Universal Reed-Solomon Decoders Based on the Berlekamp-Massey Algorithm [*]

Zhiyuan Yan
Department of Electrical and Computer Engineering
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015 USA
yan@lehigh.edu

Dilip V. Sarwate
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, IL 61801 USA
sarwate@uiuc.edu

ABSTRACT

Universal Reed-Solomon decoders allow flexibility in some or all of the code parameters, and hence lend themselves to applications wherein adaptability and reconfigurability are important considerations. In this paper, we consider universal finite field arithmetics using either canonical polynomial basis or shifted polynomial basis, and propose a high-speed universal Reed-Solomon decoder by modifying the high-speed decoder proposed in [10].

Categories and Subject Descriptors

B.2.4 [**ARITHMETIC AND LOGIC STRUCTURES**]: High-Speed Arithmetic

General Terms

Design

Keywords

Reed-Solomon codes, error-control codes, Berlekamp-Massey algorithm, VLSI architectures

1. INTRODUCTION

Reed-Solomon (RS) codes have long been used in applications ranging from consumer electronics such as compact disc (CD) [14, Chapter 4] to deep space communication [14, Chapter 3]. As a result of these applications, Reed-Solomon coding solutions, in the form of VLSI circuits, have been devised to meet various encoding/decoding requirements. At present, RS codes are being utilized or considered in various new applications such as cable modems, asymmetric digital subscriber line (ADSL), wireless communications, optical link, and digital video broadcast (DVB). These new applications pose more stringent requirements on the performances of RS codecs. Due to the ever higher data rates in many applications, RS codecs are often required to achieve very high throughputs. Another requirement with increasing importance is adaptability and reconfigurability of RS decoders. Both requirements can be satisfied by a high-speed universal RS decoder.

Universal RS decoders allow flexibility in some or all of the code parameters: the block length, the minimum distance (error correction capability), and rate of the code, and even the size of the underlying field and the generator primitive polynomial. Universal RS decoders are generally less hardware-efficient solutions when the coding scheme is *fixed*, but universal decoders are both adaptable and reconfigurable. The importance of adaptability is perfectly demonstrated by the incident of the Galileo spacecraft[1] [14, Chapter 3]. Reconfigurability of RS codecs becomes vital for the applications nowadays since a single RS codec often has to work with different RS codes for several reasons. One of the reasons is that a single chip developed today is often required to comply with more than one standard, and these standards use different RS codes. For example, the RS decoder in [8] supports $t = 8$, $(n, k) = (204, 188)$ or $(146, 130)$, or $t = 5$, $(n, k) = (125, 115)$ systematic RS codes since the chip is compatible with multiple standards for digital satellite TV systems. Also, the RS decoder designed for ADSL and cable modems in [7] works with different primitive polynomials $g(z)$ of degree 8 that are used in various ADSL and cable modem standards. Another reason for reconfigurability is that some coding schemes inherently involve several different RS codes: RS codes with different error correction capabilities are interleaved in unbalanced interleaving [14, Chapter 11], and different RS codes are used in concatenated RS codes and RS product-codes. Finally, adaptive

[*]This work was supported by the National Science Foundation under Grants CCR 99-79381 and ITR 00-85929.

[1]Launched in October 1989, Galileo is a NASA spacecraft whose mission was to explore Jupiter. In 1991, it was discovered that the high-gain X-band antenna aboard Galileo had failed and the NASA engineers had to make do with a low-gain S-band antenna. Among the changes made to make up for the loss, eight Reed-Solomon codes of different rates were interleaved in the concatenated code while the original plan was to interleave eight RS codewords of the same rate.

coding schemes which involve different RS codes are utilized to take advantage of the variations in channel conditions in wireless communications (see, for example, [3]) and optical communication systems. In all these cases, a single universal RS decoder capable of decoding many different codes can be a more hardware-efficient solution than *separate* decoders for each RS code although it may be less hardware-efficient than a RS decoder that is optimized with respect to *any individual* code.

All the RS decoders mentioned above are over $GF(2^8)$ and allow flexibility in the block length, rate and error correction capability. The decoder in [7] further allows different choices of $g(z)$ of degree 8. Additional flexibility and more choices of the parameters n, k, and t are possible if the field size is also flexible for the universal decoder. Such a decoder must use universal field arithmetic operations. That is, the field arithmetic operations work properly regardless of the size of the underlying field and the generator primitive polynomial. Besides universal RS decoders, universal field arithmetic operations are also useful in other applications. For example, using universal $GF(2^m)$ processors in cryptosystems allows changing cryptography schemes based on different field parameters without switching field processors [6].

