On the Theory and Design of Polynomial Division Circuits

Vadim Geurkov, *Senior Member, IEEE*

Abstract—We present a theory and design techniques for polynomial division circuits with the primary focus on testing of digital and mixed-signal devices. We estimate the aliasing rate for the proposed circuits (signature analyzers) and show how to improve it. Two types of design techniques are examined for mixed-signal circuit analyzers that are arithmetical by nature. The techniques are scalable and valid for an arbitrary size and base of the number system. The proposed devices have both low hardware complexity and aliasing rate. The design techniques and devices can also be used in general arithmetic/algebraic error-control coding, cryptography, digital broadcasting and communication.

Index Terms—Polynomial division circuit, signature analyzer, algebraic error-control code, arithmetic error-control code, digital system test, mixed-signal system test, cryptography, digital broadcasting.

1 Introduction

The main objective of error-control coding is to insure the reliable delivery of digital information over unreliable channels. Soon after its inception, error-control coding proliferated to other fields of engineering and technology. Further to safeguarding the communication and storage channels, error-control codes have been applied to protect data processing. This has resulted in various forms of fault tolerant design and design for testability. Error-control techniques have also been exploited in cryptography.

Error-control codes are divided into *error-detecting* and *error-correcting* ones, according to the task they perform. Normally, error-detection circuitry constitutes a part of error-correction circuitry, implying that errors are first detected and then corrected. One of the main units of the error-detection circuitry is a *polynomial division circuit* (PDC). This circuit computes a remainder after dividing one polynomial by another (fixed) polynomial [1], [2]. The remainder is then evaluated to detect errors in the first polynomial. Due to good error detection capabilities and small hardware overhead, PDCs have been widely used for digital and mixed-signal systems testing [3]–[8]. Most of these PDCs belong to a limited set of special cases. Design procedures for them are well investigated and presentaed in [9]–[12].

A vast variety of test applications for contemporary VLSI circuits utilize multiple scan chains with the objective to detect a malfunction of the circuit. The output responses received in the chains are subsequently compacted by an algebraic PDC [13]–[17]. More advanced applications use an error-correcting code to diagnose the faulty chains and respective parts of the circuit under test [18]–[20]. PDCs are utilized in both error-detecting and error-correcting codes.

Although PDCs have been well researched, their design process can yet be improved leading to creation of new circuits with useful properties. In contrast to existing methods that are aimed toward special cases, we consider an arbitrary PDC operating in an arbitrary radix.

PDCs can be classified into two types according to interpretation of the data on which they operate, specifically *arithmetic* and *algebraic* circuits. We will respectively refer to these circuits as *residue* computing circuits (RsCCs) and *remainder* computing circuits (RmCCs). The more economical RmCCs are mostly used to test logic devices, whereas the RsCCs are aimed to arithmetic devices. As arithmetic devices are built of logic gates, the RmCCs can potentially be employed for arithmetic devices. However, arithmetic devices contain carry propagating circuits that propagate errors just as well as carries. Any single error arising in these circuits instantly turns into a multifold error. If the objective is to detect errors only, the complexity of the tester for single and multifold errors is the same. But in case of self-recovery applications, the tester realizes an error-correcting code. If the code is algebraic and thus uses an RmCC, its complexity grows significantly for multifold errors and may readily become impractical. And if the code is arithmetic (and uses an RsCC), the multifold error produced by a carry propagating circuit can still be interpreted as a single (arithmetic) error, which reduces the tester complexity. Clearly, both RmCCs and RsCCs are essential for the use in testing applications and it is important to know how to design them.

In this paper, we investigate the theory and design of RsCCs and RmCCs. We also demonstrate similarity between the two classes of circuits. While these circuits are primary oriented to testing, they can also be employed in other error-control coding applications. We focus on *linear block* codes as most frequently used in testing. Since we apply coding techniques to testing, we only consider error detection procedures.

The rest of the paper is organized as follows. Section 2 demonstrates how error-control codes are used in test/fault-tolerance applications, and it also introduces error-control coding model as applied to testing. Sections 3 and 4 present theory and design methodologies for RmCCs and RsCCs respectively. Conclusion summarizes the results.

• *V. Geurkov is with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON, Canada, M5B 2K3. E-mail: vgeurkov@ee.ryerson.ca*

Manuscript received March 16, 2017; revised July 29, 2017.

2 Application of Linear Systematic Block Codes for Testing

In a *block* data transmission model, the information sequence from the source is divided into blocks of k q-ary symbols - *messages*, $u = (u_{k-1}, \ldots, u_0)$. The encoder transforms each message into a *codeword*, $v = (v_{n-1}, \ldots, v_0)$. The set of all codewords is called an (n, k) *block code*. Since we use binary signals, $q = 2^m$, where m is the width of the system bus. The codeword enters the channel and is corrupted by noise. The decoder transforms the resulting *received sequence* $\tilde{v} = (\tilde{v}_{n-1}, \ldots, \tilde{v}_0)$ into a sequence $\hat{u} = (\hat{u}_{k-1}, \ldots, \hat{u}_0)$ called the *estimated message*. The decoding strategy ensures that $\hat{u}$ is a replica of u. However, excessive noise may cause some *decoding errors*, resulting into $\hat{u} \neq u$. The implication may be twofold: (i) some errors are not detected (*aliasing* occurs); (ii) additional errors may be induced (this only happens during error correction). We only focus on error-detecting codes. For the circuits introduced, we estimate the probability of aliasing and show how to minimize it.

To reduce the encoding/decoding complexity, we restrict our consideration to *linear systematic* codes. A code is *linear* iff a linear combination of codewords is also a codeword. In a *systematic* code a codeword is divided into the k-digit *message* part and the $r = (n - k)$-digit *checking* part. Such a code is specified by a $k \times n$ generator matrix, G [21]. If $u = (u_{k-1}, \ldots, u_0)$ is the message, then the codeword is

$$\begin{aligned} v &= (u_{k-1}, \ldots, u_0) \cdot G = (v_{n-1}, \ldots, v_0) \\ &= (u_{k-1}, \ldots, u_0 \mid v_{r-1}, \ldots, v_0) \end{aligned} \tag{1}$$

The decoding procedure involves two steps. Let $\tilde{v} = (\tilde{v}_{n-1}, \ldots, \tilde{v}_0)$ be the received sequence. At the first step, the syndrome s of the sequence $\tilde{v}$ is computed with the use of a $r \times n$ parity-check matrix H (specified by G):

$$s = (\tilde{v}_{n-1}, \ldots, \tilde{v}_0) \cdot H^T = (s_{r-1}, \ldots, s_0) \tag{2}$$

If $s = 0$, the received sequence is assumed to be error-free and $\hat{u} = (\tilde{v}_{n-1}, \ldots, \tilde{v}_r) = u$. If $s \neq 0$, at the second step, the syndrome is used to compute the estimated message $\hat{u}$.

Since we only consider error detection, the decoder generates the syndrome (2) without subsequent computation of the estimated message.

The encoding/decoding complexity depends on the way how the parity-check digits/syndrome digits are computed. In time critical (e.g., self-checking) applications, the multiplication by G (H^T) is implemented by a combinational circuit. If time is not critical, a special form of the matrix G (H^T) can be used, which allows sequential execution (reducing complexity). Depending on the channel type, whether it is *algebraic* (data transmission/storage system) or *arithmetic* (data processing system), this special form of G (H^T) defines a *polynomial algebraic* code or a *polynomial arithmetic* code. In a polynomial code, the codewords are divisible by the *generator polynomial*. In case of arithmetic codes, the generator polynomial turns into an integer called the *generator*. A special case of a polynomial code is a *cyclic* code. Multiplication by G (H^T) in a cyclic algebraic/arithmetic code is equivalent to polynomial division by the generator polynomial/generator and is performed sequentially [1], [2]. Because of the more economical implementation, we assume cyclic codes for our applications. Consequently, the subject of our research is a sequential PDC.

Cyclic codes use polynomial representation of sequences of symbols, $f_{n-1}, \ldots, f_0$:

$$f(x) = f_{n-1}x^{n-1} + \ldots + f_1 x + f_0 \tag{3}$$

The coefficients of this polynomial are elements of a finite field, $GF(2^m)$, m being the system bus width. Likewise, in arithmetic codes, the coefficients (or *digits*) are integers from the set $\{0, 1, \ldots, b-1\}$, where b is the *base* of the number system. We use the base $b = 2^m$, so that the system bus state is characterized by an integer from the set $\{0, 1, \ldots, 2^m - 1\}$. Equation (3) then becomes:

$$f(b) = f_{n-1}b^{n-1} + \ldots + f_1 b + f_0 = f \tag{4}$$

Here f is the integer *value*, $0 \leq f \leq b^n - 1$.

A code consists of a subset of polynomials that form a *ring*. The ring can be constructed of type (3) polynomials (a ring of polynomials over a field), or type (4) polynomials (a ring of integers). Both ring types are special cases of a *Euclidean ring* [1].

Using polynomial representations (3) and (4), the encoding and decoding procedures (1) and (2) for systematic cyclic algebraic and arithmetic codes become respectively

$$\begin{aligned} v(x) &= u(x)x^r \oplus [u(x)x^r] \bmod g(x) \\ s(x) &= \tilde{v}(x) \bmod g(x) \end{aligned} \tag{5}$$

and

$$\begin{aligned} v(b) &= u(b)b^r - [u(b)b^r] \bmod g(b) \\ s(b) &= \tilde{v}(b) \bmod g(b) \end{aligned} \tag{6}$$

Equations (6) can be rewritten in terms of integer values:

$$\begin{aligned} v &= ub^r - (ub^r) \bmod g \\ s &= \tilde{v} \bmod g \end{aligned} \tag{7}$$

In equations (5) and (7), $g(x)$ is the generator polynomial of degree r, and g is the generator. Radix b representation of the generator occupies r digits. Multiplication of a message by x^r or b^r is equivalent to shifting it left r times.

The properties of an algebraic or arithmetic error-control code are defined by the structure of $g(x)$ or $g(b)$. Therefore, it is important to know how to design the circuits that implement operations (5) and (6) for an arbitrary form of the polynomials, $g(x)$ and $g(b)$.

Along with the codes specified by equations (5) and (7), there are other classes of codes that do not assume shifts of the message digits and are more convenient for testing. In these codes, the message digits and the parity-check digits are separated. Therefore, the unit being tested and the decoding unit operate independently. Such codes are called *separate codes*. Examples of these codes are given below.

In a separate *residue* arithmetic code [2], the codeword v for the message u is defined as

$$v = (u, \rho) \tag{8}$$

where $\rho = |u|_g = u \bmod g$ is the *residue* and g is the *modulus*.

The syndrome s is defined as

$$s = |\tilde{u} - \tilde{\rho}|_g \tag{9}$$

where $\tilde{u}$ and $\tilde{\rho}$ are *received* message and residue, respectively.

If the error control capability of a residue code is to be improved, the number of moduli is increased. In a *multiresidue* code [2], the codeword is defined analogously:

$$v = (u, \rho_1, \ldots, \rho_w) = (u, |u|_{g_1}, \ldots, |u|_{g_w}) \quad (10)$$

$$s = (s_1, \ldots, s_w) = (|\tilde{u} - \tilde{\rho}_1|_{g_1}, \ldots, |\tilde{u} - \tilde{\rho}_w|_{g_w}) \quad (11)$$

In a *residue number system* (RNS) code, the message u is not present in the codeword at all [22]. The codeword v is formed of z residues, w of which are enough to uniquely specify the message u. The remaining $z - w$ residues are redundant and used for error-control:

$$v = (\rho_1, \ldots, \rho_w, \rho_{w+1}, \ldots, \rho_z) \quad (12)$$

RNS codes drastically improve the system performance (in addition to improving its fault-tolerance).

The codes analogous to the ones specified by equations (8) - (12) have also been developed in algebraic domain. For example, the *polynomial RNS* [23] are defined as follows:

$$v(x) = [|u(x)|_{g_1(x)}, \ldots, |u(x)|_{g_z(x)}] = [\sigma_1(x), \ldots, \sigma_z(x)]$$

The Reed-Solomon codes can be regarded as a special case of polynomial RNS [23].

Polynomial multiresidue codes are defined as:

$$v(x) = [u(x), \sigma_1(x), \ldots, \sigma_w(x)]$$
$$s(x) = [|\tilde{u}(x) - \tilde{\sigma}_1(x)|_{g_1(x)}, \ldots, |\tilde{u}(x) - \tilde{\sigma}_w(x)|_{g_w(x)}]$$

Let us consider a polynomial residue code: $v(x) = [u(x), \sigma(x)]$, where $\sigma(x) = |u(x)|_{g(x)}$ is a remainder. The syndrome, $s(x) = |\tilde{u}(x) - \tilde{\sigma}(x)|_{g(x)} = |\tilde{u}(x)|_{g(x)} - \tilde{\sigma}(x)$, where $\tilde{\sigma}(x)$ is the *received remainder*. If this code is used for testing and the unit under test is fault free, $s(x) = 0$ and

$$|\tilde{u}(x)|_{g(x)} = \tilde{\sigma}(x) \quad (13)$$

Equation (13) applies to built-in (embedded) test systems. If the test system is external and its memory is reliable, then (13) transfers to

$$|\tilde{u}(x)|_{g(x)} = \sigma(x) \quad (14)$$

Introducing $\breve{\sigma}(x) = |\tilde{u}(x)|_{g(x)}$, we rewrite (13) and (14):

$$\breve{\sigma}(x) = \begin{cases} \tilde{\sigma}(x) & \text{for internal tester} \\ \sigma(x) & \text{for external tester} \end{cases} \quad (15)$$

The r-tuples composed of the coefficients of $\breve{\sigma}(x)$, $\sigma(x)$ and $\tilde{\sigma}(x)$ are referred to as *actual signature*, *reference signature* (*fault-free circuit's signature*) and *distorted reference signature*:

$$\breve{\sigma} = (\breve{\sigma}_{r-1}, \ldots, \breve{\sigma}_0);\ \sigma = (\sigma_{r-1}, \ldots, \sigma_0);\ \tilde{\sigma} = (\tilde{\sigma}_{r-1}, \ldots, \tilde{\sigma}_0)$$

In terms of signatures, equation (15) has the form:

$$\breve{\sigma} = \begin{cases} \tilde{\sigma} & \text{for internal tester} \\ \sigma & \text{for external tester} \end{cases} \quad (16)$$

If equation (16) does not hold, the circuit is certainly faulty. Otherwise, it is assumed to be fault free. To improve the aliasing rate, the number of residues can be increased [24]. Alternatively, while retaining a single residue, the degree r of the generator polynomial $g(x)$ can be raised.

If an *arithmetic (one)residue* code is used for testing, analogously to (15), for the fault-free circuit

$$\breve{\rho}(b) = \begin{cases} \tilde{\rho}(b) & \text{for internal tester} \\ \rho(b) & \text{for external tester} \end{cases} \quad (17)$$

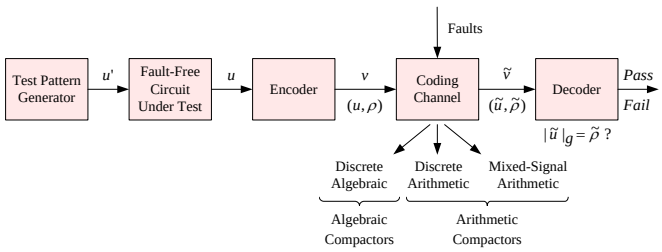

Fig. 1. Error-control coding model as applied to testing

where $\breve{\rho}(b) = |\tilde{u}(b)|_{g(b)}$.

