The International Journal Of Engineering And Science (IJES)
||Volume||2 ||Issue|| 9||Pages|| 31-40||2013||
ISSN(e): 2319 – 1813 ISSN(p): 2319 – 1805

Implementation of Unified Hybrid Reed – Solomon Decoder

[1,] Farheen Sultana , [2,]M.Madhuri Latha , [3,]Ch.Ganapathy Reddy
[1,] Masters Scholar DECE , [2,]Asst.Professor [3,]Professor
[1,2,3,]Department of ECE GNITS,JNTU

--ABSTRACT--

Reed-Solomon (RS) codes have been widely employed for error correction in modern digital communication and data storage systems. Currently, for decoding RS codes with random-error correction, numerous literatures have given extensive studies on theoretical algorithms as well as hardware implementations [4], [5], [8], [9]. However, for specific RS burst-error decoder design, although some dedicated algorithms had been reported [3], [7], the VLSI implementations for these burst-error correcting algorithms are still under-investigated which are limited by their cubic computation complexity. In this brief, starting from a recent theoretical work, a low-complexity reformulated inversionless burst-error correcting (RiBC) algorithm is developed for practical applications. Then, based on the algorithm, a unified VLSI architecture that is capable of correcting burst errors, as well as random errors and erasures, is firstly presented for multi-mode decoding requirements. This new architecture is denoted as unified hybrid decoding (UHD) architecture.

KEYWORDS: *Burst errors, RS codes, Unified Hybrid Architecture.*

Date of Submission: 16, September, 2013 ⟺ Date of Acceptance: 30, September 2013

I. INTRODUCTION

Digital communication system is used to transport an information-bearing signal from the source to a user destination via a communication channel. The information signal is processed in a digital communication system to form discrete messages, which makes the information more reliable for transmission. Channel coding is an important signal processing operation for the efficient transmission of digital information over the channel. In channel coding the number of symbols in the source encoded message is increased in a controlled manner in order to facilitate two basic objectives at the receiver: error detection and error correction.

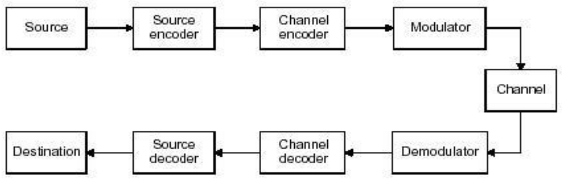


Fig. 1 Digital Communication System

Reed Solomon Codes are the non- binary form of the Block codes. These codes are defined as RS (n, k) with m- bit symbols. The RS code encoder_LFSR takes k data symbols and, parity symbols are added to it to form n symbols codeword. The RS codes detect and correct the errors in the symbols. These codes are constructed with encoding and decoding techniques, which follows directly to the BCH code techniques. The major difference between the binary BCH and non- binary RS codes is, in BCH code the error magnitude is 1 so it needs to find out only the location of error while in non binary RS codes the first step is to locate the error position and then correct it. Reed Solomon codes are also follows the GF mathematics properties for encoding and decoding techniques. Reed-Solomon codes are non-binary cyclic codes with symbols made up of m-bit sequences, where m is any positive integer having a value greater than 2. R-S (n, k) codes on m-bit symbols exist for all n and k for which

$$0 < k < n < 2^m + 2 \qquad (1.1)$$

For the most conventional R-S (n, k) code,

$$(n, k) = (2^m - 1, 2^m - 1 - 2t) \qquad (1.2)$$

where t is the symbol-error correcting capability of the code, and $n - k = 2t$ is the number of parity symbols.

II. REED – SOLOMON CODES

A brief overview of the encoding and decoding of Reed – Solomon codes is given in this section.