There has been a steady stream of work on universal RS decoders and universal field arithmetic operations. Blahut [2] proposed two time-domain algorithms that do not require syndrome computations for universal decoders. Building on these time-domain algorithms, Shayan *et al.* proposed non-cellular [11] and cellular [12] universal decoders. These decoders use a fixed primitive polynomial of a fixed degree m, but can decode RS codes of different rates [13]. These decoders can also decode shortened and singly extended RS codes. Green and Drolet [5] further improved the versatility of the time-domain decoders in [11] by allowing m to have any value up to M and also allowing $g(z)$ to be any primitive polynomial of degree m. Turning to universal field arithmetic operations, Mastrovito [9] proposed a universal architecture based on the MSR multiplier. Green and Drolet adapted the systolic multiplication architecture in [15] to universal multiplication. A universal multiplier for which one operand uses the CPB and the other the triangular basis was proposed by Hasan and Ebtedaei [4]. Hasan and Wassal [6] proposed a universal $GF(2^m)$ processor that also uses operands based on the CPB and the triangular basis.

In this paper, the time-domain decoding approach is not pursued for the universal decoders proposed in this paper since the time-domain decoders involve field inversion inside the key equation solver (KES) loop (see [5]), which leads to a long delay. Instead, we adapt the high-speed architectures proposed in [10] (referred to as the SS architectures henceforth) to universal decoding so as to achieve high throughputs. Similar to the decoders by Green and Drolet [5], the universal RS decoders proposed here allow *all* code parameters to be flexible. Also, our universal RS decoder architectures can achieve high throughputs since the SS architectures can achieve very high throughputs, especially when the RS codewords are interleaved.

The rest of the paper is organized as follows. A new method for embedding the field element in universal architectures is proposed in Section 2. Interpretation of the registers with respect to two polynomial bases is also discussed in Section 2. Universal field additions and multiplications based on the new embedding method are discussed in Section 3. Universal field inversions and divisions are not discussed in this paper since the an inversion-less Berlekamp-Massey algorithm is used to solve the key equation. Universal RS decoder architectures are proposed in Section 4. Finally, the performance of the universal RS decoder architectures are discussed in Section 5.

2. EMBEDDING OF FIELD ELEMENTS IN UNIVERSAL ARCHITECTURES

In theory, the field size for *both* universal RS decoders *and* universal field arithmetic operations can be arbitrary. However, a limit for the size of the fields has to be set in practice and such a limit is often application-specific. For example, the finite fields used in RS codes are typically no larger than $GF(2^{12})$. Since symbols for the codes over $GF(2^8)$ can be represented conveniently with a byte, a limit of $GF(2^8)$ is often assumed (see, for example, [5]). However, much larger limits are usually required for cryptography applications (see, e.g.,[6]). In this paper, a maximum size of 2^M is assumed for the fields, and the numerical value of M depends on the application of interests.

M-bit registers are allocated to store field elements in universal architectures. However, the elements of $GF(2^m)$ ($m \leq M$) are represented by m-bit vectors. Thus, the questions naturally arise as to how to embed the m-bit vectors in the M-bit registers and how to interpret these registers. Most previously proposed universal architectures embed the m-bit vectors, based on either polynomial bases or triangular bases, at the *low* order end of the M-bit registers and fill the rest with zeros. The solution proposed here uses the m bits on the *high* order end to embed the polynomial-basis-based m-bit vectors and sets the rest of the bits in the register to zeroes. In other words, only the most significant m bits are used to embed the field elements. Although m-bit vectors can be embedded into the M-bit registers *without* knowing what m is in both embedding methods, we will show below that the new embedding method simplifies the circuits of the universal arithmetic operations.