Similarly, we will refer to r-tuples composed of the coefficients of $\breve{\rho}(b)$, $\rho(b)$ and $\tilde{\rho}(b)$ as *arithmetic* actual, reference and distorted reference signatures:

$$\breve{\rho} = (\breve{\rho}_{r-1}, \ldots, \breve{\rho}_0);\ \rho = (\rho_{r-1}, \ldots, \rho_0);\ \tilde{\rho} = (\tilde{\rho}_{r-1}, \ldots, \tilde{\rho}_0)$$

And, in terms of signatures, equation (17) is

$$\breve{\rho} = \begin{cases} \tilde{\rho} & \text{for internal tester} \\ \rho & \text{for external tester} \end{cases}$$

The encoding/decoding procedures for all of the codes examined above involve polynomial division. We only focus on two representatives described by equations (15) and (17). Polynomial division occurs both in the left and right parts of these equations. We employ these codes for testing as shown in Figure 1. The figure interprets the communication model for testing (by residue codes). The test pattern generator (TPG) applies k input test stimuli, $u' = (u'_{k-1}, \ldots, u'_0)$, to the fault-free circuit under test (CUT). The corresponding k output responses (m-bit each), $u = (u_{k-1}, \ldots, u_0)$, form the message that is encoded into the codeword, $v = (u, |u|_g)$. While the message, u, comes from the CUT, the residue, $\rho = |u|_g$, arrives from the tester's storage and may also be perturbed. The codeword, $v = (u, \rho)$, enters the symbolic coding channel, where it is distorted by faults and turns into the received message, $\tilde{v} = (\tilde{u}, \tilde{\rho})$. The decoder generates the residue of the received message, $\breve{\rho} = |\tilde{u}|_g$, and compares it with the received residue, $\tilde{\rho}$, verifying the validity of (17) (respectively, (15)) and making a *pass/fail* decision.

For example, let the CUT be a finite state machine (FSM), with a single input and output (see Figure 2). Any faults are possible in the FSM (stuck-at faults, bridging faults, intermittent faults, etc.). The sequence of input test stimuli that detects all of these (single and multiple) faults is pseudo-random and its length is 8 bits, $u' = 10110010$. Let the test response of a fault free FSM be $u = 11010100$. The channel is discrete algebraic, so the encoder is a standard single input signature analyzer, e.g., the 3-bit one, with the polynomial $g(x) = x^3 + x + 1$ (and the probability of undetected error $1/2^3$). The encoder encodes the sequence $u = 11010100$, i.e., generates the fault-free circuit's signature, $\sigma(x) = |x^7 + x^6 + x^4 + x^2|_{g(x)} = x^2 + x$ (or $\sigma = 110$), and appends it to the fault free response, $v = 11010100, 110$. Let the faults occurring in the FSM modify the response to $\tilde{u} = 10101111$ and the reference signature in the tester's memory is not corrupted, i.e. $\tilde{v} = 10101111, 110$. The faulty circuit's signature (generated by the decoder) then becomes $\breve{\sigma}(x) = |\tilde{u}|_{g(x)} = |x^7 + x^5 + x^3 + x^2 + x + 1|_{g(x)} = x$ (or $\breve{\sigma} = 010$). Since $\breve{\sigma} \neq \sigma$, the decision is *Fail*.

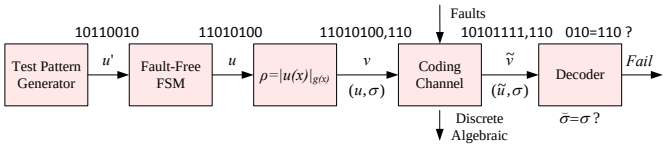

Fig. 2. Error-control coding model: an algebraic case example

Depending on the nature of the CUT, we consider three different types of channels:

- a *discrete algebraic* channel refers to a CUT being an FSM (including storage or combinational logic)
- a *discrete arithmetic* channel refers to a CUT being an arithmetic data processing device
- a *mixed-signal arithmetic* channel refers to a CUT being a mixed-signal system with discrete output: analog-to-digital converter, measurement system,etc.

The channel type specifies the encoder/decoder structure. In the first case, this is an algebraic device. In the last two cases, the encoder/decoder is arithmetic.

The principal difference of the mixed-signal channel from the two others is that its input test stimuli are analogue and the output responses are discrete finite *intervals*. In contrast to a point (exact) value, an interval contains a range of values and is defined by left and right endpoints [25].

It is important to note that the concept of polynomial division is only applicable to *off-line* testing.

3 Remainder Computing Circuits

In this section, we consider the first of the three channels represented in Figure 1, a discrete algebraic channel.

3.1 Discrete Algebraic Channel

A remainder computing circuit (RmCC) is used as a part of the encoder/decoder for a polynomial residue code. A remainder is computed by dividing the message polynomial, $u(x)$, or the received message polynomial, $\tilde{u}(x)$, by the generator polynomial $g(x)$, as shown in (14). Since the two division operations are identical, we only consider the encoding process, i.e. computation of $|u(x)|_{g(x)}$, where

$$u(x)=u_{k-1}x^{k-1}+\ldots+u_0; \qquad g(x)=g_r x^r+\ldots+g_0$$

Let us factor x in the polynomial $u(x)$:

$$\begin{aligned}|u(x)|_{g(x)} &= |u_{k-1}x^{k-1}+\ldots+u_0|_{g(x)}\\ &= |\overbrace{x\cdots(x}^{k-r}(u_{k-1}x^{r-1}+\ldots+u_{k-r})+\ldots+u_0|_{g(x)}\\ &= |x\cdots|x[p(x)]+u_{k-r-1}|_{g(x)}+\ldots+u_0|_{g(x)}\end{aligned} \tag{18}$$

Here, we introduced a degree $r-1$ *partial remainder*, $p(x)$:

$$p(x)=u_{k-1}x^{r-1}+u_{k-2}x^{r-2}+\ldots+u_{k-r}$$

According to (18), computation of $|u(x)|_{g(x)}$ consists of repetitive operations of the form $|x[p(x)]+u_{k-r-1}|_{g(x)}$. Set

$$p^+(x)=|x[p(x)]+u_{k-r-1}|_{g(x)} \tag{19}$$

then

$$|u(x)|_{g(x)}=|x\cdots p^+(x)+\ldots+u_0|_{g(x)} \tag{20}$$

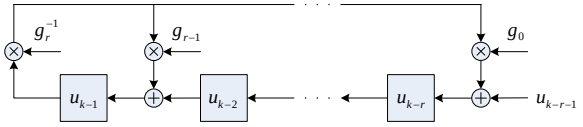

Fig. 3. Polynomial division in algebraic domain

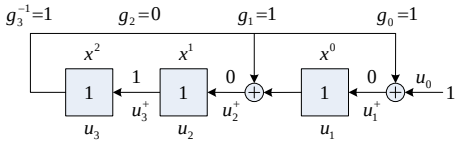

Fig. 4. A length-3 binary SA

and (20) can be implemented using a finite state machine. Indeed, $p(x)$ serves as the present state, u_{k-r-1} is the input and $p^+(x)$ is the next state. In these notations, each shift of the register that holds $p(x)$ is equivalent to multiplication of its content by $x \mod g(x)$ with further addition of u_{k-r-1}.

Equation (19) can be computed as follows:

$$\begin{array}{rl} & u_{k-1}x^r+\ldots+ \quad u_{k-r-1} \\ \oplus & u_{k-1}x^r+\ldots+ \quad u_{k-1}g_r^{-1}g_0 \\ \hline & 0 \quad +\ldots+u_{k-1}g_r^{-1}g_0+u_{k-r-1} \end{array} \quad \Bigg| \begin{array}{c} \underline{g_r x^r+\ldots+g_0} \\ u_{k-1}g_r^{-1} \end{array} \tag{21}$$

The circuit that implements this computation is shown in Figure 3. In testing applications, this circuit is called an *r-stage multiple-input signature analyzer* (SA). The circuit structure is specified by the degree r polynomial $g(x)$ over the field $GF(2^m)$. Here m is the system bus width. All of the buses in Figure 3 (and the remaining figures) are depicted as single lines, in compliance with the style adopted in error-control coding; this simplifies perception of pictures. Also, the circuits designed in this work are built of adders, XORs and flip-flops. These restrictions insure that the overall system is linear (note that we are considering *linear* codes).

Error-control capabilities of the codes defined by primitive polynomials are better and their implementation is simpler, therefore we will mostly select primitive polynomials for our applications. To further save hardware (without deteriorating error-control capabilities), the coefficient of the highest power of $g(x)$ will be 1:

$$g(x)=x^r+g_{r-1}x^{r-1}+\ldots+g_0 \tag{22}$$

Example 3.1 (a 3-stage 1-input SA)**.** Let us consider the following polynomials ($k=4, r=3$):

$$u(x)=x^3+x^2+x+1; \qquad g(x)=x^3+x+1$$

The primitive generator polynomial, $g(x)$, was taken from the table of irreducible polynomials [1].

For our example, the partial remainder:

$$p(x)=x^2+x+1$$

After the shift, the analyzer's content is (see Figure 4):

$$p^+(x)=|x(x^2+x+1)+1|_{g(x)}=x^2$$

Example 3.2 (a 1-stage 1-input SA)**.** Set $k=2$, $r=1$ and:

$$u(x)=x; \qquad g(x)=x+1$$

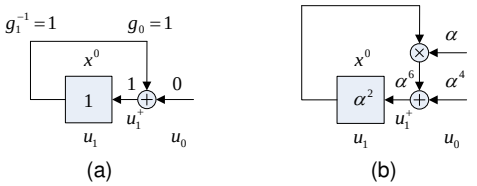

Fig. 5. A 1-stage binary (a) and octal (b) signature analyzers

TABLE 1
Three representations for elements of the field $GF(2^3)$

Power	Polynomial	Vector
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	$\alpha^2+\alpha^1$	1 1 0
α^5	$\alpha^2+\alpha^1+\alpha^0$	1 1 1
α^6	$\alpha^2+\alpha^0$	1 0 1

Then $p(x) = 1$, and after the shift, the content of the analyzer is $p^+(x) = |x \times 1 + 0|_{g(x)} = 1$.

The circuit is presented in Figure 5a and computes the modulo 2 sum (or *parity*) of the incoming bit sequence.

Example 3.3 (a 1-stage 3-input SA)**.** In a non-binary case, it is convenient to represent polynomial coefficients as powers of a primitive element of a field. For a 3-input analyzer, the field, $GF(2^3)$, can be constructed using a root α of the primitive polynomial $w(x) = x^3 + x + 1$. The relationship between different representations of the field elements is given in Table 1.

To determine a primitive irreducible generator polynomial of degree r over $GF(2^m)$ we use procedure from [26]:

1) Select a primitive polynomial of degree mr over $GF(2)$ from the table of irreducible polynomials
2) Let α denote a root of the chosen polynomial and set the correspondence rule between $GF(2^m)$ and $GF(2^{mr})$, denoted by powers of β and α:

$$\beta = \alpha^c; \quad c = [(2^m)^r - 1]/[2^m - 1]; \quad \beta^j = (\alpha^c)^j$$

3) The primitive irreducible polynomial of degree r over $GF(2^m)$ is an expansion of the following equation with (α^i)'s expressed in terms of (β^j)'s:

$$g(x) \;=\; (x - \alpha^{(2^m)^0}) \ldots (x - \alpha^{(2^m)^{r-1}}) \quad (23)$$

Proposition 3.1.1. *The degree 1 polynomial over $GF(2^m)$, $g(x) = x + \alpha$, α being a root of a binary degree m primitive polynomial, is primitive.*

Proof: The proof becomes evident if $r = 1$ in (23). □

In this example, we consider a one-stage signature analyzer. The primitive (generator) polynomial of degree $r = 1$ over $GF(2^3)$ is $g(x) = x + \alpha$, α being a root of a primitive polynomial of degree $mr = 3$ over $GF(2)$, namely $w(x) = x^3 + x + 1$ (see Table 1).

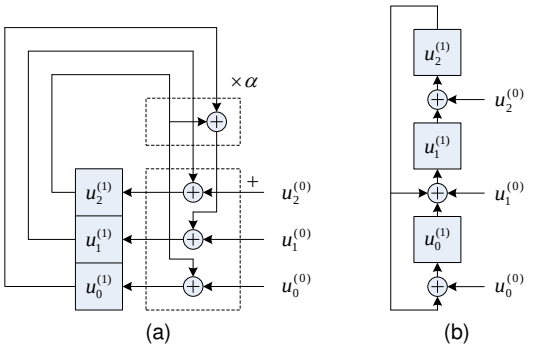

Fig. 6. Vector form of a 3-input SA

Similarly to the previous example, set $k = 2$, $r = 1$, and

$$u(x) = \alpha^2 x + \alpha^4; \qquad g(x) = x + \alpha$$

Then $p(x) = \alpha^2$, and after the shift, the content of the analyzer is $p^+(x) = |x \times \alpha^2 + \alpha^4|_{g(x)} = \alpha^6$. The circuit that implements this operation is presented in Figure 5b. In a vector form, this is equivalent to shifting in two vectors $u_1 = u_2^{(1)}u_1^{(1)}u_0^{(1)} = 100$ and $u_0 = u_2^{(0)}u_1^{(0)}u_0^{(0)} = 110$ The final result (signature) is $\sigma = 101$.

If we switch from the power representation of the field elements to their vector representation, the circuit of Figure 5b will turn into the circuit of Figure 6a. We can continue to shift in additional incoming vectors; each shift is described by the same equation, (19).

The α-multiplier was designed according to the following rule (for simplicity, we will omit the superscript *(1)* of coefficients u). If the current value of a field element is $u(\alpha) = u_2\alpha^2 + u_1\alpha + u_0$, then

$$\alpha(u_2\alpha^2 + u_1\alpha + u_0) = u_1\alpha^2 + (u_2 + u_0)\alpha + u_2$$

The circuit of Figure 6a can be redrawn into the more common form shown in Figure 6b.

One-stage non-binary RmCCs (like the one represented in Figure 6b) are referred to as *spacial (parallel, multiple input)* signature analyzers, while multi-stage binary RmCCs (like the one represented in Figure 4) are referred to as *temporal (serial, single input)* signature analyzers.

The primitive polynomial $g(x) = x + \alpha$ over $GF(2^3)$ was constructed using a root α of the binary polynomial $w(x) = x^3 + x + 1$. It can be shown that if α is a root of $w(x)$, than $\alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$ are also roots of $w(x)$ [1]. Therefore, each of the following polynomials can be used to design a 3-bit SA with identical error-control properties:

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

In Figure 3, the right most stage of the analyzer contains a multiplier by g_0. If $g_0 = \alpha$, the α-multiplier has a form of Figures 5b and 6a. Similarly, if g_0 equals to one of the: $\alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$, we will obtain the following multipliers:

$$\alpha^2 u(\alpha) = (u_2 + u_0)\alpha^2 + (u_2 + u_1)\alpha + u_1$$
$$\alpha^3 u(\alpha) = (u_2 + u_1)\alpha^2 + (u_2 + u_1 + u_0)\alpha + (u_2 + u_0)$$
$$\alpha^4 u(\alpha) = (u_2 + u_1 + u_0)\alpha^2 + (u_1 + u_0)\alpha + (u_2 + u_1)$$
$$\alpha^5 u(\alpha) = (u_1 + u_0)\alpha^2 + u_0\alpha + (u_2 + u_1 + u_0)$$
$$\alpha^6 u(\alpha) = u_0\alpha^2 + u_2\alpha + (u_1 + u_0)$$

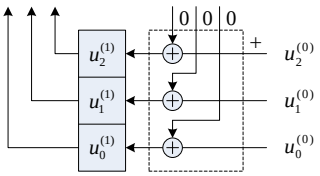

Fig. 7. 1-stage 3-input SA with 0-multiplier

Based on these equations, we conclude that the α-multiplier and α^6-multiplier have the lowest hardware complexity. They only differ by the order of components.

