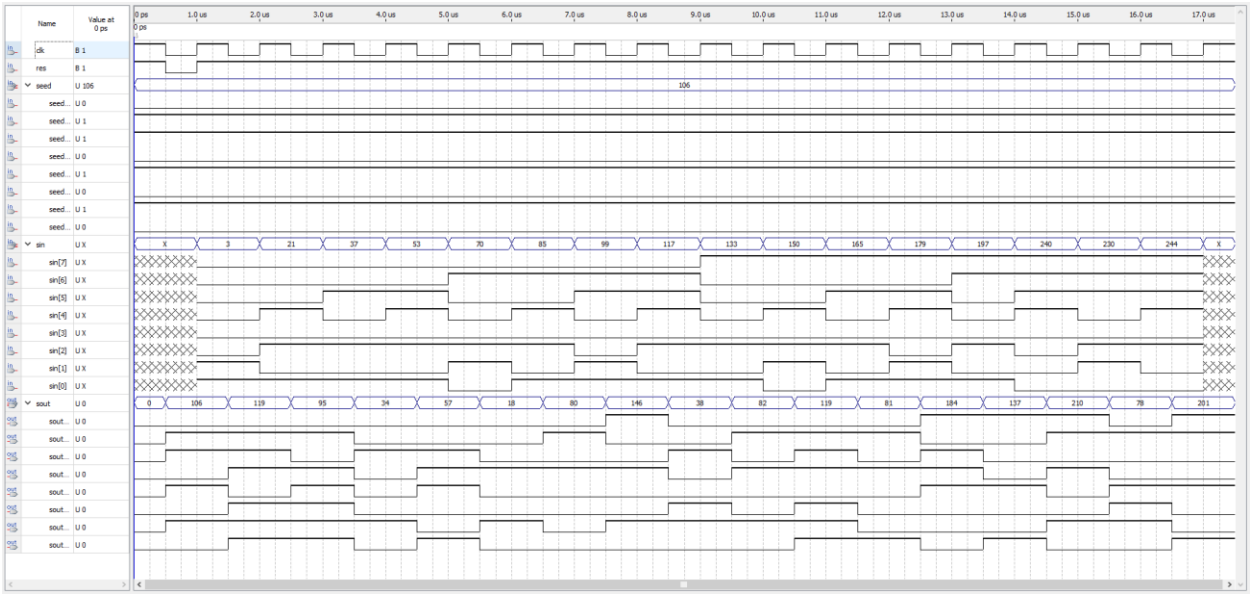


Figure 29 Some of the output code deviations exceed the tolerance bounds

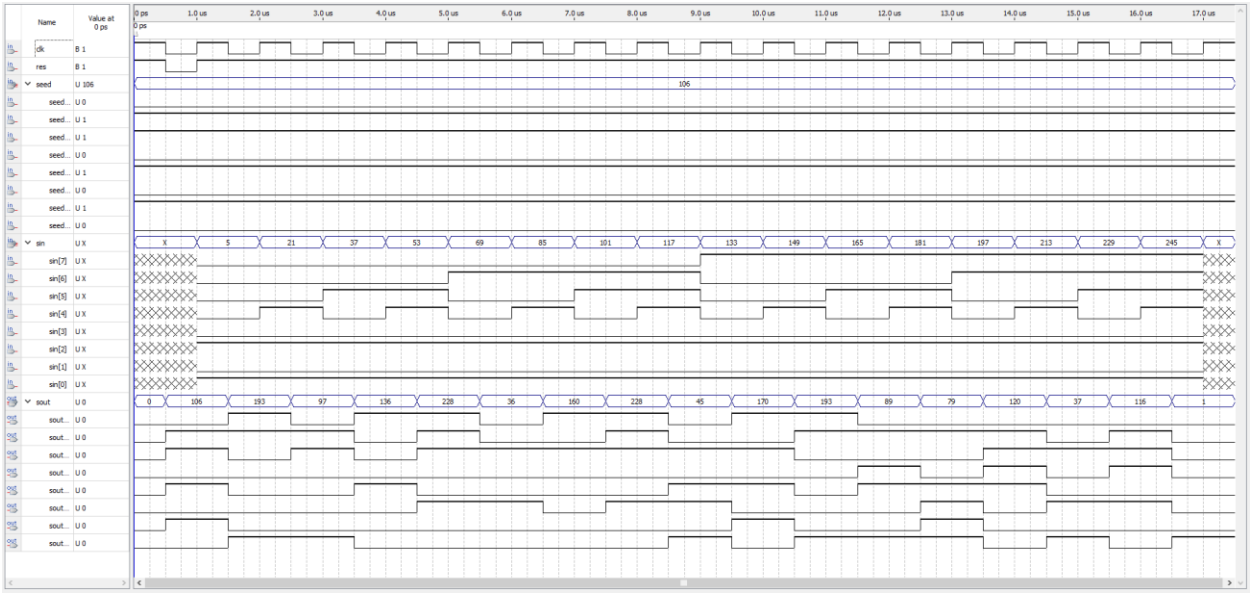


Figure 30 Some of the output code deviations in MAX

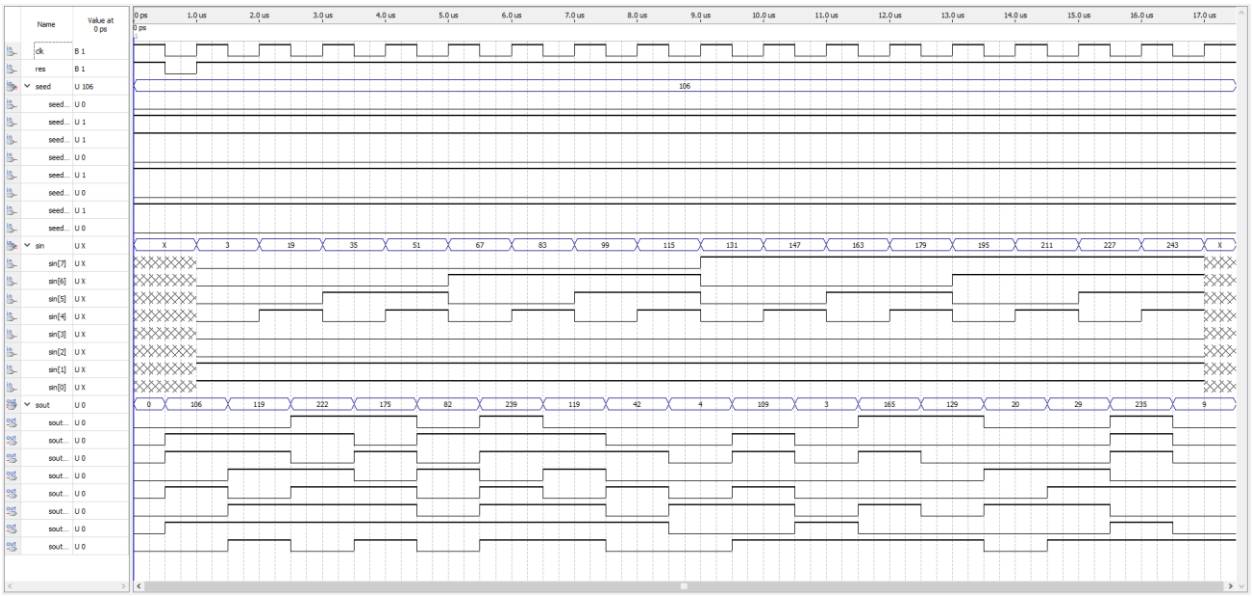


Figure 31 Some of the output code deviations in MIN

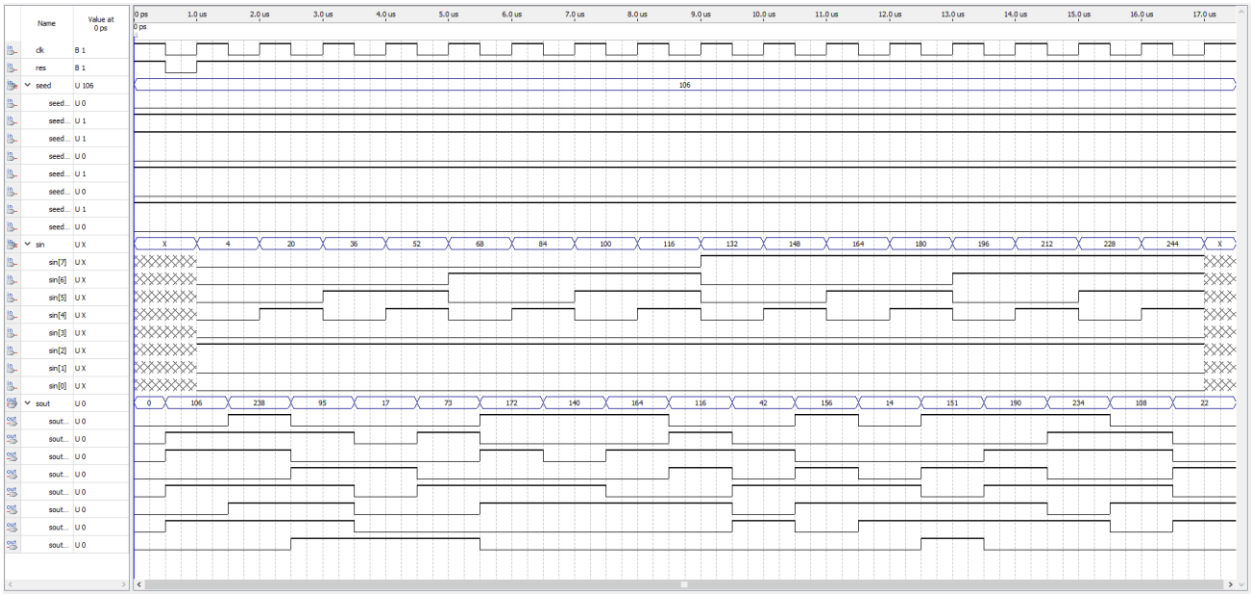


Figure 32 Some of the output code deviations in NOMINAL

REFERENCES

[1] R. Frohwerk, "Signature Analysis: A New Digital Field Service Method," *Hewlett Packard J.*, vol. 28, no. 9, pp. 2–8, 1977.

[2] G. Starr, Q. Jie, B. Dutton, C. Stroud, F. Dai, and Vector P. Nelson, "Automated Generation of Built-in Self-Test and Measurement Circuitry for Mixed-Signal Circuits and Systems," in Proc. *24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 11–19, 2009.

[3] D. K. Pradhan and S. K. Gupta, "A New Framework for Designing and Analyzing BIST Techniques and Zero Aliasing Compression," *IEEE Transactions on Computers*, vol. 40, no. 6, pp. 743–763, 1991.

[4] C. Stroud, J. Morton, T. Islam, and H. Alassaly, "A Mixed-Signal Built-in Self-Test Approach for Analog Circuits," in Proc. *Southwest Symposium on Mixed-Signal Design, 2003*, pp. 196–201, 2003.