The M-bit vector $X = (X_{M-1}, \ldots, X_1, X_0)$ stored in an M-bit register is interpreted as the field element

$$\hat{x} = X_{M-1}\alpha^{M-1} + \ldots + X_2\alpha^2 + X_1\alpha^1 + X_0\alpha^0 \qquad (1)$$

where again α is a root of $g(z)$, the primitive polynomial used to define the field $GF(2^m)$. Henceforth, $\hat{x}$ is called the register value of X. If $g(z)$ is of degree $m < M$, (1) is a redundant representation in the sense that many different X's correspond to the same $\hat{x} \in GF(2^m)$. To avoid this redundancy, when $m < M$, it is assumed that the $M - m$ lowest degree bits are always zero and that $X = (x_{m-1}, x_{m-2}, \ldots, x_2, x_1, x_0, 0, \ldots, 0)$ represents

$$\hat{x} = x_{m-1}\alpha^{M-1} + \ldots + x_2\alpha^{M-m+2} + x_1\alpha^{M-m+1} + x_0\alpha^{M-m}.$$

However, the m-bit vector $(x_{m-1}, \ldots, x_1, x_0)$ does *not* necessarily represent $\hat{x}$. In fact, the value represented by the m-bit vector, denoted here as x, depends on which basis the vector is interpreted with respect to. Note that $\hat{x}$ is the value used in the field multiplication, or in other words, the interpretation of the register X based on the operations of the universal multiplication. On the other hand, x is the human interpretation of the register X. Clearly, the vector $(x_{m-1}, \ldots, x_1, x_0)$ represents $\hat{x}$ with respect to a *shifted polynomial basis* (SPB) $\{\alpha^{M-m}, \ldots, \alpha^{M-2}, \alpha^{M-1}\}$. That is, x

and $\hat{x}$ collapse into each other in this case. On the other hand, $(x_{m-1}, \ldots, x_1, x_0)$ represents

$$x = x_{m-1}\alpha^{m-1} + x_{m-2}\alpha^{m-2} + \ldots + x_1\alpha^1 + x_0\alpha^0 \qquad (2)$$

with respect to the basis set $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$, referred to as the canonical polynomial basis (CPB) henceforth. In this case, $\hat{x} = \alpha^{M-m}x$. Hence, the relation between the register value and the field element represented by the m-bit vectors depends on the basis of representation. This basis dependency turns out to be important also for universal field multiplication. However, it will be shown below that the basis dependency does not affect the operations of the Reed-Solomon decoder.

3. UNIVERSAL FIELD ARITHMETICS

The arithmetic operations used in the universal decoder should be universal. This has two implications. First of all, all arithmetic operations need to work properly *and* obliviously to the size of the field. That is, all arithmetic operations work for any field $GF(2^m)$ where $m \leq M$, the size of the registers. The second implication of the universality is more subtle. Certain arithmetic operations in $GF(2^m)$, such as multiplication and division (inversion), depend on the primitive polynomial $g(z)$ which was used to define $GF(2^m)$. Normally, the circuits for these arithmetic operations are simplified utilizing the knowledge of $g(z)$. In other words, the circuits are hardwired according to $g(z)$ (see Figure 1). However, universal arithmetic operations must work for different choices of $g(z)$. Although the dependency of the aforementioned operations on $g(z)$ remains, it cannot be used to simplify or hardwire the circuits. Instead, $g(z)$ is part of the inputs for these operations, and these operations should operate in an oblivious manner to what $g(z)$ is.

3.1 Universal addition/subtraction

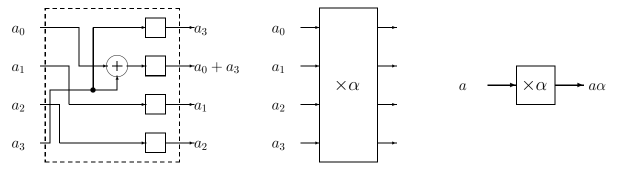
The addition of vectors X and Y can be implemented using XOR gates. Although it is not necessary to carry out the XOR operations in the lower order $M - m$ bits, doing so makes the addition universal and *oblivious* in the sense that the circuit does not need to know the size of the field. Note that the addition is correct regardless of which basis is used to interpret the operands and the result of the adder.