The remaining degree $r = 1$ polynomial over $GF(2^3)$, $g(x) = x + 1$ is not primitive. Nevertheless, it can be used for testing. This analyzer coincides with the one in Figure 6a, where α-multiplier constitutes three straight lines. Clearly, this circuit is a collection of three analyzers of Figure 5a and can similarly be regarded as computing the parity of the sequence of *octal* symbols. The circuit has low hardware complexity but is unsusceptible to even number of errors.

Formally speaking, we have to also consider the generator polynomial with $g_0 = 0$, that is $g(x) = x$. The circuit does not have a feedback and only stores the last incoming symbol. The circuit is depicted in Figure 7.

Using the same technique, we can design a signature analyzer of any length and width. For example, any of the following degree 1 primitive polynomials over $GF(2^6)$ can be used to construct a 1-stage 6-input analyzer (here $w(x) = x^6 + x^5 + 1, \alpha \in GF(2^m), m = 6, r = 1$):

$$x+\alpha,\quad x+\alpha^2,\ \ x+\alpha^4,\ \ x+\alpha^8,\ \ x+\alpha^{16},\ x+\alpha^{31}$$
$$x+\alpha^{32},\ x+\alpha^{47},\ x+\alpha^{55},\ x+\alpha^{59},\ x+\alpha^{61},\ x+\alpha^{62}$$

The lowest hardware complexity is normally associated with the first polynomial that is available for $GF(2^m)$ due to Proposition 3.1.1.

Example 3.4. If the preliminary cleared analyzer of Figure 6 receives the responses $\alpha^5, \alpha^6, \alpha^4, \alpha^2, \alpha^1, \alpha^0$ from a CUT, then after the 6-th shift its content becomes:

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

The power representation of the field element α corresponds to the vector representation 010 (see Table 1) which is the CUT signature.

3.1.1 Aliasing Rate

In an optimal error-detecting (n, k) code, the number of parity check digits, $r = n - k$, always matches the type and character of the error pattern. Therefore, decoding error never occurs. For example, if the message sequence contains $k = 1024$ bits and only single errors are possible in the communication channel, the $(1025, 1024)$ code with a single parity bit, $r = 1$, will detect any of these errors. However, if this code is used for testing, the number of possible errors rises to $k = 1024$. If the number of parity check digits is not increased proportionally, the code ceases to remain optimal and the decoding error may occur (i.e. some errors are not detected). In situations like this, it is important to know the *probability of undetected error*, P_{nd} (*aliasing* or *error escape* rate). Aliasing occurs when the reference signature coincides with a faulty circuit's signature.

In order to facilitate computation of the aliasing rate, we will assume from now on that all errors in the CUT output response, including single, multiple, burst errors, etc., and caused by stuck-at, bridging, intermittent, etc. faults inside the CUT, are equally likely. The exploration of other error models implies modification of the encoder/decoder and is out of the scope of this work.

Proposition 3.1.2. *The aliasing rate for the signature analyzer presented in Figure 3 is estimated as*

$$P_{nd} \approx 2^{-mr} \tag{24}$$

provided that the sequence of k 2^m-ary symbols to be compacted into an r-symbol signature is sufficiently long and all error patters in the sequence are equally likely (independent).

Proof: The aliasing rate is defined as the ratio of the number of errors that are not detected by the chosen code, to the total number of possible errors in the sequence of k 2^m-ary symbols (output responses). The total number of errors in the sequence is $2^{mk} - 1$ (one sequence of symbols is error free). The number of erroneous sequences that will produce the signature that coincides with the reference signature (or the number of errors that are not detected by the code with r 2^m-ary parity check digits) is $2^{mk}/2^{mr} - 1 = 2^{m(k-r)} - 1$. Thus, the aliasing rate

$$P_{nd} = [2^{m(k-r)} - 1]/[2^{mk} - 1]$$

In testing applications $k \geq r$, therefore

$$P_{nd} \approx 2^{m(k-r)}/2^{mk} = 2^{-mr}$$

□

The aliasing rate decreases with the growth of m and r. Normally, in testing applications $r = 1$, therefore

$$P_{nd} \approx 2^{-m} \tag{25}$$

If the width of the analyzer (which matches the width m of the system bus), is not sufficient to achieve the required aliasing rate, it can be increased to $m + i$. The output responses are then interpreted as elements of the extension field, $GF(2^{m+i})$. Usually, this is not required, because the system bus of contemporary systems contains 16 or more lines and the aliasing rate is at most $1/2^{16} = 0.0000153$.

Equation (25) suggests that for the given m, the aliasing rate does not depend on the structure of the generator, $g(x)$. Thus, any analyzer from the previous section (e.g. presented in Figure 6 or 7) will ensure the same aliasing rate, 2^{-m}.

Proposition 3.1.2 refers to the case when the reference signature is uncorrupted. If the test system is built-in (embedded), there is a chance that the signature is corrupted.

Proposition 3.1.3. *The aliasing rate, $\tilde{P}_{nd}$, for the* ***embedded*** *analyzer of Figure 3 is estimated as*

$$\tilde{P}_{nd} = 2^{-mr}$$

provided that all error patterns in the sequence of k 2^m-ary digits to be compacted into an r-digit signature are equally likely.

Proof: The aliasing rate is the ratio of the number of errors that are not detected by the code, to the total number

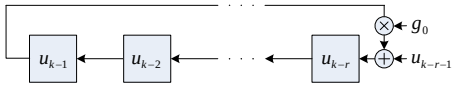

Fig. 8. Low cost algebraic polynomial division circuit

of errors in the sequence of symbols (output responses). Error-detection is performed by verifying equation (13). An error is being detected if this equation does not hold. If equation (13) holds, then either there are no errors at all, or an undetectable error pattern is present.

The total number of errors in the sequence is $2^{mk}-1$. The number of errors that produce the signature that matches $\tilde{\sigma}(x)$ is $2^{mk}/2^{mr}$. The number of possible $\tilde{\sigma}(x)$ patterns is 2^{mr}. Therefore, the number of undetectable errors is $[(2^{mk}/2^{mr}) \cdot 2^{mr}] - 1$ (one of the $\tilde{\sigma}(x)$ patterns is, actually, the undistorted reference signature).

Equation (13) is verified for each $\tilde{\sigma}(x)$ pattern, so the total number of possible errors will increase to $(2^{mk} - 1) \cdot 2^{mr}$. Thus, the aliasing rate

$$\tilde{P}_{nd} = \frac{[(2^{mk}/2^{mr}) \cdot 2^{mr}] - 1}{(2^{mk} - 1) \cdot 2^{mr}} = 2^{-mr} \tag{26}$$

□

Comparing equations (24) and (26), we conclude that for embedded analyzers the equality holds even for small values of k.

3.1.2 Low Cost Circuits

Given the aliasing rate, P_{nd}, the analyzer must have the lowest complexity. This objective can be achieved by analysis of the circuit in Figure 3 and expression (21). The implementation complexity depends on the number of nonzero coefficients of the generator, $g(x)$. This number should be kept low, while preserving the polynomial degree (not to deteriorate the aliasing), which results in the following:

$$g(x) = x^r + g_0 \tag{27}$$

The corresponding *low cost* circuit is shown in Figure 8. If all errors are equally likely, then the coefficient g_0 can be chosen any element of $GF(2^m)$. If $g_0 = 0$, then

$$\begin{aligned} &|u_{k-1}x^{k-1} + \ldots + u_r x^r + u_{r-1}x^{r-1} + \ldots + u_0|_{x^r} \\ &= u_{r-1}x^{r-1} + \ldots + u_0 \end{aligned} \tag{28}$$

This circuit is an r-stage shift register with no feedbacks. The remainder (28) only depends on the last r symbols of the sequence of k 2^m-ary symbols. Since $r \ll k$, only a small portion of all possible errors in the sequence is detected. Therefore, this analyzer is normally not used in practice.

Clearly, all 1-stage ($r = 1$) signature analyzers are low cost circuits. Some of these analyzers were presented in examples 3.2 and 3.3 (figures 5a and 5b, respectively).

Example 3.5 (a 2-stage 3-input low cost SA)**.** Let $r = 2$ and the generator polynomial over $GF(2^3)$ is $g(x) = x^2 + g_0$. Let $g_0 = \alpha$, where α is a root of a primitive binary polynomial, $w(x) = x^3 + x + 1$ (see Table 1). Two equivalent circuits that implement this analyzer are presented in Figure 9.

Note that the polynomial $w(x)$ could also be chosen low cost, for example, $w(x) = x^3 + 1$. The middle feedback in Figure 9b would then disappear.

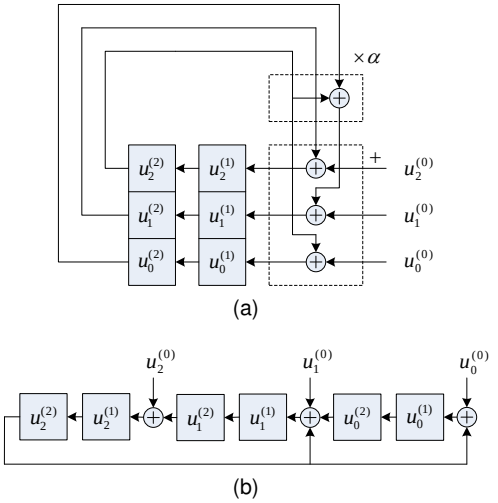

Fig. 9. A 2-stage 3-input low cost SA

4 Residue Computing Circuits

In the previous section, we discussed a discrete algebraic channel. In this section, we consider two other channel types presented in Figure 1, namely a discrete arithmetic channel and a mixed-signal arithmetic channel.

4.1 Discrete Arithmetic Channel

For the arithmetic channel (see Figure 1), the coefficients of the polynomial, $u(b)$, that represents the output response, are integer numbers. Therefore, all (arithmetic) operations on the polynomial will produce carries. This increases the hardware complexity of the arithmetic residue computing circuit compared to a similar algebraic remainder computing circuit. Although we consider sequential implementation of the decoder for a residue code (implying off-line testing), the results can be used in a combinational design (i.e. for on-line testing). Residue codes can protect a single processing unit or the entire computer [27].

The design methodology for a residue computing circuit (RsCC) was developed in arithmetic error-control coding, but it has mainly been oriented to binary case [1]. A non-binary design techniques have been limited to a special type of modulus [2]. We design a residue computing circuit with an arbitrary modulus and base of the number system.

As in the case of the remainder computing circuit, the residue, $|u(b)|_{g(b)}$, is computed by dividing the polynomials $u(b) = u_{k-1}b^{k-1} + \ldots + u_0$ and $g(b) = g_{r-1}b^{r-1} + \ldots + g_0$.

Factoring b in $u(b)$, we obtain

$$|u(b)|_{g(b)} = |b \cdots |b[p(b)] + u_{k-r-1}|_{g(b)} + \ldots + u_0|_{g(b)} \tag{29}$$

where $p(b)$ is a degree $r-1$ *partial residue* polynomial:

$$p(b) = u_{k-1}b^{r-1} + u_{k-2}b^{r-2} + \ldots + u_{k-r}$$

According to (29), computation of $|u(b)|_{g(b)}$ consists of recursive operations

$$p^+(b) = |b[p(b)] + u_{k-r-1}|_{g(b)} \tag{30}$$

Consequently

$$|u(b)|_{g(b)} = |b \cdots p^+(b) + \ldots + u_0|_{g(b)} \tag{31}$$

Similarly to equation (20), equation (31) can be implemented using a finite state machine, if $p(b)$ serves as the present state, $p^+(b)$ is the next state, and u_{k-t-1} is the input. Each shift of the register that holds $p(b)$ is equivalent to multiplication of its content by $b \bmod g(b)$ with further addition of u_{k-r-1}. Note that multiplication and addition are performed over integers; these operations produce carries.

Manipulating (30), we obtain

$$\begin{aligned} p^+(b) &= |b[p(b) + u_{k-r-1}b^{-1}]|_{g(b)} \\ &= |b(u_{k-1}b^{r-1} + \ldots + u_{k-r}b^0 + u_{k-r-1}b^{-1})|_{g(b)} \end{aligned} \tag{32}$$

Equation (32) states that the radix b fraction, $u_{k-1}u_{k-2}\ldots u_{k-r}.u_{k-r-1}$, is multiplied by b and divided by $g(b)$. The residue is computed by subtraction of a multiple of $g(b)$ from the result of the multiplication (actually, subtraction of a multiple of $g(b)/b$ from the content of the analyzer before the multiplication). The value of the subtrahend is computed by a *comparator* and depends on the value of the fraction $u_{k-1}u_{k-2}\ldots u_{k-r}.u_{k-r-1}$. The comparator implements a truth table and is constructed on combinational logic. We have transformed (30) to (32), because the subtraction decision must be made before the shift. The complexity of the comparator depends on the polynomial, $g(b)$, as well as the base of the number system, b.

Proposition 4.1.1. *The number of times, q, that the polynomial $g(b)$ is subtracted from $bp(b) + u_{k-r-1}$ in order to compute the residue $|b[p(b)] + u_{k-r-1}|_{g(b)}$, is upper bounded by $b-1$:*

$$q \le b - 1$$

Proof: Because $u_{k-1}b^{r-1}+\ldots+u_{k-r}b^0$ is a remainder, it is less than $g(b)$, that is, $u_{k-1}b^{r-1}+\ldots+u_{k-r-1}b^{-1} < g(b)$.

Further, because $p^+(b) = |b(u_{k-1}b^{r-1} + \ldots + u_{k-r}b^0 + u_{k-r-1}b^{-1})|_{g(b)}$ and the expression in parentheses is less than $g(b)$, q can't be equal to b, that is $q \le b-1$. □

Corollary 4.1.1. *The number of times, q, that the binary polynomial $g(2)$ is subtracted from another binary polynomial $2[p(2)] + u_{k-r-1}$ in order to compute the residue $|2[p(2)] + u_{k-r-1}|_{g(2)}$, is upper bounded by 1: $q \le 1$*

Proof: Based on 4.1.1 above: $q \le b - 1 = 2 - 1 = 1$ □

The number q serves as a measure of complexity of the residue computing circuit. Corollary 4.1.1 states that the hardware complexity of a binary comparator is less than that of a non-binary comparator.