2.1 Encoding of Reed – Solomon Codes

Let $(d_{k-1}, d_{k-2}, \ldots, d_1, d_0)$ denote k m - bit data symbols (bytes) that are to be transmitted over a communication channel (or stored in memory). These bytes are regarded as elements of the finite field (also called Galois field) $GF(2^m)$, and encoded into a codeword $(c_{n-1}, c_{n-2}, c_{n-3} \ldots, c_1, c_0)$ of $n > k$ bytes. These codeword symbols are transmitted over the communication channel (or stored in memory). For Reed–Solomon codes over $GF(2^m)$, $n = 2^m - 1$, k is odd, and the code can correct $t = \frac{n-k}{2}$ byte errors. The encoding process is best described in terms of the data polynomial $D(x) = d_{k-1}x^{k-1} + d_{k-2}x^{k-2} + \ldots \ldots d_1 x^1 + d_0 x^0$ being transformed into a codeword polynomial $C(x) = c_{n-1}x^{n-1} + c_{n-1}x^{n-1} + \ldots \ldots + c_1 x^1 + c_0 x^0$. All codeword polynomials $C(x)$ are polynomial multiples of $G(x)$, the generator polynomial of the code, which is defined as

$$G(x) = \prod_{i=0}^{2t-1} (x - \alpha^{(m_0+i)}) \qquad (2.1)$$

where m_0 is typically zero or one. Since $2t$ consecutive powers $\alpha^{m_0}, \alpha^{m_0+1} \ldots . \alpha^{m_0+2t-1}$ of α are roots of $G(x)$,and $C(x)$ is a multiple of $G(x)$, it follows that

$$C(\alpha^{m_0+i}) = 0 \qquad 0 \le i \le 2t\text{-}1 \qquad (2.2)$$

for all codeword polynomials $C(x)$ .Systematic encoding produces codeword's that are comprised of data symbols followed by parity-check symbols and are obtained as follows: Let $Q(x)$ and $P(x)$ denote the quotient and remainder respectively when the polynomial $x^{n-k}D(x)$ of degree n - 1 is divided $G(x)$ by of degree $2t = n - k$ .Thus, $x^{n-k}D(x) = Q(x)G(x) + P(x)$ where $\deg P(x) < n - k$.
Clearly $Q(x)G(x) = x^{n-k}D(x) - P(x) = C(x)$, is a multiple of $G(x)$. Furthermore, since the lowest degree term in $x^{n-k}D(x)$ is $d_0 x^{n-k}$ while $P(x)$ is of degree at most $n - k$ - 1, it follows that the codeword is given by $(c_{n-1}, c_{n-2}, \ldots \ldots c_1, c_0) = (d_{k-1}, \ldots \ldots d_1, d_0, -p_{n-k-1}, -p_{n-k-2}, \ldots \ldots - p_1, -p_0)$ and consists of the data symbols followed by the parity-check symbols.

2.2 Reed – Solomon Decoding

RS decoding is done in three levels. First one being, syndrome calculation that tells us whether an error has occurred during the transmission of data. The second step includes error location, which tells us where the error is present, and the third one is the error evaluation, which corrects the error. However, the decoder has the capability of correcting t errors where $n=k+2t$.

Let $C(x)$ denote the transmitted codeword polynomial and let $R(x)$ denote the received word polynomial. The input to the decoder is $R(x)$, and it assumes that

$$R(x) = C(x) + E(x) \qquad (2.3)$$

where, if $e > 0$ errors have occurred during transmission, the error polynomial $E(x)$ can be written as

$$E(x) = Y_1 x^{i1} + Y_2 x^{i2} + Y_3 x^{i3} + \ldots \ldots . Y_e \; (2.4)$$

It is conventional to say that the error values $Y_1, Y_2, Y_3 \ldots Y_e$ have occurred at the error locations $X_1 = \alpha^{i1}, X_2 = \alpha^{i2}, \ldots \ldots . X_e = \alpha^{ie}$. Note that the decoder does not know $E(x)$; in fact, it does not even know the value of e . The decoder's task is to determine $E(x)$ from its input $R(x)$, and thus correct the errors by subtracting off $E(x)$ from $R(x)$. If $e \le t$, then such a calculation is always possible, that is, t or fewer errors can always be corrected. The decoder begins its task of error correction by computing the syndrome values

$$s_i = R(\alpha^{m_0+i}) = C(\alpha^{m_0+i}) + E(\alpha^{m_0+i}) = E(\alpha^{m_0+i}) \qquad 0 \le i \le 2t - 1 \quad (2.5)$$