3.2 Universal α-multiplier

The universal α-multiplier is the building block of the universal multiplier. For any given $g(z)$, multiplication by α is very simple. For example, in $GF(2^4)$ generated by $g(z) = z^4 + z + 1$, multiplying $a = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$ by α results in

$$\begin{aligned} \alpha a &= \alpha(a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3) \\ &= a_3 + (a_0 + a_3)\alpha + a_1\alpha^2 + a_2\alpha^3 \end{aligned}$$

since $\alpha^4 = \alpha + 1$. The circuit for this α-multiplier is illustrated in Figure 1(a). In Figure 1(a), the bits a_0, a_1, and a_2 are shifted downward and a_3 is fed back and added onto the coefficients specified by the nonzero g_i's where $i < m$. Note that no AND gates are used in Figure 1(a). The block diagrams in Figure 1(b) shall be used to represent the α-multiplier when it is unnecessary to include the details. However, universal α-multiplier is slightly more complicated. Let $\hat{y} = \alpha\hat{x}$ where $\hat{x}$ and $\hat{y}$ are the register values



(a) example of an α-multiplier (b) block diagrams

Figure 1: The circuit for the α-multiplier for $GF(2^4)$ generated by $g(z) = z^4 + z + 1$ and the block diagrams used for α multiplier

of $X = (x_{m-1}, \ldots, x_1, x_0, 0, \ldots, 0)$ and
$Y = (y_{m-1}, \ldots, y_1, y_0, 0, \ldots, 0)$ respectively. It is clear that

$$\hat{y} = \alpha\hat{x} \;\; = \;\; x_{m-1}\alpha^M + \left(x_{m-2}\alpha^{M-1} + \ldots + x_0\alpha^{M-m+1}\right).$$

But since $g(\alpha) = \alpha^m + g_{m-1}\alpha^{m-1} + \ldots + g_0\alpha^0 = 0$, it follows that $\alpha^M = g_{m-1}\alpha^{M-1} + \ldots + g_0\alpha^{M-m}$. Hence,

$$\begin{aligned} \hat{y} &= \left(x_{m-2}\alpha^{M-1} + \ldots + x_1\alpha^{M-m+2} + x_0\alpha^{M-m+1}\right) \\ &+ \left(g_{m-1}\alpha^{M-1} + \ldots + g_1\alpha^{M-m+1} + g_0\alpha^{M-m}\right)x_{m-1}. \end{aligned}$$

That is, $y_i = x_{i-1} + x_{m-1}g_i$ for $i = 1, \ldots, m-1$ and $y_0 = x_{m-1}g_0$. Note that all the bits of X are shifted leftward and the highest-order bit x_{m-1} is fed back to the bits determined by the nonzero coefficients of $[g(z) - z^m]\,z^{M-m}$. Because the m high-order bits are used, the bit which should be fed back is always the highest-order bit and it should be fed back to the m highest order bits regardless of the value of m. Denote $[g(z) - z^m]\,z^{M-m}$ as $G(z)$. It follows that the coefficients vector

$$(G_{M-1}, \cdots, G_2, G_1, G_0) = (g_{m-1}, \cdots, g_2, g_1, g_0, 0, \cdots, 0)$$

determines which bit positions x_{M-1} should be fed back to. This implies that an α-multiplier can be devised to take coefficients of $g(z)$ as inputs and to operate obliviously to the actual values of the coefficients of $g(z)$. In fact, the α-multiplier does not need to know the value of m. Note that the way the coefficients vector $(g_{m-1}, \cdots, g_2, g_1, g_0)$ is embedded in the input coefficients vector $(G_{M-1}, \cdots, G_1, G_0)$ is consistent with how the field elements are represented. Also, note that the α-multiplier works regardless of the degree of $g(z)$ and what the coefficients of $g(z)$ are: the highest order bit is fed back and added into the positions where $G(z)$ has nonzero entries. The circuit of the universal α-multiplier is illustrated in Figure 2.

Note that by embedding the m-bit vector at the high order end, the bit that should be fed back is always at a fixed position regardless of the value of m. In contrast, the position of this bit depends on the value of m if the m-bit vector is embedded at the low order end and additional control registers and complementary switches (cf. Figure 10.1 of [9]) are required to find the bit that should be fed back. The new embedding method thus simplifies the universal α-multiplier, and hence the universal multiplier uses less hardware than the universal MSR multiplier in [9].

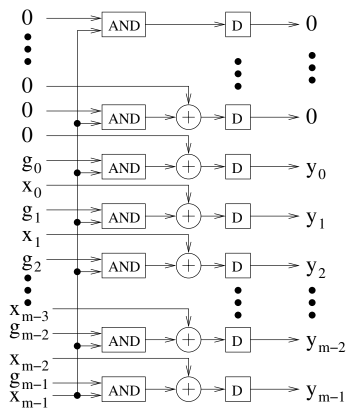


Figure 2: The circuit for universal α-multipliers

As a final comment, note that the universal α-multiplier multiplies the content of the X register by α regardless of the basis with respect to which the content is interpreted to.