The division operation (33) demonstrates how to compute the right part of equation (32).

$$\begin{array}{r} u_{k-1}b^r \quad +\ldots+ u_{k-r-1} \\ +\quad \dot{\mu}_r b^r \quad +\ldots+ \quad \dot{\mu}_0 \\ \hline p^+_{k-1}b^{r-1} + \ldots + \; p^+_{k-r} \end{array} \Bigg| \begin{array}{c} g_{r-1}b^{r-1} + \ldots + g_0 \\ \hline q \end{array} \tag{33}$$

In this equation, subtraction is substituted with addition to b's complement. Here q is the quotient, $\mu = q \times g$ is the multiple of g, and $\dot{\mu}$ is the b's complement of μ:

$$\mu = \mu_r b^r + \ldots + \mu_0; \qquad \dot{\mu} = \dot{\mu}_r b^r + \ldots + \dot{\mu}_0$$

The implementation of (33) is shown in Figure 10. The symbol Σ denotes an *arithmetic* adder, and the adder inputs that equal to 0 are removed. The red arrows indicate the carry propagation path. We call this circuit *arithmetic r-stage multiple-input signature analyzer*. The value g of the generator polynomial $g(b)$ is called the *generator* or *modulus*. In our applications we will prefer prime integers g due to better error-control capabilities and simpler implementation.

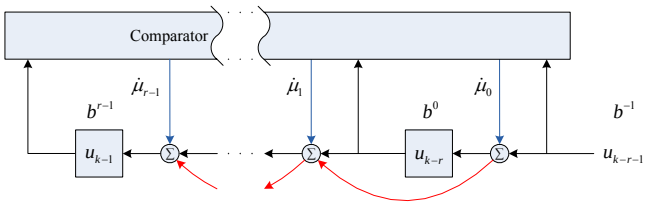

Fig. 10. Polynomial division in arithmetic domain

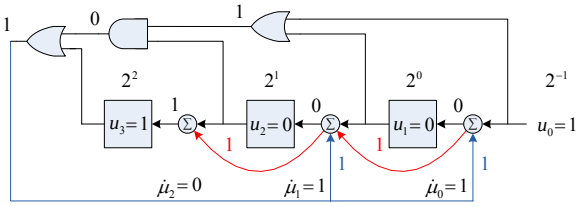

Fig. 11. Arithmetic length-3 binary SA; $g = 5$

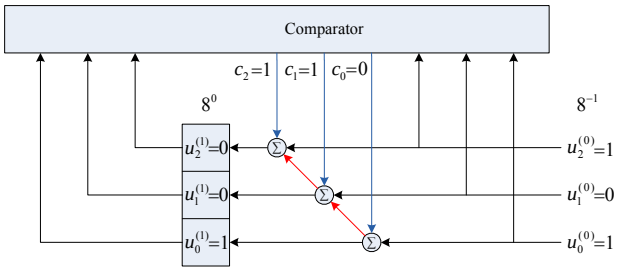

Fig. 12. Arithmetic 1-stage 3-input SA; $g = 5$

Example 4.1 (an arithmetic 3-stage 1-input SA)**.** Let us choose the following polynomials ($k = 4, r = 3$):

$$u(2) = 1 \cdot 2^3 + 1 = 9; \qquad g(2) = 1 \cdot 2^2 + 1 = 5$$

The partial residue polynomial is: $p(2) = 1 \cdot 2^2 = 4$. And after the shift, the content of the analyzer is

$$p^+(2) = |2(1 \cdot 2^2 + 0 \cdot 2 + 0 + 1 \cdot 2^{-1})|_{g(2)}$$

If the analyzer content before the shift is greater than or equal to $g/2 = 5/2 = 2.5_{10}$, the comparator must initiate subtraction of 2.5_{10} from this content. In our case

$$1 \cdot 2^2 + 0 \cdot 2 + 0 + 1 \cdot 2^{-1} \ge g/2 \tag{34}$$

Because (34) holds, the 2's complement of 10.1_2 (i.e. 01.1_2) is added to the content and the result is shifted left:

$$p^+ = 2(1 \cdot 2^2 + (0+0) \cdot 2 + (0+1) + (1+1) \cdot 2^{-1}) = 1|100_2$$

Indeed, $9 \bmod 5 = 4$. The logic expression for the signal c that adds 01.1_2, if (34) holds, is: $c = u_3 + u_2(u_1 + u_0)$.

This analyzer is depicted in Figure 11.

Example 4.2 (a 1-stage 1-input SA)**.** Set $k = 2$, $r = 2$ and $u(2) = 1 \cdot 2 + 1 = 3$, $g(2) = 1 \cdot 2 + 0 = 2$. Then $p(2) = 1$, and after the shift $p^+(2) = |2 \cdot 1 + 1|_{g(2)} = |2(1 + 1 \cdot 2^{-1})|_{g(2)}$.

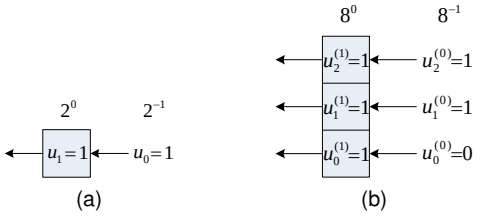

Fig. 13. Arithmetic 1-stage binary (a) and octal (b) signature analyzers

As $1 + 1 \cdot 2^{-1} = 1.1_2 = 1.5_{10}$ is greater than $g(2)/2 = 1.0_2 = 1_{10}$, the 2's complement of 1.0_2 (i.e. 1.0_2), must be added to the register content and the result must be shifted left. The same would have happened, if the register content were $1.0_2 = 1_{10}$. However, there is only 1 flip-flop available in the analyzer (as modulo 2 residue is either 0 or 1). There is no need to add a 1 to the content of this flip-flop, u_1, since the result will be lost anyway. On the other hand, if the data were $0.0_2 = 0_{10}$ or $0.1_2 = 0.5_{10}$, a 0 must have been added to the content. Thus, a 0 should be added to the content in all cases, i.e. the feedback in the analyzer disappears. The value of $u_0 = 1$ will pass to the flip-flop. Indeed, $3 \bmod 2 = 1$.

The circuit is presented in Figure 13a. It computes the parity of the binary integer formed by the incoming bits. Apparently, the parity equals to the value of the last bit.

Example 4.3 (a 1-stage 3-input SA)**.** Let $b = 2^3, k = r = 2$: $u(8) = 7 \cdot 2^3 + 6 = 62$, $g(8) = 1 \cdot 2^3 + 0 = 8$ Then $p(b) = 7$, and after the shift, the analyzer contains $p^+(8) = |8 \cdot 7 + 6|_8 = |8(7 + 6 \cdot 8^{-1})|_8$. As $7 + 6 \cdot 8^{-1} = 7.6_8 = 7.75_{10}$ is greater than $8/8 = 1.0_8 = 1_{10}$, the 8's complement of $q \times 1.0 = 7 \times 1.0 = 7.0_8$ (i.e. 1.0_8), must be added to the register content and the result must be shifted left. The least significant digit of the addend is the only essential digit that is used for addition. If we similarly consider the 8's compliments of all possible cases of the subtrahend (ranging from 0.0_8 to 7.7_8), this essential digit will always be 0. Therefore, the feedback in the analyzer disappears. The value $u_0 = 6$ will be passed to the register. Indeed, $62 \bmod 8 = 6$.

The circuit is presented in Figure 13b. We can consider this circuit as computing the *octal* parity of the octal integer formed by the incoming octal symbols. Clearly, the octal parity equals to the value of the last symbol.

Absence of the feedback (examples 4.2 and 4.3) may lower the error detecting capability. Therefore, the moduli equal to powers of b should be avoided. A better choice would be prime moduli (as mentioned earlier).

The circuit of figure 13b is an arithmetic equivalent of the algebraic circuit presented in Figure 7. This is the consequence of the identical polynomial formats:

$$g(x) = x + 0; \qquad g(b) = b + 0$$

Example 4.4 (an arithmetic 1-stage 3-input SA with prime modulus)**.** Let us consider an analyzer that uses the same modulus, $g = 5$, as the one represented in Figure 11, but is 1-stage. Note that the incoming symbols are *octal* numbers, i.e. they may exceed the modulus 5. When designing the circuit, we therefore distinguish two cases: (a) the content of the 3-bit analyzer is always less than 5, and (b) the content of the 3-bit analyzer may range from 0 to 7; however, if it exceeds 4, it must be interpreted as follows: $5 \to 0, 6 \to 1, 7 \to 2$. Essentially, this means that in case (a), the analyzer transforms all incoming integers into real residues, whereas in case (b) this is done by the observer. The hardware complexity of the circuits (b) might be lower, since they are less restricted.

Set $b = 2^3, k = 2, r = 1, u(b) = 1 \cdot 2^3 + 5 = 13, g(b) = 5$. Then $p(b) = 1$, and after the shift, the analyzer contains $p^+(2^3) = |1 \cdot 8 + 5|_5 = |8(1 + 5 \cdot 8^{-1})|_5$. As $1 + 5 \cdot 8^{-1} = 1.5_8 = 1.625_{10}$ is greater than $5/8 = 0.5_8 = 0.625_{10}$, the 8's complement of $q \times 0.5_8 = 2 \times 0.5_8 = 1.2_8 = 1.25_{10}$ (i.e. 6.6_8), must be added to the register content, 1.5_8, and the result, 0.3_8, must be shifted left. After the shift, the content of the analyzer becomes $3_8 = 3_{10}$. Indeed, $13 \bmod 5 = 3$.

The circuit is presented in Figure 12, where the signals c_2, c_1, c_0 that initiate addition of the 8's complement are defined as follows (with $\bar{u}_j^{(i)}$ being inversion of $u_j^{(i)}$):

$$\begin{aligned}
c_2 &= u_0^{(1)}(u_2^{(0)} \oplus u_1^{(0)}) + u_1^{(1)}\bar{u}_0^{(1)}u_2^{(0)} + u_2^{(1)}u_1^{(0)}u_0^{(0)} \\
&\quad + u_2^{(1)}u_2^{(0)} + \bar{u}_1^{(1)}u_0^{(1)}u_1^{(0)}\bar{u}_0^{(0)} + u_1^{(1)}u_0^{(1)}\bar{u}_1^{(0)} \\
c_1 &= u_2^{(1)}\bar{u}_2^{(0)}(\bar{u}_1^{(0)} + \bar{u}_0^{(0)}) + u_1^{(1)}u_0^{(1)}u_2^{(0)} \\
&\quad + u_0^{(1)}u_1^{(0)}\bar{u}_0^{(0)} + \bar{u}_2^{(1)}\bar{u}_1^{(1)}\bar{u}_0^{(1)}u_2^{(0)}(u_1^{(0)} + u_0^{(0)}) \\
&\quad + \bar{u}_1^{(1)}u_0^{(1)}\bar{u}_1^{(0)} + u_0^{(1)}\bar{u}_2^{(0)}u_0^{(0)} \\
c_0 &= \bar{u}_1^{(1)}\bar{u}_0^{(1)}u_2^{(0)}(u_0^{(0)} + u_1^{(0)}) + \bar{u}_1^{(1)}u_0^{(1)}\bar{u}_2^{(0)}\bar{u}_1^{(0)} \\
&\quad + \bar{u}_1^{(1)}u_2^{(0)}u_1^{(0)}u_0^{(0)} + u_1^{(1)}\bar{u}_2^{(0)}(\bar{u}_0^{(1)} + u_1^{(0)} + u_0^{(0)}) \\
&\quad + u_1^{(1)}\bar{u}_2^{(0)}u_1^{(0)} + u_1^{(1)}u_0^{(1)}u_2^{(0)}\bar{u}_1^{(0)} + u_2^{(1)}u_1^{(0)}u_0^{(0)} \\
&\quad + u_2^{(1)}u_2^{(0)}
\end{aligned}$$

As evident from these examples, the architecture and design procedure for arithmetic circuits are more complex compared to algebraic circuits. In the next section, we consider conditions under which this complexity is reduced.

4.1.1 Aliasing Rate

The aliasing rate for arithmetic codes can be derived in the way similar to algebraic codes.

Proposition 4.1.2. *The aliasing rate for the modulo g signature analyzer represented in Figure 10 is upper bounded by g^{-1}:*

$$P_{nd} \leq g^{-1}$$

provided that the sequence of k 2^m-ary symbols to be compacted into an r-symbol signature is long and all error patterns in the sequence are equally likely.

Proof: The total number of errors in the sequence of k 2^m-ary symbols (output responses) is $2^{mk} - 1$. The compaction modulus is $g = g_{r-1}(2^m)^{r-1} + g_{r-2}(2^m)^{r-2} + \ldots + g_0, g_i < 2^m, i = 0, \ldots, r-1$. The number of errors which are not detected by the code that uses modulus g is $\lfloor 2^{mk}/g \rfloor - 1$, where $\lfloor x \rfloor$ means the largest integer less than or equal to x. Thus, the aliasing rate is

$$P_{nd} = (\lfloor 2^{mk}/g \rfloor - 1)/(2^{mk} - 1)$$

For large k, $P_{nd} \approx \frac{\lfloor 2^{mk}/g \rfloor}{2^{mk}} \leq \frac{2^{mk}/g}{2^{mk}} = g^{-1}$. □

Corollary 4.1.2. *The aliasing rate decreases with the growth of modulus g and reaches its minimum when $g = 2^{mr} - 1$ (with m and r being the analyzers width and length respectively):*

$$min(P_{nd}) = (2^{mr} - 1)^{-1} \tag{35}$$

Proof: The proof becomes evident if take into consideration the fact that the maximum value for g is $2^{mr}-1$. Indeed, if we select the higher modulus (yet available with the m-bit bus), $g = 2^{mr} = b^r$, then, similarly to (28),

$$|u_{k-1}b^{k-1} + \ldots + u_0|_{b^r} = u_{r-1}b^{r-1} + \ldots + u_0 \qquad (36)$$

The residue (36) only depends on the last r symbols of the sequence of k 2^m-ary symbols. Since $r \ll k$, only a small portion of errors in the sequence of k symbols can be detected. The appropriate circuit coincides with the one shown in Figure 8 with no feedback. Because of high error escape rate, this circuit is not used in practice. □

For testing applications $r = 1$ and

$$min(P_{nd}) = (2^m - 1)^{-1} \qquad (37)$$

For larger values of mr, we can ignore subtrahend in (35) and (37). Then, the *lowest* values for the aliasing rate are

$$P_{nd} \approx 2^{-mr}; \qquad P_{nd} \approx 2^{-m}$$

These expressions correlate respectively with expressions (24) and (25) for an algebraic signature analyzer.

In the case of an algebraic analyzer, the degree m of the polynomial defines the aliasing rate. This rate does not depend on the structure of the polynomial (provided that all errors in the stream under compression are equally likely).

In the case of an arithmetic analyzer, the aliasing rate depends on the modulus, g. However, the moduli formed by distinct polynomials of the same degree, differ from each other. Accordingly, the error escape rates will be different. As it follows from Corollary 4.1.2, the aliasing rate reaches its minimum, (37), for the following degree m polynomial: $g = 2^m - 1$. Under certain conditions, this modulus can be further increased (and the minimum (37) further reduced).