If all $2t$ syndrome values are zero, then $R(x)$ is a codeword and it is assumed that $C(x) = R(x)$, that is, no errors have occurred. Otherwise, the decoder knows that $e > 0$ and uses the syndrome polynomial $S(x)$ which is defined to be

$$S(x) = s_0 + s_1 x + \ldots \ldots \ldots \ldots s_{2t-1} x^{2t-1} \qquad (2.6)$$

to calculate the error values and error locations. Define the error locator polynomial $\Lambda(x)$ of degree e and the error evaluator polynomial $\Omega(x)$ of degree at most e - 1 to be

$$\Lambda(x) = \prod_{j=1}^{e}(1 - X_j x) = 1 + \lambda_1 x + \ldots\ldots + \lambda_e x^e \quad (2.7)$$

$$\Omega(x) = \sum_{i=1}^{e} Y_i X_i^{m_0} \prod_{j=1, j\neq i}^{e}(1 - X_j x) = \omega_0 + \omega_1 x + \omega_2 x^2 + \ldots \omega_{e-1} x^{e-1} \quad (2.8)$$

These polynomials are related to through the *key equation* [1], [2]

$$\Lambda(x)S(x) = \Omega(x) \bmod x^{2t} \quad (2.9)$$

Solving the key equation to determine both $\Lambda(x)$ and $\Omega(x)$ from $S(x)$ is the hardest part of the decoding process. The BM algorithm and the eE algorithm can be used to solve (4.14). If $e \leq t$, these algorithms find $\Lambda(x)$ and $\Omega(x)$, but if , then the algorithms almost always fail to find $\Lambda(x)$ and $\Omega(x)$. Fortunately, such failures are usually easily detected. Once $\Lambda(x)$ and $\Omega(x)$ have been found, the decoder can find the error locations by checking whether $\Lambda(\alpha^{-j}) = 0$ for each *j*, $0 \leq j \leq n-1$. Usually, the decoder computes the value of just $\Lambda(\alpha^{-j})$ before the *j*-th received symbol r_j leaves the decoder circuit. This process is called a Chien search [1], [2]. if $\Lambda(\alpha^{-j}) = 0$, then α^{-j} is one of the error locations (say X_i). In other words, r_j is in error, and needs to be corrected before it leaves the decoder. The decoder can calculate the error value Y_i to be subtracted from r_j via Forney's error value formula [2]

$$Y_i = \frac{x^{m_0}\Omega(x)}{x\Lambda'(x)}\Big|_{x=\alpha^{-j}} \quad (2.10)$$

where $\Lambda'(x) = \lambda_1 + 2\lambda_2 x + 3\lambda_3 x^2 + \ldots..$ denotes the formal derivative of $\Lambda(x)$. Note that the formal derivative simplifies $\Lambda'(x) = \lambda_1 + \lambda_3 x^2 + \ldots$ to since we are considering codes over $GF(2^m)$. Thus, $x\Lambda'(x) = \lambda_1 x + \lambda_3 x^3 + \ldots$, which is just the terms of odd degree in $\Lambda(x)$. Hence, the value of $x\Lambda'(x)$ at $x = \alpha^{-j}$ can be found during the evaluation of $\Lambda(x)$ at $x = \alpha^{-j}$ and does not require a separate computation.

2.3 RS Decoder Structure

In summary, a Reed–Solomon decoder structure consists of three blocks:

• The Syndrome Computation (SC) block;
• The Key Equation Solver (KES) block;
• The Chien Search and Error Evaluator (CSEE) block.