3.3 Universal multipliers

There are many possible ways to implement the multiplication of two arbitrary field elements. Two approaches based on the CPB-based representation are considered here because circuitry based on these two approaches can be easily pipelined and readily adapted to universal multiplication. These two approaches are essentially the *modified shift-register* (MSR) and *standard shift-register* (SSR) multipliers in [9]. The basic idea of the MSR multiplier [10, 9] is the following:

$$\begin{aligned} ab &= a(b_0 + b_1\alpha + b_2\alpha^2 + \cdots b_{m-1}\alpha^{m-1}) \\ &= ab_0 + (a\alpha)b_1 + \cdots + ((\cdots((a\alpha)\alpha)\cdots\alpha)b_{m-1}. \end{aligned}$$

That is, the sum can be computed using an m-stage approach. The i-th ($1 \le i \le m$) stage has inputs $a\alpha^{i-1}$, b_{i-1}, and a partial sum $\sum_{j=0}^{i-2}(a\alpha^j)b_j$. The i-th stage adds $a\alpha^{i-1}$ or 0 to the partial sum depending on whether b_{i-1} is 1 or 0, and computes $a\alpha^i$ by multiplying $a\alpha^{i-1}$ by α using an α-multiplier such as the one shown in Figure 1. Then, $a\alpha^i$ and a new partial sum $\sum_{j=0}^{i-1}(a\alpha^j)b_j$ are passed on to the next stage. Note that the accumulator is initially 0, and equals ab after the m-th stage.

Alternatively, the product ab can be computed via Horner's rule (called the SSR multiplier in [9]):

$$\begin{aligned} ab &= a(b_0 + b_1\alpha + b_2\alpha^2 + \cdots + b_{m-1}\alpha^{m-1}) \\ &= (\cdots((ab_{m-1})\,\alpha + ab_{m-2})\,\alpha \cdots + ab_1)\,\alpha + ab_0. \end{aligned}$$

This is also an m-stage approach except that this approach accounts for the high degree bits of b first. The i-th ($1 \le i \le m$) stage has inputs a, b_{m-i}, and a partial sum

$$(\cdots((ab_{m-1})\,\alpha + ab_{m-2})\,\alpha \cdots + ab_{m-i+2})\,\alpha + ab_{m-i+1}.$$

The i-th stage multiplies the partial sum by α using an α-multiplier and adds ab_{m-i} to the partial sum. Then, a and the new partial sum

$$(\cdots(ab_{m-1})\,\alpha + \cdots + ab_{m-i+1})\,\alpha + ab_{m-i}$$

are passed on to the next stage.

The MSR and SSR multipliers both have m-stage structures and hence can be pipelined easily into m-stages. The structure for a four-stage pipelined MSR-based multiplier in $GF(2^4)$ is illustrated in Figure 3. The pipelined multiplier consists of four systolic processing units, referred to as *ppm* unit. The circuit of the *ppm* unit is shown in Figure 4. Here $g(z)$ is assumed to be given. The SSR multipliers can be pipelined similarly, and details are omitted due to space limitation. The pipelined MSR and SSR multipliers described

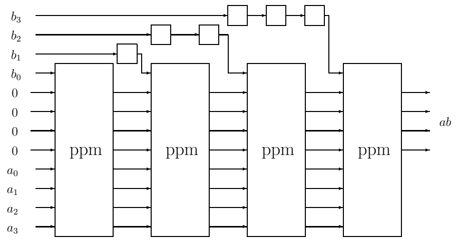
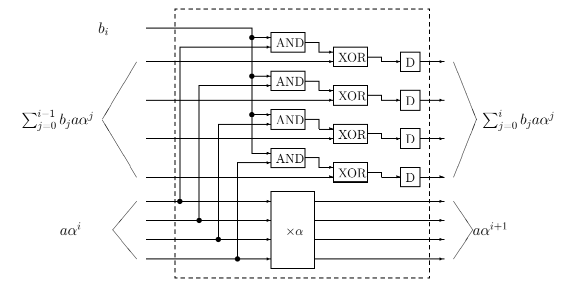