Example 4.5. The aliasing rates for the parallel ($r = 1$) algebraic signature analyzers based on the degree 3 ($m = 3$) algebraic polynomials, $g_1(x) = x^3+x+1$ and $g_2(x) = x^3+1$, are the same: $P_{nd} = 2^{-mr} = 2^{-3} = 0.125$. And the aliasing rates for the parallel ($r = 1$) arithmetic signature analyzers based on degree 2 (note that the number of lines m in the bus is the same, $m = 3$) arithmetic polynomials, $g_1^*(2) = 2^2+1 = 5$ and $g_2^*(2) = 2^2+2+1 = 7$, are different: $P_{1nd} = (g_1^*)^{-1} = 0.2$ and $P_{2nd} = (g_2^*)^{-1} = 0.143$. The latter value signifies the lowest aliasing rate for a 3-bit arithmetic analyzer: $min(P_{nd}) = (2^{mr} - 1)^{-1} = (2^{1\cdot 3} - 1)^{-1} = 0.143$, which is higher than that for the equivalent algebraic analyzer, $2^{-3} = 0.125$. Under certain conditions, the aliasing rate for the 3-bit arithmetic analyzer can be further reduced to that of the 3-bit algebraic analyzer, i.e. $(2^{1\cdot 3})^{-1} = 0.125$ (no feedback is removed in the analyzer).

The algebraic analyzers based on the polynomials $g_1(x)$ and $g_2(x)$ are respectively shown in Figures 6 and 14 (note that $|x(u_2x^2 + u_1x + u_0)|_{x^3+1} = u_1x^2 + u_0x + u_2$).

The modulo $g_1^*(2) = 5$ analyzer is presented in Figure 12. The analyzer with $g_2^*(2) = 7$ is shown in the next example.

Likewise to an algebraic analyzer case, we can state the following proposition for an arithmetic analyzer.

Proposition 4.1.3. *The aliasing rate for the modulo g* ***embedded*** *analyzer of Figure 10 is upper bounded by g^{-1}:*

$$\tilde{P}_{nd} \leq g^{-1}$$

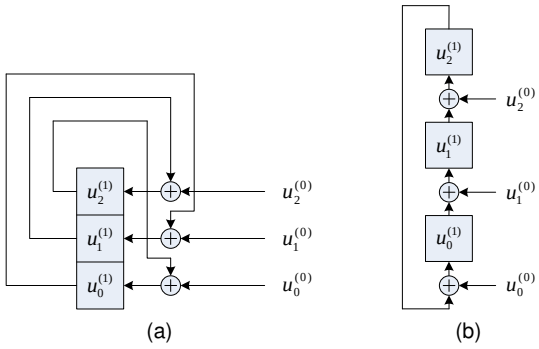

Fig. 14. 3-input SA with $g_2(x) = x^3 + 1$

provided that all error patterns in the sequence of k 2^m-ary digits to be compacted into an r-digit signature are equally likely.

Proof: Error-detection is performed by verifying equation (17) (internal tester part). If it holds, then there are no errors or an undetectable error occurred.

The total number of errors in the sequence is $2^{mk}-1$. The number of errors that produce the signature that matches $\tilde{\rho}(b)$ is $\lfloor 2^{mk}/g \rfloor - 1$. The number of possible $\tilde{\rho}(b)$ patterns is g. Therefore, the total number of undetectable errors is $[\lfloor 2^{mk}/g \rfloor \cdot g] - 1 \leq [(2^{mk}/g) \cdot g] - 1 = 2^{mk} - 1$.

Equation (17) is verified for each $\tilde{\rho}(b)$ pattern, so the total number of possible errors will increase to $(2^{mk} - 1) \cdot g$.

Thus, $\tilde{P}_{nd} = \frac{[\lfloor 2^{mk}/g \rfloor \cdot g]-1}{(2^{mk}-1)\cdot g} \leq \frac{2^{mk}-1}{(2^{mk}-1)\cdot g} = g^{-1}$ □

4.1.2 Choice of a Compaction Modulus

Let us synthesize an m-input analyzer of Figure 10 that would have the lowest aliasing rate.

According to Corollary 4.1.2, the aliasing rate reaches its minimum if modulus $g = 2^{mr} - 1$. The same modulus ensures low complexity. Indeed, the complexity depends on the number of nonzero coefficients in base b representation of the modulus, which is the smallest for our choice:

$$g(b) = g_{r-1}b^{r-1} + \ldots + g_0 = b^r - 1$$

Using equation (30) and taking into consideration that $|b^r|_{b^r-1} = 1$, we derive

$$\begin{aligned} p^+(b) &= |b(u_{k-1}b^{r-1} + \ldots + u_{k-r}) + u_{k-r-1}|_{g(b)} \\ &= |u_{k-2}b^{r-1} + \ldots + (u_{k-1} + u_{k-r-1})|_{b^r-1} \end{aligned}$$

The appropriate circuit is shown in Figure 15. It is a *low cost* residue computing circuit. The red arrows "compensate" for the possible overflow (they implement the part shown in red in the above equation). Indeed, the *carry out* bit that may appear from the last adder represents the $c_{out}b^r$ term, which, if not fed anywhere, will introduce an error. In order to rectify this error, the *truncated* part of the term ($|c_{out}b^r|_{b^r-1} = c_{out}$) is fed back into the circuit:

$$\begin{aligned} p^+(b) &= |c_{out}b^r + u_{k-2}b^{r-1} + \ldots + u_{k-r-1}|_{b^r-1} \\ &= |u_{k-2}b^{r-1} + \ldots + (u_{k-1} + u_{k-r-1} + c_{out})|_{b^r-1} \end{aligned}$$

For the circuits of Figure 15, where $g(b) = b^r - 1$, the red feedback is always a single-bit signal, whereas the black feedback may be a single-bit signal or a multiple-bit signal (depending on the size m of the system bus).

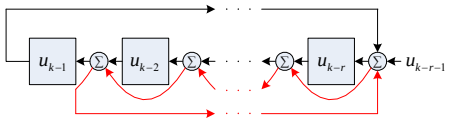

Fig. 15. Low cost residue computing circuit

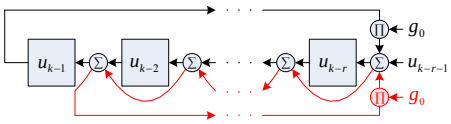

Fig. 16. Generic low cost residue computing circuit

In general, the circuit that uses the following modulus can be regarded as a low cost circuit:

$$g(b) = b^r - g_0 \tag{38}$$

where $0 \leq g_0 < b$. This equation correlates with equation (27) that we used for an algebraic analyzer. In order to save hardware, the polynomial (38) is chosen monic.

Analogously, $|c_{out}b^r|_{b^r-g_0} = c_{out}g_0$, where c_{out} is a single-bit signal. Respectively,

$$\begin{aligned} p^+(b) &= |b(u_{k-1}b^{r-1} + \ldots + u_{k-r}) + u_{k-r-1}|_{g(b)} \\ &= |c_{out}b^r + u_{k-1}b^r + u_{k-2}b^{r-1} + \ldots + u_{k-r-1}|_{b^r-g_0} \\ &= |u_{k-2}b^{r-1} + \ldots + c_{out}g_0 + u_{k-1}g_0 + u_{k-r-1}|_{b^r-g_0} \end{aligned}$$

This analyzer is shown in Figure 16. Note the similarity with the algebraic low cost analyzer presented in Figure 8. The g_0 multiplier is denoted by $\textcircled{\Pi}$ to reflect its difference from the algebraic multiplier. The feedback colored in red ensures the correctness of the modular operation.

The red feedback in Figure 16 (including the g_0 multiplier) is asynchronous by nature, and if $g_0 > 1$, a race condition may occur. To make the circuit operation more stable, we can feed the most significant carry out signal (along with the g_0 multiplier) forward into the input of the subsequent registered digits, e.g. u_{k-1}. This may, however, increase the hardware complexity of the design.

Equation (38) can be extended to modulus of the form:

$$g(b) = b^r - g_{r-1}b^{r-1} - \ldots - g_0$$

Note the similarity with the respective algebraic generator polynomial (22).

By the same reasoning, taking into account that $|b^r|_{g_b} = g_{r-1}b^{r-1} + \ldots + g_0$, we can write:

$$\begin{aligned} p^+(b) &= |b(u_{k-1}b^{r-1} + \ldots + u_{k-r}) + u_{k-r-1}|_{g(b)} \\ &= |c_{out}b^r + u_{k-1}b^r + \ldots + u_{k-r-1}|_{g(b)} \\ &= |(c_{out}g_{r-1} + u_{k-1}g_{r-1} + u_{k-2})b^{r-1} + \ldots \\ &+ (c_{out}g_0 + u_{k-1}g_0 + u_{k-r-1})b^0|_{g(b)} \end{aligned}$$

The implementation of this expression is shown in Figure 17. This is not a low cost circuit anymore. Note the similarity with the circuit of Figure 3. The chosen (generic) form of modulus increases the number of feedbacks and complicates the structure of the analyzer (bringing it closer to the complexity of the one in Figure 10). As well, the feedbacks that propagate carries may increase the operational

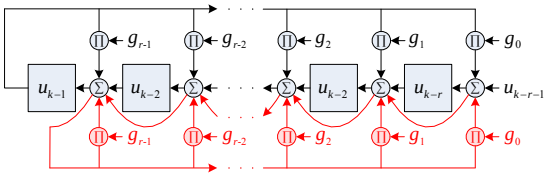

Fig. 17. Generic residue computing circuit

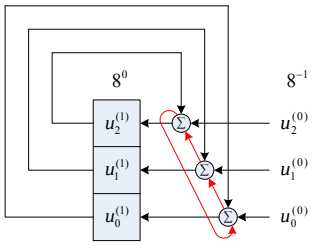

Fig. 18. Low cost $\bmod 7$ arithmetic 1-stage 3-input signature analyzer

delay. The optimal choice between this structure and the one of Figure 10 depends on the values of g, r and m.

Example 4.6 (a 1-stage 3-input low cost $\bmod 7$ SA)**.** Here $r = 1$ and $g(b) = 2^{3\cdot1} - 1 = 7$. The circuit implementing the analyzer is shown in Figure 18. It constitutes an arithmetic counterpart of the one in Figure 6a with $\times\alpha^0$ multiplier.

Example 4.7 (a 1-stage 3-input low cost $\bmod 5$ SA)**.** Here $r = 1, g = 2^{m\cdot r} - g_0 = 2^{3\cdot1} - 3 = 5$, and

$$\begin{aligned} p^+ &= |2^3u_1 + u_0|_5 = |(2+1)u_1 + u_0|_5 \\ &= |c_{out}2^3 + (u_2^{(1)} + u_1^{(1)})2^2 + (u_2^{(1)} + u_1^{(1)} + u_0^{(1)})2 \\ &+ (u_2^{(1)} + u_0^{(1)}) + u_2^{(0)}2^2 + u_1^{(0)}2 + u_0^{(0)}|_5 \\ &= |(u_2^{(1)} + u_1^{(1)} + u_2^{(0)})2^2 + (u_2^{(1)} + u_1^{(1)} + u_0^{(1)} \\ &+ u_1^{(0)} + c_{out})2 + (u_2^{(1)} + u_0^{(1)} + u_0^{(0)} + c_{out})|_5 \end{aligned}$$

One implementation of this expression is shown in Figure 19. This circuit is redrawn in Figure 20 in the way that is more consistent with the scheme of Figure 16. The upper $\times 3$ multiplier can also be implemented as $\times 3 \bmod 5$ multiplier. Both multipliers can be designed as arithmetic or logical circuits. In the latter case, the expressions for the upper $\times 3 \bmod 5$ multiplier are:

$$\begin{aligned} o_2 &= \bar{u}_2^{(1)}u_1^{(1)}u_0^{(1)} \\ o_1 &= \bar{u}_2^{(1)}\bar{u}_1^{(1)}u_0^{(1)} + u_2^{(1)}\bar{u}_0^{(1)} \\ o_0 &= \bar{u}_2^{(1)}\bar{u}_1^{(1)}u_0^{(1)} + u_1^{(1)}\bar{u}_0^{(1)} + u_2^{(1)}u_1^{(1)} \end{aligned}$$

This greatly improves the hardware complexity (see Figure 21). And the red feedback (actually, feed-forward) signal becomes a single-line signal. Essentially, the circuit performs the following operation:

$$p^+ = |8c_{out} + 8u_1 + u_0|_5 = |3c_{out} + |3u_1|_5 + u_0|_5$$

The use of the feed-forward signal decreases complexity of the circuit and improves its stability.

The circuits of Figures 19 - 21 constitute alternative implementations of the $\bmod 5$ computing circuit presented in Figure 12. These circuits are more structured and thus easier to design and program by a hardware description

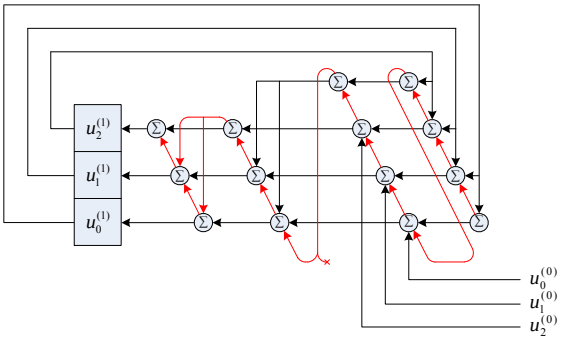

Fig. 19. Low cost $\mathrm{mod}\ 5$ arithmetic 1-stage 3-input signature analyzer

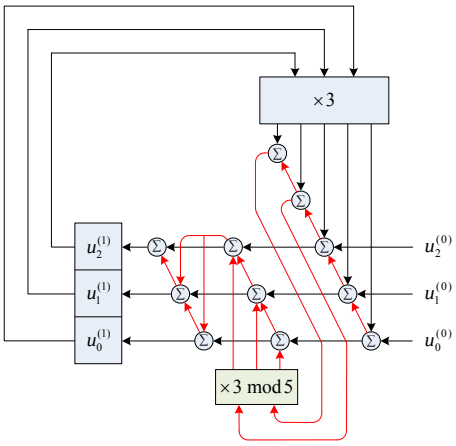

Fig. 20. Alternative form of the circuit of Figure 19

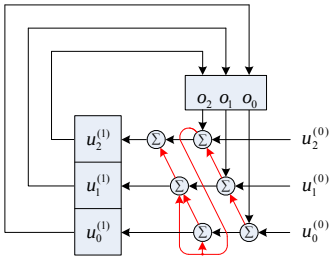

Fig. 21. Combinational implementation of the upper $\times 3\ \mathrm{mod}\ 5$ multiplier in Figure 20

language. However, their operational cycle is longer (due to carry propagation delays) and they may require more gates.

Figure 22 demonstrates simulation results for all of the alternative choices. Here, $u0\_pres$ signifies the present input, $u^{(0)} = (u_2^{(0)}2^2 + u_1^{(0)}2 + u_0^{(0)})$; $u1\_pres$ signifies the present state of the register, $u^{(1)} = (u_2^{(1)}2^2 + u_1^{(1)}2 + u_0^{(1)})$; and $u1\_next$ signifies its next state. The circuit performs the following operation: $u1\_next = |u1\_pres \times 8 + u0\_pres|_5$.

Example 4.8 (a 2-stage 3-input low cost $\mathrm{mod}\ 63$ SA)**.** Here $r = 2$ and $g(b) = 2^{3\cdot 2} - 1 = 63$. The circuit is presented in Figure 23. The feedback colored in red insures the correctness of $\mathrm{mod}\ 63$ operation.