[5] N. Nagi, A. Chatterjee, Y. Heebyung, and J. 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, pp. 540–546, 1998.

[6] N. Nagi, A. Chatterjee, and J. Abraham, "A Signature Analyzer for Analog and Mixed-Signal Circuits," in Proc. *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 284–287, 1994.

[7] S. Mir, M. Lubaszewski, V. Liberali, and B. Courtois, "Built-in Self-Test Approaches for Analogue and Mixed-Signal Integrated Circuits," in Proc. *38th Midwest Symposium on Circuits and Systems. Proceedings*, vol. 2, pp. 1145–1150, 1995.

[8] J. Rajski and J. Tyszer, "The Analysis of Digital Integrators for Test Response Compaction," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 293–301, 1992.

[9] W. Liu and J. Lei, "An Approach to Analog and Mixed-Signal BIST based on Pseudo Random Testing," in Proc. *IEEE International Conference on Communications, Circuits and Systems*," pp. 1192–1195, 2008.

[10] F. Corsi, C. Marzocca, and G. Matarrese, "Defining a BIST-Oriented Signature for Mixed-Signal Devices," in Proc. *IEEE Southwest Symposium on Mixed-Signal Design*, pp. 202–207, 2003.

[11] S. Demidenko, V. Piuri, V. Yarmolik, and A. Shmidman, "BIST Module for Mixed-Signal Circuits," in Proc. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (Cat. No.98EX223)*, pp. 349–352, 1998.

[12] T. Damarla, "Implementation of Signature Analysis for Analog and Mixed Signal Circuits," *U.S. Patent US6367043 B1*, Apr. 2, 2002.

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

[14] V. Geurkov, "Optimal Choice of Arithmetic Compactors for Mixed-Signal Systems," in Proc. *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 182–186, 2012.

[15] S. Lin and D. Costello, Error Control Coding. Upper Saddle River, *NJ: Pearson Education, Inc.*, 2004.

[16] V. Geurkov, V. Kirischian, L. Kirischian, and R. Sedaghat, "Concurrent Testing of Analog-to-Digital Converters," *I-manager's Journal on Electronics Engineering*, vol. 1, no. 1, pp. 8–14, 2010.

[17] MC9S12DT128 Device User Guide, *Motorola Inc*, V02.11. Rev. May 2004, [Online]. Available: (http://www.cse.chalmers.se/~svenk/mikrodatorsystem/HC12/reference_manuals/9S12DT128/9S12DT128DGV2.pdf)