Figure 3: The structure for the pipelined MSR multiplier for $GF(2^4)$, which consists of four systolic processing units



b_i
$\sum_{j=0}^{i-1} b_j a\alpha^j$
AND
XOR
D
$\sum_{j=0}^{i} b_j a\alpha^j$
$a\alpha^i$
$\times\alpha$
$a\alpha^{i+1}$

Figure 4: The detailed circuit for the $(i+1)$-th *ppm* unit

above can be adapted to universal multipliers with just a few changes. Depending on the interpretation of the inputs and outputs, the result of the universal multipliers can be correct or can include a scaling factor. (It is shown later that this scaling factor does not affect the operations of the Reed-Solomon decoders.) Note that both universal multipliers depends on $g(z)$ only through the α-multipliers. Since the latter works obliviously to the value of $m \le M$ and the coefficients of $g(z)$, so do the universal multipliers. Furthermore, when the universal multipliers are used in pipelined mode, each of the M stages could potentially use a different $g(z)$ as long as the $g(z)$'s move along synchronously with the elements being multiplied.

3.3.1 Universal pipelined MSR multiplier

The structure of the universal pipelined MSR multiplier based on the universal α-multiplier is shown in Figure 5. The universal pipelined MSR multiplier consists of M systolic processing units, referred to as *uppm* units. The circuit of the *uppm* units is the same as the *ppm* unit shown in Figure 4 except that all the inputs and outputs are M-bit vectors and the α-multipliers used in the *ppm* units are replaced by the universal α-multiplier of Figure 2. Note that the coefficients of $g(z)$ are part of the inputs to each universal α-multiplier. Hence, although not shown explicitly in Figure 5, the coefficients of $g(z)$ are also part of the inputs to the universal multiplier, which uses the universal α-multipliers as building blocks.

The inputs to the multiplier are

$$A = (A_{M-1}, A_{M-2}, \ldots, A_1, A_0)$$

and

$$B = (B_{M-1}, B_{M-2}, \ldots, B_1, B_0)$$

whose register values are

$$\hat{a} = A_{M-1}\alpha^{M-1} + A_{M-2}\alpha^{M-2} + \ldots + A_1\alpha^1 + A_0\alpha^0$$

and

$$\hat{b} = B_{M-1}\alpha^{M-1} + B_{M-2}\alpha^{M-2} + \ldots + B_1\alpha^1 + B_0\alpha^0$$

respectively. The universal MSR multiplier produces

$$\begin{aligned}\hat{a}\hat{b} &= \hat{a}(B_0\alpha^0 + B_1\alpha^1 + \ldots + B_{M-2}\alpha^{M-2} + B_{M-1}\alpha^{M-1}) \\ &= \hat{a}B_0 + (\hat{a}\alpha)B_1 + \cdots + ((\cdots((\hat{a}\alpha)\alpha)\cdots\alpha)B_{M-1}.\end{aligned}$$

That is, the universal MSR multiplier produces the product $\hat{a}\hat{b}$ of the register values $\hat{a}$ and $\hat{b}$. But, when $m < M$,

$$A = (a_{m-1}, a_{m-2}, \ldots, a_1, a_0, 0, \ldots, 0)$$

and

$$B = (b_{m-1}, b_{m-2}, \ldots, b_1, b_0, 0, \ldots, 0)$$

can be interpreted either as $\hat{a}$ and $\hat{b}$ with respect to the SPB as in (1) or as a and b with respect to the CPB as in (2). The multiplier computes

$$\hat{a} \cdot \hat{b} = \left(\alpha^{M-m}a\right)\left(\alpha^{M-m}b\right) = \alpha^{M-m}\left[\alpha^{M-m}(ab)\right].$$

Since the product ab would be stored in the register as $\left[\alpha^{M-m}(ab)\right]$, the result of the computation is correct with respect to the SPB but has an extra scaling factor α^{M-m} with respect to the CPB.

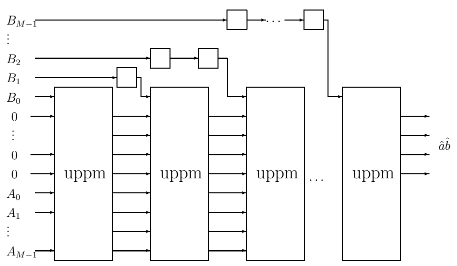


Figure 5: The circuit for the universal MSR pipelined multiplier

3.3.2 *Universal pipelined SSR multiplier*

A universal pipelined SSR multiplier can be designed based on the pipelined SSR multiplier architecture in a similar fashion, the above observation about the results of the universal MSR multiplier is also applicable to the universal SSR multiplier. The details are omitted.