Example 4.9 (a 2-stage 3-input low cost $\mathrm{mod}\ 59$ SA)**.** Here $r = 2$ and $g(b) = 2^{3\cdot 2} - 5 = 59$. The modular operation

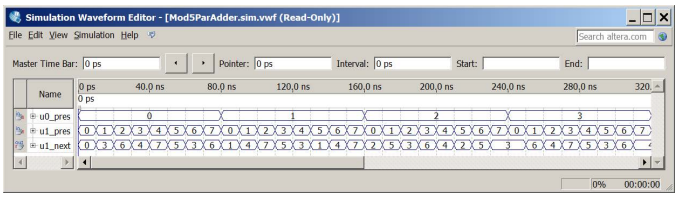

Fig. 22. Simulation of the combinational part of the $\mathrm{mod}\ 5$ residue generator performing the operation $u1\_next = |u1\_pres \times 8 + u0\_pres|_5$

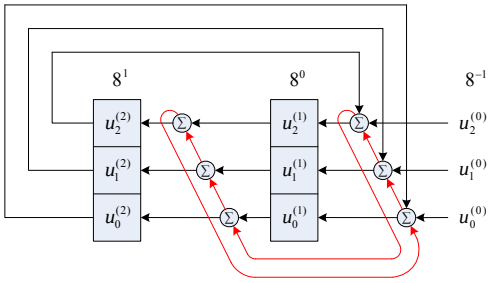

Fig. 23. Low cost $\mathrm{mod}\ 63$ arithmetic 2-stage 3-input signature analyzer

performed by the circuit can be written as:

$$p^+ = |8(8u_2 + u_1) + u_0|_{59} = |5u_2 + u_0 + 8u_1|_{59}$$

We can implement this operation in the few steps shown below in different colours. These operations are performed in the sequence: blue, green, magenta. The sum is split into two parts, $h = h_2 2^2 + h_1 2 + h_0$ and $s = s_2 2^2 + s_1 2 + s_0$, which represent the next state of the circuit (actually, u_2^+ and u_1^+, or, more precisely, $p^+ = 8u_2^+ + u_1^+$).

The circuit (with the feed-forward carry out signal) is presented in Figure 24. The feedback colored in red insures the correctness of $\mathrm{mod}\ 59$ operation. When implementing the $g_0 = 5$ multiplier, we took into account that

$$\begin{aligned} u_2 5 &= (u_2^{(2)}2^2 + u_1^{(2)}2 + u_0^{(2)})5 = u_2^{(2)}2^4 \\ &\quad + u_1^{(2)}2^3 + (u_0^{(2)} + u_2^{(2)})2^2 + u_1^{(2)}2 + u_0^{(2)} \\ c_3'''2^6 &= c_3'''5 \bmod 59 \end{aligned}$$

where c_3''' is the carry-out signal produced by the most significant digit of the $\mathrm{mod}\ 59$ adder.

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ & & & & c_3''' & & c_3''' \\ c_3''' & c_2''' & c_1''' & c_3'' & c_2'' & c_1'' & \\ & c_5' & c_4' & c_3' & & & \\ + & & u_2^{(2)} & u_1^{(2)} & u_0^{(2)} & & \\ & & & & u_2^{(2)} & u_1^{(2)} & u_0^{(2)} \\ & u_2^{(1)} & u_1^{(1)} & u_0^{(1)} & u_2^{(0)} & u_1^{(0)} & u_0^{(0)} \\ \hline & h_2 & h_1 & h_0 & s_2 & s_1 & s_0 \end{array}$$

Feeding the carry out signal forward eliminates the race condition, but it results into two extra adders. If the carry out signal is fed back, the circuit becomes more compact (see Figure 25). However, the simulation software (such as, Altera Quartus II) refuses to simulate the combinational part of

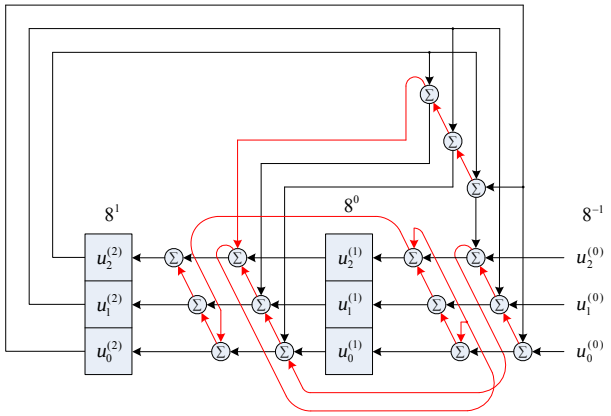

Fig. 24. Low cost $\mathrm{mod}\ 59$ arithmetic 2-stage 3-input SA

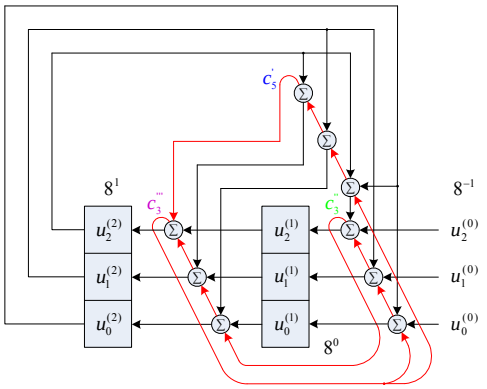

Fig. 25. The $\mathrm{mod}\ 59$ generator with feedback

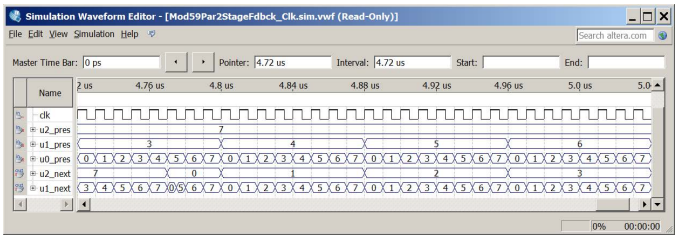

Fig. 26. Simulation results for the combinational part of the circuit of Figure 25 performing the operation $|(u^{(2)} \times 8 + u^{(1)}) \times 8 + u^{(0)})|_{59}$

this circuit. To resolve the problem, a flip-flop was inserted in the feedback. The simulation results completely matched those for Figure 24. They are partially presented in Figure 26. The combinational circuit computes the next state of the two-digit octal register by performing the following operation: $|(u^{(2)} \times 8 + u^{(1)}) \times 8 + u^{(0)})|_{59}$. The simulation demonstrated the perfect match between the true (desired) values of the $\mathrm{mod}\ 59$ residues and the actual values produced by the circuit. For example, $|(7 \times 8 + 4) \times 8 + 3 = 1 \times 8 + 3|_{59}$.

The entire circuit presented in Figure 25 (including the registers, u_2 and u_1) was simulated with the inclusion of a flip-flop into the feedback. The simulation results are presented in Figure 27. The flip-flop is triggered by the falling edge of the clock; and the input data are updated at the rising edge. As it can be seen from the diagram, after shifting in the sequence of octal numbers, 5321407601234567, the result of the modulo division is $u_2^+ = u_1^+ = 4$, i.e. $4 \times 8 + 4 = 36_{10}$. Indeed, $5321407601234567_8 \bmod 59_{10} = 36_{10}$.

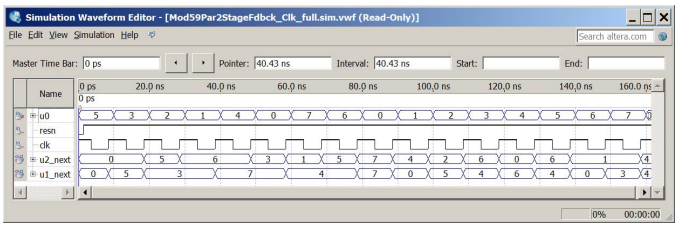

Fig. 27. Simulation results for the entire circuit of Figure 25 with a flip-flop in the feedback

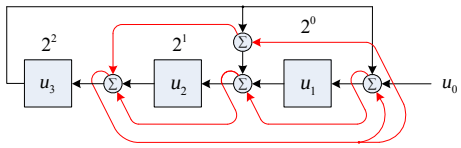

Fig. 28. Arithmetic length-3 binary SA revised; $g = 5$

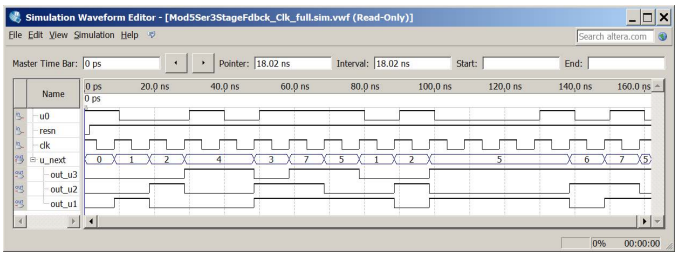

Fig. 29. Simulation results for the circuit of Figure 28: the sequence 1001011101000101 is shifted in (with each shift equivalent to multiplication of the entire register content by $2 \bmod 5$) resulting in $|0|_5$

Example 4.10 (an arithmetic 3-stage 1-input SA - Example 4.1 revised)**.** We will now re-design the $\mathrm{mod}\ 5$ generator circuit (introduced in Example 4.1) using the approach of Figure 17. Once again, $r = 3$ and $g(b) = 5$.

The operation performed by the circuit is:

$$p^+ = |2(4u_3 + 2u_2 + u_1) + u_0|_5 = |3u_3 + 4u_2 + 2u_1 + u_0|_5$$

This operation is implemented by the circuit shown in Figure 28 (note the similarity with the circuit of Figure 4). The simulation results for the circuit of Figure 28 are presented in Figure 29. The following binary sequence is shifted in to the circuit: $1001011101000101_2 = 38725_{10}$. Each shift is equivalent to the multiplication of the entire register content by $2 \bmod 5$. The final result of this modulo operation is $u_3^+ u_2^+ u_1^+ = 101_2 = 5_{10}$. Indeed, $38725_{10} \bmod 5_{10} = 0$.

The hardware complexity of the circuit in Figure 28 slightly exceeds the complexity of the circuit of Figure 11. However, the circuit of Figure 28 is more structured and easier to describe in a hardware description language.

Simulation results for the circuits of Figures 28 and 11 complitely match each other (this can be verified by observing the register content in Figure 29 after shifting in only 4 bits of the sequence, i.e. 1001; the residue left in the register is, indeed, $4 = 9 \bmod 5$).

4.1.3 *Further Improvement of Aliasing Rate*

According to Corollary 4.1.2, the aliasing rate for a 1-stage low cost residue computing circuit with $g = (2^m)^r - 1 =$

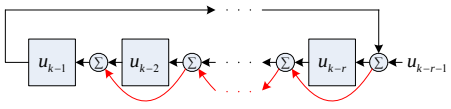

Fig. 30. Low cost $\bmod [(b+1)^r - 1]$ SA

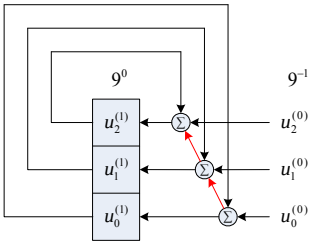

Fig. 31. 3-bit $\bmod 8$ checksum generator: $(u_1 \times 9 + u_0) \bmod 8$

$b^r - 1$ equals to $(b^r - 1)^{-1}$. To further improve the aliasing rate, we could increase the modulus from $b^r - 1$ to b^r. However, as it was observed earlier, this type of modulus is not preferable due to low error detecting capabilities (the corresponding signature analyzer does not have feedbacks). In order to preserve error detecting capabilities while extending the modulus size, we can increase the base of the system by 1. That is, we use the following number system:

$$u = u_{k-1}(b+1)^{k-1} + u_{k-2}(b+1)^{k-2} + \ldots + u_0$$
$$g = (b+1)^r - 1$$
$$p = u_{k-1}(b+1)^{r-1} + u_{k-2}(b+1)^{r-2} + \ldots + u_{k-r}$$

where the polynomial coefficients range from 0 to b. And $p^+ = |(b+1)p + u_{k-r-1}|_{(b+1)^r-1} = |u_{k-2}(b+1)^{r-1} + \ldots + (u_{k-1} + u_{k-r-1})|_g$.

The circuit implementing this expression is shown in Figure 30. It does not have a carry-out feedback.

If $r = 1$, the circuit transfers to the device known as a *modulo adder*, *checksum generator*, *accumulator*, *integrator*.

Example 4.11 (a 3-input checksum generator)**.** Let $r = 1$ and $g = (b+1)^r - 1 = (2^3+1)^1 - 1 = 8$. A 3-bit checksum generator is shown in Figure 31. Each shift is equivalent to multiplication of its content by $9 \mod 8$.

Proposition 4.1.4. *All properties of a checksum generator are inherited from the properties of the corresponding residue code.*

Proof: The proof directly follows from the above discussion. □

We can choose a more general form of the base, such as $b + g_0^* = 2^m + g_0^*$, where $0 \le g_0^* < 2^m$. Then, the modulus g equals: $g = (b + g_0^*)^r - g_0$. And, the partial residue is: $p^+ = |(b + g_0^*)p + u_{k-r-1}|_{(b+g_0^*)^r - g_0} = |u_{k-2}(b + g_0^*)^{r-1} + \ldots + (u_{k-1}g_0 + u_{k-r-1})|_g$.

The implementation of this analyzer is the same as represented in Figure 16 with the only difference that each shift of the register is equivalent to multiplication by $2^m + g_0^*$.

Based on the above expression, we can introduce a few more novel circuits.

Case 1. If $g_0^* = g_0$ and $r = 1$, then $p^+ = |u_{k-1}g_0 + u_{k-r-1}|_{2^m}$. This will lead to the circuit of Figure 16, but without the red carry-out feedback. And if $g_0 = 1$ the upper g_0-multiplier will turn into a straight line. The circuit also has the lowest aliasing rate, since the modulus 2^m ensures the largest possible range for m-bit residues.

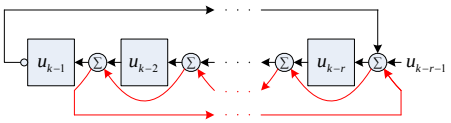

Fig. 32. Low cost $\bmod [(b-1)^r + 1]$ SA

Case 2. If $g_0^* = g_0 = 1$ and $r \ge 1$, then $p^+ = |u_{k-2}(b+1)^{r-1} + \ldots + (u_{k-1} + u_{k-r-1})|_{(b+1)^r-1}$. The equality $|(b+1)^r|_{(b+1)^r-1} = 1$ will then lead to the circuit of Figure 15.

Case 3. The coefficients g_0 and g_0^* are negative. That is, the base is $b - g_0^*$ and the modulus is $g = (b - g_0^*)^r + g_0$.

Respectively

$$p^+ = |(b - g_0^*)p + u_{k-r-1}|_{(b-g_0^*)^r+g_0}$$
$$= |u_{k-2}(b - g_0^*)^{r-1} + \ldots + (-u_{k-1}g_0 + u_{k-r-1})|_g$$

We only consider the case $g_0^* = g_0 = 1$, since it ensures the lowest hardware complexity and aliasing rate. Then, $p^+ = |u_{k-2}(b-1)^{r-1} + \ldots + (-u_{k-1} + u_{k-r-1})|_{(b-1)^r+1}$. Taking into account that $|(b-1)^r|_{(b-1)^r+1} = -1$, the above equation leads to the implementation shown in Figure 32. Because CMOS logic gates are realized with inverted outputs, the proposed circuit may save hardware.

If $r = 1$, then the base equals to 2^m. The red carry-out feedback in Figure 32 disappears and the carry-in signal to the right most digit becomes a constant "1".