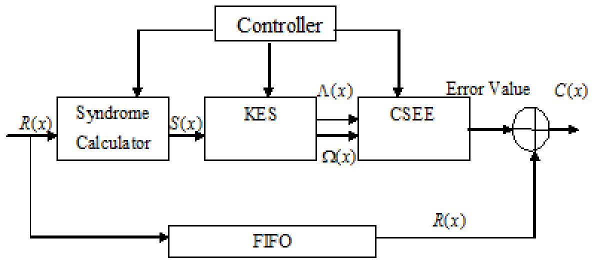


Fig. 2 Decoder Structure

1) **SC-Block**: At the end of received word, all cells store the syndrome values and Syndrome Computation (SC) block sets its flag high if one or more syndrome values are not zero.
2) **KES-Block*:*** The Key equation solver (KES) block provides two polynomial -error locator polynomial and error magnitude polynomial.
3) **CSEE-Block**: Chien Search and Error Evaluator (CSEE) block identifies error location while computes its error magnitude.
4) **FIFO register:** FIFO register stores 32 received word symbols. To match the order of the bytes in error vector and received codeword FIFO is applied as the error vector is produced in the reverse order of the received codeword.
5) **Controller***:* The controller provides synchronization among all four modules -SC, KES, CSEE and FIFO Registers.

III. DECODING ALGORITHMS

3.1 The Berlekamp – Massey Algorithm

The Berlekamp and Massey Algorithm is a iterative procedure for finding the error locator polynomial

$$\Lambda(x) = \prod_{j=1}^{e}(1 - X_j x) = 1 + \lambda_1 x + \lambda_2 x^2 + \ldots + \lambda_e x^e \quad (3.1)$$

After calculating the syndromes, there exist a relation between syndrome and coefficient of error locator polynomial which is called as the Newton's identities.

$$\begin{aligned} &S_1 + \lambda_1 = 0 \\ &S_2 + \lambda_1 S_1 + 2\lambda_2 = 0 \qquad (3.2) \\ &\vdots \\ &S_{v+1} + \lambda_1 S_v + \ldots + \lambda_v S_1 = 0 \end{aligned}$$

The first step of iteration is to find a minimum degree polynomial $\Lambda^{(1)}(x)$ whose coefficients satisfies the first Newton's identity of equation (3.2),the next step is to test whether the coefficient of $\Lambda^{(1)}(x)$ also satisfy the second Newton's identity of (3.2). If the coefficients do satisfy the second Newton's identity then set $\Lambda^{(2)}(x) = \Lambda^{(1)}(x)$. If the coefficients do not satisfy the second identity, add a correction term to $\Lambda^{(1)}(x)$ to form $\Lambda^{(2)}(x)$ such that $\Lambda^{(2)}(x)$ has minimum degree and its coefficients satisfy the first two identities of (3.2).Therefore, at the end of the second step of iteration, a minimum degree polynomial $\Lambda^{(2)}(x)$ is obtained whose coefficients satisfies first two Newton's identity. Iterations continues until $\Lambda^{(2t)}(x)$ is obtained then $\Lambda(x) = \Lambda^{(2t)}(x)$. At the *r*-th step after finding $\Lambda^{(r)}(x)$ in order to compute $\Lambda^{(r+1)}(x)$ we calculate discrepancy which is $\delta^{(r)} = \sum \Lambda_i^r S_{r-i}$ .if $\delta^{(r)} = 0$ the coefficients of $\Lambda^{(r)}(x)$ satisfies $(r+1)$ th identity, so that $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x)$. If $\delta^{(r)} \neq 0$ add a correction factor to $\Lambda^{(r)}(x)$ to obtain $\Lambda^{(r+1)}(x)$.