4. UNIVERSAL BERLEKAMP-MASSEY ALGORITHM

Hardware implementation of a Reed-Solomon decoder usually consists of three blocks operating in a pipelined fashion: Syndrome Computation (SC) block, Key Equation Solver (KES) block, and Chien Search and Error Evaluation (CSEE) block[2]. In order to design a universal RS decoder, all three blocks have to be studied from the viewpoint of universality. Similar to [10], we focus on only the key equation solver in this paper since the design of the other two blocks of the RS decoder, syndrome computer and error corrector, are relatively easy and hence omitted in this paper. Sarwate and Shanbhag proposed a reformulated Berlekamp-Massey algorithm and designed high-speed RS decoder architectures based on the reformulated algorithm [10]. We show here that the algorithm proposed by Sarwate and Shanbhag can be adapted to universal decoding of RS codes by simply accounting for the potential scaling factors by the universal multipliers. Thus, the SS architectures can be adapted to universal decoder architectures by simply replacing the finite field arithmetics with universal arithmetics discussed above.

The main issue with the replacement is that, as described in the previous section, there is possibly a scaling factor associated with the universal multiplier. Consider the following algorithm where we associate each Galois field multiplication with a scaling factor C:

Universal Berlekamp-Massey Algorithm

1. **Initialization:**
   $\hat{\Delta}^{(0)}(z) = \hat{\Theta}^{(0)}(z) = S(z); \hat{\Lambda}^{(0)}(z) = \hat{B}^{(0)}(z) = 1;\ \hat{\gamma}^{(0)} = 1.$

2. **for** $r = 0$ **until** $2t-1$ **do**

$$\hat{\Delta}^{(r+1)}(z) = \left\lfloor \frac{C\hat{\gamma}^{(r)}\hat{\Delta}^{(r)}(z)}{z} \right\rfloor - C\hat{\Delta}_0^{(r)}\hat{\Theta}^{(r)}(z) \quad (3)$$

$$\hat{\Lambda}^{(r+1)}(z) = C\hat{\gamma}^{(r)}\hat{\Lambda}^{(r)}(z) - Cz\hat{\Delta}_0^{(r)}\hat{B}^{(r)}(z) \quad (4)$$

$$\hat{\Theta}^{(r+1)}(z) = \begin{cases} \left\lfloor \frac{\hat{\Delta}^{(r)}(z)}{z} \right\rfloor \\ \hat{\Theta}^{(r)}(z) \end{cases} \quad (5)$$

$$\hat{B}^{(r+1)}(z) = \begin{cases} \hat{\Lambda}^{(r)}(z) \\ z\hat{B}^{(r)}(z) \end{cases} \quad (6)$$

$$\hat{\gamma}^{(r+1)} = \begin{cases} \hat{\Delta}_0^{(r)} \\ \hat{\gamma}^{(r)} \end{cases} \quad (7)$$

3. **Output:**
   $\hat{\Lambda}(z) = \hat{\Lambda}^{(2t)}(z); \quad \hat{\Omega}_h(z) = \hat{\Delta}^{(2t)}(z).$

In (5), $(\Theta^{(r)}(z), zB^{(r)}(z), \gamma^{(r)})$ is updated in two different ways depending on the values of certain parameters, but the details are not important for our discussion. Note that C takes into account the possible scaling factor associated with the universal field multipliers discussed in the previous section. We can show that

Theorem 1. *There exists a nonzero factor c such that $\hat{\Lambda}(z) = c\Lambda(z)$ and $\hat{\Omega}_h(z) = c\Omega_h(z)$.*

[2]We omit the background introduction of the decoding of RS codes and the structure of RS decoders. Interested readers are referred to [1, 10].

The above theorem shows that the outputs of the universal Berlekamp-Massey algorithm, which accounts for the possible scaling factor of the universal field multipliers, are multiples of $\Lambda(z)$ and $\Omega_h(z)$ respectively. Since $c\Lambda(z)$ $(c \neq 0)$ has the same roots as $\Lambda(z)$, the error locations, which are given by the roots of $\Lambda(z)$, can thus be obtained from $c\Lambda(z)$. Furthermore, the scaling factors cancel out in Forney's formula, used to evaluate error values, and thus correct error values can be obtained from the universal Berlekamp-Massey algorithm. This shows that the possible scaling factor associated with the universal field multipliers is inconsequential.