Another useful property of this circuit can be derived from the following observation. For simplicity, we assume that the information sequence contains $k = 3r$ digits. Then, introducing notation $B = b - 1$ and taking into account (31),

$$|u(B)|_{g(B)} = |u_{3r-1}B^{3r-1} + \ldots + u_0|_{B^r+1} = |u_{3r-1}B^{r-1}$$
$$+ \ldots + u_{2r} - \ u_{2r-1}B^{r-1} - \ldots - u_r + \ u_{r-1}B^{r-1}$$
$$+ \ldots + u_0|_{B^r+1} = |(u_{3r-1} - u_{2r-1} + u_{r-1})B^{r-1}$$
$$+ \ldots + (u_{2r} - u_r + u_0)|_{B^r+1}$$

This equation shows that the analyzer adds the distant r symbols with alternating signs. One possible use of this property is that if the sequence of digits is disturbed by an additive noise, the overall effect of the noise to the signature will be neutralized (provided that the number of r-digit sets in the incoming sequence is even).

Example 4.12 (a low cost 1-stage $\bmod 7$ SA)**.** Let $r = 1$ and $g = b^r + g_0 = 6^1 + 1 = 7$. The circuit that implements this analyzer is presented in Figure 33a. Each shift of this circuit/register is equivalent to multiplication of its content by $6 \mod 7$. Or, more precisely, $|u_1 \times 6 + u_0|_7 = |-u_1 + u_0|_7$.

If we feed this analyzer, for example, by the sequence $101, 110, 011$, the signature will be 010. The binary integers $101, 110, 011$ correspond, respectively, to the decimal integers $-2, -1, 3$. Thus, the resulting signature (in decimal form) will be: $(-2) - (-1) + 3 = 2$ (or 010 in binary).

If the sequence was $101, 110, 011, 000$, the result would be 101. Indeed, $-(-2) + (-1) - 3 + 0 = -2$, and $-2 = 5 \mod 7$. And if this sequence is disturbed by additive noise, e.g., +1, the signature will be the same: $-(-2+1) + (-1+1) - (3+1) + (0+1) = -2$.

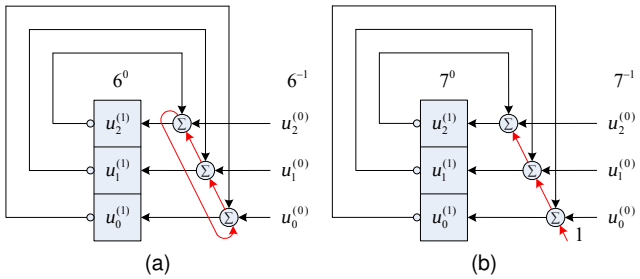

Fig. 33. Low cost 1-stage SAs with (a) $\bmod 7$ and (b) $\bmod 8$

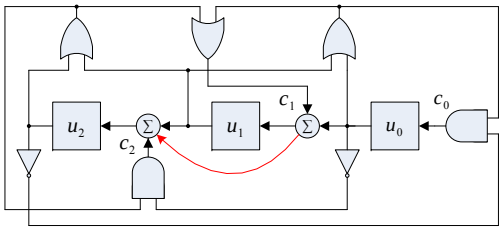

Fig. 34. $(-2)^i = 5^i \bmod 7$ - multiplier used as a pseudo-random number generator; any non-zero 3-bit number placed into this register will go through all 6 possible combinations when the register is clocked

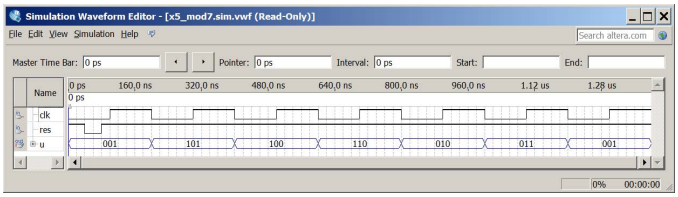

Fig. 35. Simulation of $\times 5 \bmod 7$ residue generator shown in Figure 34 with the seed value of 001; for example, $|110 \times 5 = 010|_7$

Example 4.13 (a low cost 1-stage $\bmod 8$ SA)**.** Let $r = 1$ and $g = (b-1)^r + 1 = (2^3-1)^1 + 1 = 8$. This analyzer is shown in Figure 33b. Each shift of this analyzer is equivalent to the following operation: $|u_1 \times 7 + u_0|_8 = |-u_1 + u_0|_8$.

If we feed this analyzer by the sequence used in the previous example, $101, 110, 011, 000$, the signature will be 110. Indeed, $-(-3) + (-2) - 3 + 0 = -2 = 6 \,(\mathrm{mod} 8)$.

Example 4.14 (a pseudo-random number generator)**.** If α is a primitive element of a finite field $GF(q)$, then each field element can be written as α^i for some integer i. That is, consecutive powers of a primitive element will generate all the non-zero elements of a field. Integer -2 is a primitive element in $GF(7)$. Note that $(-2)^i = (+5)^i \bmod 7$. We will now build a circuit, where each shift is equivalent by multiplication of its content by $(-2) \bmod 7$. The circuit is able to generate all the non-zero $GF(7)$ field elements by rising the content into consecutive powers. We can start from a primitive element itself, or from any other field element. The circuit is presented in Figure 34.

The logic expressions for the signals c_2, c_1, c_0 are:

$$c_2 = \bar{u}_0(u_1 + u_2); \quad c_1 = u_0 + u_1 + u_2; \quad c_0 = \bar{u}_2(u_0 + u_1)$$

Simulation results for the circuit of Figure 34 are presented in Figure 35. Here $q(2)q(1)q(0)$ is the (octal) signal at the outputs $u_2u_1u_0$.

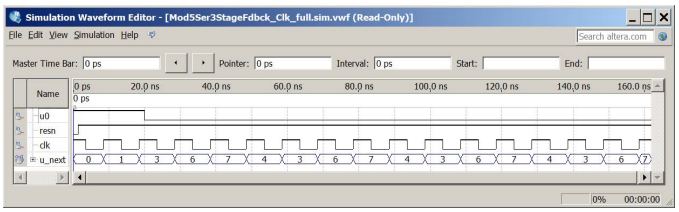

Fig. 36. Simulation of a pseudo-random number generator: each shift is equivalent to multiplication by $3 \bmod 5$

Apparently, this circuit is a pseudo-random (octal) number generator. We can also call it a $[(-2)^i \bmod 7]$-multiplier.

Note that the field $GF(7)$ contains $q = 7$ elements, where q is a prime number and not a power of 2. So that algebraic polynomials can not be utilized here.

Generally, we can select any modulo computing circuit, place a primitive element into it and start shifting it. The circuit will then generate pseudo-random numbers. For example, if we select the circuit of Figure 28, it will produce the following sequence of numbers: $3, 1, 2, 4, 3, 1, 2, 4, \ldots$. This is shown in Figure 36. Here the primitive element is the same, $-2 = 3 \bmod 5$. The first two clocks are used to shift in two $1's$ resulting in $3(= -2)$. Then the register input turns into 0 and the content is shifted left: $|3 \times 2 = 6 = 1|_5$, $|6 \times 2 = 7 = 2|_5$, $|2 \times 2 = 4|_5$, $|4 \times 2 = 3|_5$, $|3 \times 2 = 6 = 1|_5$.

4.2 Mixed-Signal Arithmetic Channel

Among all mixed-signal devices (carrying both continuous and discrete signals), we consider only those having discrete outputs (such as analog-to-digital converters, measurement instruments, etc.). These devices can potentially be tested by any of the analyzers introduced in sections 3.1 and 4.1.

In contrast to digital circuits, the output response of a mixed-signal CUT that is fed by a test stimulus is distorted even for the fault free CUT. This, in turn, distorts the fault free circuit's signature into a range of reference signatures. We determine the operational status of the CUT by verifying whether or not the actual signature drops into this range.

If we employ algebraic signature analyzers of section 3.1, the range of reference signatures will contain gaps. This increases complexity of the verifying device (further referred to as a comparator) which must now contain a memory to store all the reference signatures. If the range was contiguous, we could use a *window* comparator (that only requires two signatures, *lower* and *upper* ones) to verify that the actual signature drops into the range. One solution to make the range contiguous is to rearrange the set of reference signatures [28]. Another solution is the use of any arithmetic signature analyzers examined in section 4.1. Under certain conditions, these analyzers produce a contiguous signature range. In this section, we will focus on this particular case. Two questions of interest arise thereupon: what is the aliasing rate and what is the reference signature range?

4.2.1 Aliasing Rate

Proposition 4.2.1. *The aliasing rate of the modulo g signature analyzer (Figure 10 or Figure 17) used for compaction of output signals of a mixed-signal circuit, is upper bounded by g^{-1}:*

$$P_{nd} \leq g^{-1}$$

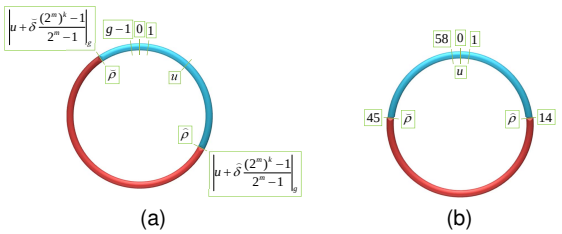

Fig. 37. Reference signature ring for (a) general case (b) Example 4.15

provided that the sequence of k 2^m-ary symbols to be compacted into an r-symbol signature is long and all error patterns in the sequence are equally likely.

Proof: The polynomial to be compacted and the compaction modulus are: $u = u_{k-1}(2^m)^{k-1} + u_{k-2}(2^m)^{k-2} + \ldots + u_0$ and $g = g_{r-1}(2^m)^{r-1} + g_{r-2}(2^m)^{r-2} + \ldots + g_0$, where $g_j < 2^m$, $j = 0, \ldots, r-1$.

For a fault-free mixed-signal circuit, the actual value of the digit, u_i^a, $i = 0, \ldots, k-1$, must fall into the predefined interval, $u_i + \check{\delta} \le u_i^a \le u_i + \hat{\delta}$, where u_i is the *nominal* value of the digit, and $\check{\delta}$, $\hat{\delta}$ are, respectively, the lower and upper tolerance bounds: $\check{\delta} \le 0 \le \hat{\delta}$. Then, the total number of errors in the sequence of k 2^m-ary digits (output responses) is $2^{mk} - (|\check{\delta}| + \hat{\delta} + 1)^k$. The number of errors which are not detected by the code that uses modulus g is $\lfloor 2^{mk}/g \rfloor - (|\check{\delta}| + \hat{\delta} + 1)^k$. Therefore, the aliasing rate,

$$P_{nd} \le \frac{1/g \cdot [2^m/(|\check{\delta}| + \hat{\delta} + 1)]^k - 1}{[2^m/(|\check{\delta}| + \hat{\delta} + 1)]^k - 1}$$

Taking into account that $|\check{\delta}|$, $\hat{\delta}$ are small compared to 2^m, k is large and g is relatively small, we obtain $P_{nd} \le g^{-1}$. □

It follows from this proposition that aliasing rate decreases with the grows of the compaction modulus, g.

4.2.2 Reference Signature Range

Proposition 4.2.2. *Let $u_i, i = 0, \ldots, k-1$ be a nominal 2^m-ary digit and let u_i^* be the interval digit defined as $u_i^* = u_i + \delta$, where δ is the interval (called a deviation) defined by the left and right endpoints $\check{\delta}$ and $\hat{\delta}$, $\delta = [\check{\delta}, \hat{\delta}]$.*

Then, the range of the reference signatures is the interval, ρ, with the left and right endpoints $\check{\rho}, \hat{\rho}$, $\rho = [\check{\rho}, \hat{\rho}]$, where:

$$\check{\rho} = \left| u + \check{\delta}\frac{(2^m)^k - 1}{2^m - 1} \right|_g, \quad \hat{\rho} = \left| u + \hat{\delta}\frac{(2^m)^k - 1}{2^m - 1} \right|_g$$

Proof: The reference signature is computed as follows: $\rho = |(u_{k-1} + \delta)(2^m)^{k-1} + \ldots + (u_1 + \delta)2^m + (u_0 + \delta)|_g = |u + \delta(2^m)^{k-1} + \delta(2^m)^{k-2} + \ldots + \delta 2^m + \delta|_g = \left| u + \delta\frac{(2^m)^k - 1}{2^m - 1} \right|_g$

Hence, the reference signatures interval, $\rho = [\check{\rho}, \hat{\rho}]$, and $\check{\rho} = \left| u + \check{\delta}\frac{(2^m)^k - 1}{2^m - 1} \right|_g$, and $\hat{\rho} = \left| u + \hat{\delta}\frac{(2^m)^k - 1}{2^m - 1} \right|_g$ □

The endpoints of the reference signature interval are shown in Figure 37a. The ring appears because of modular operations. If the actual signature, u^a, falls into the red region, the circuit is faulty. Otherwise, it is considered to be fault-free (with the aliasing defined in Proposition 4.2.1).

Example 4.15 (signature range for mod 59 SA)**.** Let us choose the signature analyzer of Figure 25 with $g = 59$. We will shift in $k = 3$ octal digits, and assume that the nominal signature for these digits is $u = 0$. Set $|\check{\delta}| = \hat{\delta} = 1$, then, based on Proposition 4.2.2:

$$\check{\rho} = \left| 0 - \frac{(2^3)^3 - 1}{2^3 - 1} \right|_{59} = 45; \quad \hat{\rho} = \left| 0 + \frac{(2^3)^3 - 1}{2^3 - 1} \right|_{59} = 14$$

The end points of the reference signatures interval are shown in Figure 37b. The sets of reference signatures occupy the following symmetrical (with respect to 0) intervals on the ring: $1 - 9, 12 - 14$ and $45 - 47, 50 - 58$.

Corollary 4.2.1. *For a low cost analyzer with $g = 2^{mr} - 1$, the end points of the interval of reference signatures are:*

$$\check{\rho} = \left| u + \check{\delta}\,\frac{k}{r}\,\frac{(2^m)^r - 1}{2^m - 1} \right|_g, \quad \hat{\rho} = \left| u + \hat{\delta}\,\frac{k}{r}\,\frac{(2^m)^r - 1}{2^m - 1} \right|_g$$

Proof: The signature is computed as:

$$\begin{aligned} |u|_g &= |u_{k-1}(2^m)^{k-1} + \ldots + u_0|_g \\ &= |(2^m)^{k-r}[u_{k-1}(2^m)^{r-1} + \ldots + u_{k-r}] + \ldots \\ &+ (2^m)^0[u_{r-1}(2^m)^{r-1} + \ldots + u_0]|_g \end{aligned}$$

For simplicity, we assume that k is multiple of r, that is $k = \lambda r$, where λ is an integer. Then, the signature $|u|_g$ is:

$$\begin{aligned} &|(2^m)^{r(\lambda-1)}[u_{k-1}(2^m)^{r-1} + \ldots + u_{k-r}] + \ldots \\ &+ (2^m)^0[u_{r-1}(2^m)^{r-1} + \ldots + u_0]|_{2^{mr}-1} \\ &= |(u_{k-1} + u_{k-r-1} + \ldots + u_{2r-1} + u_{r-1})(2^m)^{r-1} \\ &+ \ldots + (u_{k-r} + u_{k-2r} + \ldots + u_r + u_0)|_{2^{mr}-1} \end{aligned}$$

Introducing notations:

$$U_i = u_{(\lambda-1)r+i} + \ldots + u_{2r+i} + u_{r+i} + u_i$$

for $i = 0, \ldots, r-1$, we obtain:

$$|u|_g = |U_{r-1}(2^m)^{r-1} + \ldots + U_1 2^m + U_0|_{2^{mr}-1} \qquad (39)$$

The modular operation here only handles the effect of the carry out signal.