3.2 Random Error and Erasure Correcting Algorithm

A conventional error alone RS decoding procedure can be modified to correct both error and erasure [2] (an erasure occurs when position of the corrupted symbol is known). The (n, k) RS code that has minimum distance $d = n - k + 1$ when used on the channel that makes both errors and erasures can be corrected provided that $d \geq 2v + \rho + 1$. If $C(x)$ is a codeword polynomial and $R(x)$ is received polynomial then

$$R(x) = C(x) + \tilde{E}(x) \qquad (3.3)$$

Where $\tilde{E}(x) = E(x) + F(x)$ is called errata polynomial which is the sum of error polynomial and erasure polynomial.

Let α be the primitive element in $GF(2^m)$ we say errata magnitude $\tilde{Y}_l = \tilde{e}_{i_l}$ for $l = 1, \ldots\ldots, v + \rho$, occurred at errata locations $Z_l = \alpha^{i_l}$ for $l = 1, \ldots, v + \rho$. Note that provided with $d \geq 2v + \rho + 1$ when $\rho = d - 1$ or $\rho = d - 2$ no errors occur and hence errata polynomial is same as errata locator polynomial $\Lambda(x)$. The authors of [4] proposed a novel algorithm transformation to reformulate the errors-alone correcting BM algorithm to significantly reduce the critical path, where the basic idea is to remove the data dependency inside of one iteration at the cost of increased computation complexity so that the critical path can be reduced. In this section, we apply the same methodology to reformulate the above BM algorithm for errors-and-erasures RS decoding.

3.3 Correcting Algorithm single Long Burst error

A novel low complexity RS burst - error correcting algorithm that only requires *O*(2*nt*) computations, where *n* and *t* are length of the code and traditional correction capability respectively. Wu in [6] proposed a new approach to track the position of burst of errors. By introducing a new polynomial that is a special linear function of syndromes, the desired single burst of errors can be acquired by tracking the longest consecutive roots of new polynomial.The errors to be a single burst of length *f* if their locations are within the interval of *f*. Note that it is not necessary that all symbols in the burst interval are erroneous, that is a burst might be inconsecutive. In this a new polynomial $\Gamma(x)$ is defined which is a special linear function is syndromes given by

$$\Gamma(x) \square \sum_{i=0}^{r-1} S_{r-1-i} \overline{\Lambda}_i^{r-1} x^i \qquad (3.4)$$

Where $\overline{\Lambda}_i^{\,r-1}$ are the coefficients of $\overline{\Lambda}^{(p)}(x) = (1-\alpha^{1-p}x)(1-\alpha^{2-p}x)....(1-\alpha^{-1}x)(1-x)$ A burst is started at position α^{-s} with the length *f*, if and only if $\Gamma(x)$ has consecutive roots $\alpha^{s+f-1}, \alpha^{s+f-1}, ...\alpha^{s+r-2}$ .These can correct up to burst of length *f* maximum of $r-1$ where $t \square \frac{r}{2}$. Note that $\Gamma(x)$ has degree $r-1$ and thus may have up to $r-1$ valid roots, that is there are $r-1$ candidate single bursts in the worst case, hence objective is to determine the shortest one. The next step is to compute the error magnitudes through Forney formula, which is

$$Y_i = \frac{\Omega(x)}{\Lambda'(x)}\Big|_{x=X^{-1}} \tag{3.5}$$

Where X_i^{-1} denotes the root of $\Lambda(x)$ and $\Lambda'(x)$ denotes the derivative of $\Lambda(x)$. Denote s by the beginning position of the burst, then the genuine error locator polynomial can be computed by