5. PERFORMANCES

As shown above, the high-throughput architectures for RS decoders in [10] are adapted to universal architectures by simply replacing the finite field arithmetics with universal field arithmetics discussed above. Note that the SS architectures can achieve very high throughputs, especially when the RS codewords are interleaved. It is shown that the critical path delay (CPD) of the SS decoder architectures is *two* logic gates when the codewords are interleaved. Since our universal decoder architectures are obtained by using universal field arithmetics in the SS architectures, it can be shown that our universal decoder architectures achieve high throughputs as well. When the codewords are interleaved, the critical path delay of our universal decoder architecture is *three* logic gates. It appears that the penalty on the CPD for the flexibility in the parameters is *only one* logic gate.

The hardware requirements of our universal RS decoder architectures are exactly the same as those of a decoder architecture that works for all possible primitive polynomials of $GF(2^M)$. The only differences between our universal architecture and an SS architecture for a fixed primitive polynomial of $GF(2^M)$ is that g_i's can be hard-wired in the latter. Due to the modular and regular structures of the universal multipliers and adders, the unnecessary parts may be turned off to save power consumption when a RS code defined on $GF(2^m)$ $(m < M)$ is being decoded. Overall, we conjecture that the penalty on hardware requirements and power consumption for the universality is small.

We remark that the universal RS decoders proposed in this paper can be adapted to decode BCH codes, as well as extended and shortened RS codes with flexible code parameters [2].

6. REFERENCES

[1] R. E. Blahut, *Theory and Practice of Error-Control Codes*, Addison-Wesley, Reading MA, 1983.

[2] R. E. Blahut, "A universal Reed-Solomon decoder," *IBM Journal of Research and Developments*, vol. 28, no. 2, pp. 150–158, 1984.

[3] A. J. Goldsmith and P. P. Varaiya, "Capacity of fading channels with channel side information," *IEEE Transactions on Information Theory*, vol. IT-43, pp. 1986–1992, November 1997.

[4] M. A. Hasan and M. Ebtedaei, "Efficient architectures for computations over variable dimensional Galois fields," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 45, pp. 1205–1211, 1998.

[5] B. Green and G. Drolet, "A universal Reed-Solomon decoder chip," in *Proceedings of the 16th Biennial Symposium on Communications*, 1992, pp. 327–330.

[6] M. A. Hasan and A. G. Wassal, "VLSI algorithms, architectures, and implementation of a versatile $GF(2^m)$ processor," *IEEE Transactions on Computers*, vol. 49, no. 10, pp. 1064–1073, October 2000.

[7] J.-C. Huang, C.-M. Wu, M.-D. Shieh, and C.-H. Wu, "An area-efficient versatile Reed-Solomon decoder for ADSL," in *Proceedings of the IEEE International Symposium on Circuits and Systems 1999*, pp. 517–520.

[8] A. Y. Kwentus et al., "A single-chip universal digital satellite reciever with 480-MHz IF input," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 1634–1646, November 1999.

[9] E. D. Mastrovito, *VLSI Architectures for Computations in Galois Fields*, Ph.D. thesis, Linköping University, 1991.

[10] D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for Reed-Solomon decoders," *IEEE Transactions on VLSI Systems*, vol. 9, no. 5, pp. 941–955, October 2001.

[11] Y. R. Shayan, T. Le-Ngoc, and V. K. Bhargava, "A versatile time-domain Reed-Solomon decoder," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 8, pp. 1535–1542, 1990.

[12] Y. R. Shayan and T. Le-Ngoc, "A cellular structure for a versatile Reed-Solomon decoder," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 80–85, 1997.

[13] Y. R. Shayan and T. Le-Ngoc, "Decoding Reed-Solomon codes generated by any generator polynomial," *Electronics Letters*, vol. 25, no. 18, pp. 1223–1224, 1989.

[14] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*, IEEE Press, New York, 1994.

[15] C. S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields $GF(2^m)$," *IEEE Transactions on Computers*, vol. 33, pp. 357–360, April 1984.