By the same reasoning, introducing $\Delta = \delta\lambda$, we obtain:

$$|\delta|_g = |\Delta(2^m)^{r-1} + \ldots + \Delta 2^m + \Delta|_{2^{mr}-1} = |\delta\lambda[(2^m)^{r-1} + \ldots + 2^m + 1]|_{2^{mr}-1} = \left| \delta\,\frac{k}{r}\,\frac{(2^m)^r - 1}{2^m - 1} \right|_{2^{mr}-1}$$

Taking into account this equation and equation (39),

$$\rho = |u + \delta|_{2^{mr}-1} = \left| u + \delta\,\frac{k}{r}\,\frac{(2^m)^r - 1}{2^m - 1} \right|_{2^{mr}-1} \qquad (40)$$

where u is computed using equation (39). Then, the proof directly follows from equation (40). □

Example 4.16 (signature range for mod 63 SA)**.** Let us choose the signature analyzer of Figure 23 with $g = 63$, and $r = 2$. We shift in $k = 4$ octal digits, and assume again that the nominal signature for these digits is $u = 0$. Set $|\check{\delta}| = \hat{\delta} = 1$, then, based on Corollary 4.2.1, we obtain the following estimates: $\check{\rho} = 45$ and $\hat{\rho} = 18$.

The sets of reference signatures occupy the following intervals: $1 - 2, 6 - 10, 14 - 18$ and $45 - 49, 53 - 57, 61 - 62$.

As we have seen, the range $[\check{\rho}, \hat{\rho}]$ is not always contiguous, which complicates the testing device (bringing the complexity closer to the algebraic analyzer case). The contiguousness is certainly achieved for the modulus $g = 2^{mr} - 1$,

with $r = 1$ (i.e., one-stage analyzers). For example, if $m = 3$, $k = 2, |\check{\delta}| = \hat{\delta} = 1$, then $\check{\rho} = 5, \hat{\rho} = 2$, and the reference signatures interval is $5, 6, 0, 1, 2$. However, the analyzer with these parameters (as the one of Figure 18) does not detect error patterns of the form $+t, -t$, where $t < 2^m$. The same is true for the algebraic analyzer with the generator polynomial over $GF(2^m)$ of the form $g(x) = x^r + 1, r = 1$ (like the one in Figure 6a). This follows from the fact that $|2^m|_{2^m-1} = 1$ and $|x|_{x+1} = 1$. For example, if the first and second digits of the following octal polynomial are, respectively, corrupted by +2 and -2, the code with the generator $g = 7$ (Figure 18) will not detect the errors, while the one with $g = 5$ and almost the same hardware complexity (Figure 28) will do: $|5{\cdot}8^2{+}3{\cdot}8{+}4|_7 = 5$, and $|5{\cdot}8^2{+}3{\cdot}8{+}4|_5 = \mathbf{3}$. While, in case of errors: $|5 \cdot 8^2 + (3 + 2) \cdot 8 + (4 - 2)|_7 = 5$, and $|5 \cdot 8^2 + (3 + 2) \cdot 8 + (4 - 2)|_5 = \mathbf{2}$.

4.2.3 Bound on Number of Responses to be Compacted

The validity of a pass/fail decision by an arithmetic signature analyzer, depends on the number of digits to be compacted. It is important to estimate the maximum number of digits that can be shifted into the analyzer, while a reliable pass-fail decision can yet be made.

Proposition 4.2.3. *The maximum number of digits, k, that can be shifted into a $\bmod g$ signature analyzer used for testing of the $[\check{\delta}, \hat{\delta}]$ mixed-signal channel, is defined as:*

$$k = \lfloor \log_{|\check{\delta}|+\hat{\delta}+1} g \rfloor$$

Proof: Each "fault-free" digit shifted into the analyzer produces one of the $|\check{\delta}|+\hat{\delta}+1$ possible reference signatures (1 in this expression corresponds to the "nominal" signature). If k digits are shifted in, the number of possible reference signatures is $(|\check{\delta}|+\hat{\delta}+1)^k$. This number must not exceed the modulus: $(|\check{\delta}| + \hat{\delta} + 1)^k \leq g$. Or, $k = \lfloor \log_{|\check{\delta}|+\hat{\delta}+1} g \rfloor$ □

It follows from this Proposition that the number, k, of digits that could be reliably compacted increases with the growth of the modulus and contraction of the tolerance bounds. In this regard, when testing an analog-to-digital converter, it is suggested to apply the test stimuli that coincide with the transition points between code levels of the converter's transfer function. The tolerance bounds will then shrink from $[\check{\delta}, \hat{\delta}]$ to $[0, \hat{\delta}]$.

Example 4.17 (an estimate of the maximum number of octal digits to be compacted by a $\bmod 59$ SA)**.** Let us consider the mixed-signal channel with the parameters of Example 4.15.

Using Proposition 4.2.3, we derive: $k = \lfloor \log_3 59 \rfloor = 3$, meaning that we can safely shift 3 digits into the analyzer. The number of possible reference signatures after this shift becomes $3^3 = 27$, which is less than 59. So that, the "fault-free" and "faulty" signature sets can still be distinguished.

If we selected 4 digits, the number of reference signatures would increase to $3^4 = 81$, which overlaps the modulus 59. Therefore, no true pass/fail decision is possible.

Note that Proposition 4.2.3 defines the bound on the maximum number of digits that can be safely shifted into the analyzer. The actual number of allowable digits may be higher (depending on the concrete values of m, r, g, k). For example, if $m = 3, g = 7, |\check{\delta}| = \hat{\delta} = 1$, then it follows from the Proposition that $k = \lfloor \log_3 7 \rfloor = 1$. However, as we have seen, if the number of digits is 2, then the set of reference signatures is $5, 6, 0, 1, 2$ and they are still distinguishable from the faulty circuit signatures. This happens because some of the reference signatures overlap with others, thus, narrowing the reference signature range and allowing the number of digits, k, to be increased.

5 Conclusion

A polynomial division circuit is an important part of any technical system that exercises principles of the theory of error-control coding. These circuits are used in high reliability communication, computing, test, cryptography and other systems. All the variety of polynomial division circuits is split into two classes, algebraic and arithmetic circuits. The design procedures for them are distinct and considered separately. In our work, we investigated these two classes of circuits from common perspective and presented the theory and design techniques for them. The similarity between the two circuit classes was revealed. One of the benefits of this similarity is that arithmetic signature analyzers can be designed directly by observing the modulus polynomial, much in the same way as for algebraic analyzers. We showed how to utilize the polynomial division circuits for testing digital and mixed-signal devices. The approach is applicable to an arbitrary algebraic/arithmetic system with any number base. We also demonstrated how to eliminate the race condition in the arithmetic circuits, which may occur because of the asynchronous feedback introduced by the carry propagation circuitry. Several numeric characteristics/bounds for polynomial division circuits were introduced and estimated (including the aliasing rate for mixed-signal systems).

Although the main objective of this work was the application of the polynomial division circuits for testing, the proposed circuits can also be used in communication channels as a part of coders/decoders. As well, they can be utilized in arithmetic devices protected by arithmetic error-control codes (including computing systems, analog-to-digital converters and digital filters implemented in a residue number system).

While the design procedures for algebraic and arithmetic polynomial division circuits elaborated in this work look similar, they are yet distinct. The future work carried out in this area could result in the development of a unified design procedure applicable to both types of circuits.

The other anticipated implications include the following:

- Since the operation of arithmetic polynomial division circuits is similar to the operation of digital filters, the development of a unified digital systems processing algorithm for designing both of these devices could be considered. The filter based approach has been used for designing algebraic encoders/decoders. The counterpart for arithmetic devices is yet to be developed.
- As it follows from the work, it is more difficult to design an arithmetic polynomial division circuit rather than an algebraic one. It is anticipated, that simpler and lower-cost arithmetic circuits with a good aliasing rate will be developed in the future. It is also anticipated that the aliasing rate bounds

will be "narrowed", making the rate estimates more accurate.

- When arithmetic polynomial division circuits are utilized for mixed-signal systems testing, the aliasing rate rapidly increases with the growth of the number of patterns to be compressed (due to the unavoidable quantization error). It may become more economical to "shift" the execution of modular operations from a digital to an analog part of the circuit. This will lead to the development of analog filters working in a residue number system. These filters could possess improved characteristics (like speed, accuracy, etc.).
- We gradually demonstrate how to use error-control codes for testing and propose the appropriate model. In our work, we only consider error-detection block codes. However, other codes could also be investigated (e.g., non-block convolutional codes, or block error-locating codes that would locate faulty units/scan chains in a device under test).
- It is expected that the work will narrow down the gap between the abstract math and practical engineering.

The outcomes of the work are anticipated to have an impact on the researchers and engineers working in the fields of reliable digital computing (including re-configurable computing), noise-immune data communication, built-in self-test, cryptography, and test & measurement area.

ACKNOWLEDGMENT

This research has been supported by the Ryerson University Faculty of Engineering and Architectural Science Dean's Research Fund.

The author would like to thank Dr. V. Kneller and Dr. V. Dyn'kin, Institute of Control Science, RAS, for their creative ideas, boundless inspiration and invaluable advise.

REFERENCES

[1] W. Peterson and E. Weldon, *Error Correcting Codes*. Cambridge, MA: The MIT Press, 1972.
[2] T. Rao, *Error Coding for Arithmetic Processors*. New York, NY: Academic Press, 1974.
[3] G. Starr, Q. Jie, B. Dutton, C. Stroud, F. Dai, and V. Nelson, "Automated generation of built-in self-test and measurement circuitry for mixed-signal circuits and systems," in *Proc. 15th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2009, pp. 11–19.
[4] J. Rajski and J. Tyszer, "The analysis of digital integrators for test response compaction," *IEEE Trans. Circuits Syst.*, vol. 39, no. 5, pp. 293–301, 1992.
[5] L. Wei and L. Jia, "An apprach to analong and mixed-signal BIST based on pseudo-random testing," in *Proc. International Conference on Communications, Circuits and Systems*, 2008, pp. 1192–1195.
[6] M. Poolakkaparambil, J. Mathew, A. Jabir, D. Pradhan, and S. Mohanty, "BCH code based multiple bit error correction in finite field multiplier circuits," in *Proc. Int. Symp. Quality Electronic Design*, 2011, pp. 1–6.
[7] K. Chakrabarty, "Low-cost modular testing and test resource partitioning for SOCs," *Proc. IEE Comput. Digit. Techn*, vol. 152, no. 3, pp. 427–441, 2005.
[8] I. Voyiatzis, A. Paschalis, D. Gizopoulos, N. Kranitis, and C. Halatsis, "A concurrent built-in self-test architecture based on a self-testing RAM," *IEEE Trans. Rel.*, vol. 54, no. 1, pp. 69–78, 2005.
[9] K. Tsoumanis, C. Efstathiou, and N. Moschopoulos, "On the design of modulo $2^n \pm 1$ residue generators," in *Proc. IFIP/IEEE 21st Int. Conf. on VLSI-SoC*, 2013, pp. 33–38.
[10] S. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Trans. Comput.*, vol. 423, no. 1, pp. 68–77, 1994.
[11] G. Redinbo, "Protecting data compression: arithmetic coding," *Proc. IEE Comput. Digit. Techn*, vol. 147, no. 4, pp. 221–228, 2000.
[12] A. Ahmad and L. Hayat, "On design of 16-bit signature analyzer circuits equipped with primitive characteristic polynomials," in *Proc. 1st Taibah University Int. Conf. Computing and Information Technology*, 2012, pp. 660–664.
[13] P. Wohl, J. Waicukauski, S. Patel, C. Hay, E. Gizdarski, and B. Mathew, "Hierarchical compactor design for diagnosis in deterministic logic BIST," in *Proc. VLSI Test Symposium*, 2005, pp. 359–365.
[14] T. Indlekofer, "Signature rollback with extreme compaction - a technique for testing robust VLSI circuits with reduced hardware overhead," *Annales Univ. Sci. Budapest., Sect. Comp.*, vol. 39, pp. 161–180, 2013.
[15] N. Badereddine, Z. Wang, P. Girard, K. Chakrabarty, A. Virazel, S. Pravossoudovitch, and C. Landrault, "A selective scan slice encoding technique for test data volume and test power reduction," *J Electron Test*, vol. 24, pp. 353–364, 2008.
[16] T. Hiraide, K. Boateng, H. Konishi, K. Itaya, M. Emori, H. Yamanaka, and T. Mochiyama, "BIST-aided scan test - a new method for test cost reduction," in *Proc. VLSI Test Symposium*, 2003, pp. 359–364.
[17] K. Namba, Y. Matsui, and H. Ito, "Test compression for IP core testing with reconfigurable network and fixing-flipping coding," *J Electron Test*, vol. 25, pp. 97–105, 2009.
[18] X. Tang and S. Wang, "A low hardware overhead self-diagnosis technique using Reed-Solomon codes for self-repairing chips," *IEEE Trans. Comput.*, vol. 59, no. 10, pp. 1309–1319, 2010.
[19] C. Liu and K. Chakrabarty, "Design and analysis of compact dictionaries for diagnosis in scan-BIST," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 8, pp. 979–984, 2005.
[20] S. Wichlund, F. Berntsen, and E. Aas, "Scan test response compaction combined with diagnosis capabilities," *J Electron Test*, vol. 24, pp. 235–246, 2008.
[21] S. Lin and D. Costello, *Error Control Coding*. Upper Saddle River, NJ: Pearson Education, Inc., 2004.
[22] N. Szabo and R. Tanaka, *Residue Arithmetic and Its Application to Computer Technology*. New York, NY: McGraw-Hill Book Company, 1967.
[23] J. Stone, "Multiple-burst error correction with the Chinese Remainder theorem," *J. SIAM*, vol. 11, pp. 74–81, 1963.
[24] S. Hassen and E. McCluskey, "Increased fault coverage through multiple signatures," in *Proc. International Fault-Tolerant Computing Symposium*, 1984, pp. 354–359.
[25] G. Alefeld and G. Mayer, "Interval analysis: Theory and applications," *Journal of Computational and Applied Mathematics*, vol. 121, no. 1-2, pp. 421–464, 2000.
[26] S. Chang and L. Weng, "Error-locating codes," in *IEEE International Convention Record*, 1965, Part 7, pp. 252–258.
[27] U. Sparmann and M. Reddy, "On the effectiveness of residue code checking for parallel two's complement multipliers," *IEEE Trans. VLSI Syst.*, vol. 4, no. 2, pp. 227–239, 1996.
[28] V. Geurkov and L. Kirischian, "On the use of an algebraic signature analyzer for mixed-signal systems testing," *VLSI Design, Hindawi Publishing Corporation*, vol. 2014, no. 465907, pp. 1–8, 2014.

Vadim Geurkov received the M.S. degree in computer engineering from Georgian Polytechnic University, Tbilisi, Georgia, in 1981, and the Ph.D. degree in computer engineering from the Institute of Control Science, Moscow, Russia, in 1993. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, Canada. His research interests include design-for-test and built-in self-test for digital and mixed-signal systems.