$$\Lambda(x) = \overline{\Lambda}^{(r-1)}(\alpha^{s+r-2}x) \tag{3.6}$$

And the error evaluator polynomial is the computed through

$$\Omega(x) = \Lambda(x)S(x) \bmod x^r \tag{3.7}$$

3.4 Reformulated inversionless Burst error Correcting Algorithm

The single long burst error correcting algorithm is extended to BC algorithm for correcting a long burst of error with length up to $2t-1-2\beta$ plus a maximum of β random errors. In the BC algorithm, β is a pre-chosen parameter that determines the specific error correcting capability. It indicates that the decoder is capable of correcting a *f*-length $(f < 2t - 2\beta)$ burst of errors plus a maximum of β random errors. The BC Algorithm is reformulated to a proposed RiBC algorithm for the decoding of Reed Solomon codes to achieve efficient VLSI design and overcome its disadvantages. That is long data path and data dependencies are removed using intermediate polynomials also there is simultaneous computation of discrepancies and updates all the polynomial. Further there is no need of extra 2*t* cycles or another copy of original circuitry. Finally the inversion operation that exists in error locating polynomial coefficients is removed.The RiBC algorithm is a kind of list decoding algorithm. Eight polynomials are updated simultaneously in each iteration. After every 2β inner iterations, $\tilde{\Lambda}^{(2\beta)}(x)$ as the candidate of the error locator polynomial of the random errors, is computed for current *l* th outer iteration. When *l* reaches *n* , we track the $\tilde{\Lambda}^{(2\beta)}(x)$ that is identical for longest consecutive *l* , and record the last element of l^* the consecutive *l*'s. Then the corresponding $\Lambda^{(2\beta)}(x)$ and $\tilde{\Delta}^{(2\beta)}(x)$ at the l^* th loop are marked as overall error locator polynomial $\Lambda^*(x)$ and error evaluator polynomial $\Omega^*(x)$ respectively. Finally Forney algorithm is used to calculate the error value in each error position with the miscorrection probability up to $(n-2f)(n-f)^{\beta}2^{m(\beta+f-2t)}$.

```
A. Reformulated inversionless Burst Error Correcting Algorithm
$f$-length ($f < 2t - 2\beta$) burst of error plus maximum $\beta$ of random errors:
Input: Syndromes $S_0, S_{1,} S_2, ......, S_{2t-1}$;
A1: Compute $\Xi(x) = (1-\alpha^{1-(2t-2\beta)}x)(1-\alpha^{2-(2t-2\beta)}x)......(1-\alpha^{-1}x)(1-x)$
        $= 1 + \xi_1 x + \xi_2 x^2 + ...... + \xi_{2t-2\beta} x^{2t-2\beta}$;
A2: For $l = 0$ Step 1 Until $n-1$ do
    A2.1: Compute $\Phi(x) = \Xi(\alpha^l x) = 1 + \phi_1 x + \phi_2 x^2 + ......\phi_{2t-2\beta} x^{2t-2\beta}$;
    A2.2: Compute $\Psi(x) = \psi_0 + \psi_1 x + \psi_2 x^2 + .. + \psi_{2t-1} x^{2t-1}$, where $\psi_i = \sum_{j=0}^{2t-2\beta} \phi_j S_{i+2t-2\beta-j}$;
    A2.3: Intialize $\Lambda^{(0)}(x) = \lambda_0^{(0)} + \lambda_1^{(0)} x + ..... + \lambda_{2t-2}^{(0)} x^{2t-2} = \Phi(x)$;
            $B^{(0)}(x) = b_0^{(0)} + b_1^{(0)} x + ....... + b_{2t-2}^{(0)} x^{2t-2} = \Phi(x)$;
            $\tilde{\Lambda}^{(0)}(x) = \tilde{\lambda}_0^{(0)} + \tilde{\lambda}_1^{(0)} x + ..... + \tilde{\lambda}_{2t-2}^{(0)} x^{2t-2} = 1$;
            $\tilde{B}^{(0)}(x) = \tilde{b}_0^{(0)} + \tilde{b}_1^{(0)} x + ....... + \tilde{b}_{2t-2}^{(0)} x^{2t-2} = 1$;
            $\tilde{\Delta}^{(0)}(x) = \tilde{\delta}_0^{(0)} + \tilde{\delta}_1^{(0)} x + ....... + \tilde{\delta}_{2t-1}^{(0)} x^{2t-1} = \Psi(x)$;
            $\tilde{\Theta}^{(0)}(x) = \tilde{\theta}_0^{(0)} + \tilde{\theta}_1^{(0)} x + ........ + \tilde{\theta}_{2t-1}^{(0)} x^{2t-1} = \Psi(x)$;
            $\tilde{\Delta}^{*(0)}(x) = \tilde{\delta}_0^{*(0)} + \tilde{\delta}_1^{*(0)} x + ....... + \tilde{\delta}_{2t-1}^{*(0)} x^{2t-1} = S(x)$;
            $\tilde{\Theta}^{*(0)}(x) = \tilde{\theta}_0^{*(0)} + \tilde{\theta}_1^{*(0)} x + ........ + \tilde{\theta}_{2t-1}^{*(0)} x^{2t-1} = S(x)$;
            $\gamma^{(0)} = 1, k^{(0)} = 0$;
A2.4: For $r = 0$ Step 1 Until $2\beta$-1 do
        A2.4.1 Compute $\tilde{\delta}_i^{(r+1)} = \gamma^{(r)} \tilde{\delta}_{i+1}^{(r)} - \tilde{\delta}_0^{(r)} \tilde{\theta}_i^{(r)}$;
                $\tilde{\delta}_i^{*(r+1)} = \gamma^{(r)} \tilde{\delta}_{i+1}^{*(r)} - \tilde{\delta}_0^{(r)} \tilde{\theta}_i^{*(r)}$;
                $\lambda_i^{(r+1)} = \gamma^{(r)} \lambda_i^{(r)} - \tilde{\delta}_0^{(r)} b_{i-1}^{(r)}$;
                $\tilde{\lambda}_i^{(r+1)} = \gamma^{(r)} \tilde{\lambda}_i^{(r)} - \tilde{\delta}_0^{(r)} \tilde{b}_{i-1}^{(r)}$;
```

A2.4.2: If $\tilde{\delta}_0^{(r)} \neq 0$ and $k^{(\mathrm{r})} \geq 0$ then $a=1$; else $a=0$;

A2.4.3: $$\begin{pmatrix} b_i^{(r+1)} \\ \tilde{b}_i^{(r+1)} \\ \theta_i^{(r+1)} \\ \theta_i^{(r+1)} \\ \gamma^{(r+1)} \\ k^{(r+1)} \end{pmatrix} = \begin{pmatrix} \lambda_i^{(r)} & b_{i-1}^{(r)} \\ \tilde{\lambda}_i^{(r)} & \tilde{b}_{i-1}^{(r)} \\ \tilde{\delta}_{i+1}^{(r)} & \tilde{\theta}_i^{(r)} \\ \tilde{\delta}_{i+1}^{*(r)} & \tilde{\theta}_i^{*(r)} \\ \tilde{\delta}_0^{(r)} & \gamma^{(r)} \\ -k^{(r)}-1 & k^{(0)}+1 \end{pmatrix} \begin{pmatrix} a \\ \bar{a} \end{pmatrix}$$

A3: Tract the longest consecutive $\tilde{\Lambda}^{(2\beta)}(x)$ that are identical, record the last element l^* of the consecutive $l's$, then the overall error locator polynomial is $\Lambda^*(x) = \Lambda^{(2\beta)}(x)$ at the l^*- th outer loop.

The overall error evaluator polynomial $\Omega^*(x)$ is corresponding $\tilde{\Lambda}^{(2t)}(x)$ at the l^*- th outer loop.
Output: $\Lambda^*(x), \Omega^*(x)$

B. Single Long Burst Error correcting algorithm

```
Input: Syndromes $S_0, S_{1,} S_2, \ldots\ldots, S_{2t-1}$;
B1: Compute $\Xi(x) = (1-\alpha^{1-(2t-1)}x)(1-\alpha^{2-(2t-1)}x)\ldots\ldots(1-x)$
            $= 1+\xi_1 x + \xi_2 x^2 + \ldots\ldots + \xi_{2t-1}x^{2t-1}$;
B2: Compute $\Gamma(x) = S_{2t-1} + S_{2t-2}\xi_1 x + S_{2t-3}\xi_2 x^2 + .. + S_0 \xi_{2t-1} x^{2t-1}$;
B3: B3.1: Find the roots of $\Gamma(x)$;
    B3.2: Track the longest consecutive roots sequence $\alpha^{s+f-1}, \alpha^{s+f}, ..\alpha^{s+2t-2}$;
    B3.3: Use $t$ to calculate the value of $s$, $f$,
   Then it can claim that burst of errors starts at position $\alpha^{-s}$, with length $f$;
B4: Compute new error locator polynomial $\Lambda^*(x) = \Xi(\alpha^{s+2t-2}x)$;
B5: Compute the new error evaluator polynomial $\Omega^*(x) = \omega_0 + \omega_1 x + .. + \omega_{2t-1}x^{2t-1}$
    where $\omega_i = \Lambda_0^* S_i + \Lambda_1^* S_{i-1} + \ldots.. + \Lambda_i^* S_0$ for $i$=0,1,2....2$t$-1;
Output: $\Lambda^*(x), \Omega^*(x)$
```

C. Random Error and Erasure Correcting Algorithm

```
Input: Syndromes $S_0, S_{1,} S_2, \ldots\ldots, S_{2t-1}$;
       Errasure Locations $Z_0, Z_1, \ldots.., Z_{\rho-1}$;   $\rho \leq 2t$ - 2
C1: Compute $\Xi(x) = (1-Z_0 x)(1-Z_1 x)\ldots(1-Z_{\rho-1}x) = 1+\xi_1 x + \xi_2 x^2 + .. + \xi_\rho x^\rho$;
C2: Compute $\Psi(x) = \psi_0 + \psi_1 x + \ldots\ldots. + \psi_{2t-1}x^{2t-1}$, where $\psi_i = \sum_{j=0}^{\rho} \xi_j S_{i+\rho-j}$;
C3: Initialization $\Lambda(0) = B(0) = \Xi(x)$;
        $\Delta^{(0)}(x) = \delta_0 + \delta_1 x + \ldots\ldots + \delta_{2t-1}x^{2t-1} = \Psi(x)$;
        $\Theta^{(0)}(x) = \theta_0 + \theta_1 x + \ldots\ldots..\theta_{2t-1}x^{2t-1} = \Psi(x)$;
        $k^{(0)} = 0, \gamma^{(0)} = 1$;
C4: For $r$=0 step 1 until 2$t$ - $\rho$ -1 do
    C4.1 Compute $\Delta^{(r+1)}(x) = x^{-1}\gamma^{(r)}\Delta^{(r)}(x) - \delta_0^{(r)}\Theta^{(r)}(x)$;
         Compute $\Lambda^{(r+1)}(x) = \gamma^{(r)}\Lambda^{(r)}(x) - x\delta_0^{(r)}B^{(r)}(x)$;
    C4.2 $\delta_0^{(r)} \neq 0$ and $k^{(r)} \geq 0$ then $a$ = 1; else $a$ = 0;
```

$$\begin{pmatrix} B^{(r+1)}(x) \\ \Theta^{(r+1)}(x) \\ \gamma^{(r+1)} \\ k^{(r+1)} \end{pmatrix} = \begin{pmatrix} \Lambda^{(r)} & xB^{(r)}(x) \\ x^{-1}\Delta^{(r)}(x) & \Theta^{(r)}(x) \\ \delta_0^{(r)} & \gamma^{(r)} \\ -k^{(r)}-1 & k^{(r)}+1 \end{pmatrix} \begin{pmatrix} a \\ \bar{a} \end{pmatrix}$$

Output: $\Lambda^*(x) = \Lambda^{(2t-\rho)}(x)$, $\Omega^*(x) = \Delta^{(2t-\rho)}(x)$

IV. UNIFIED HYBRID RS DECODER

The proposed RiBC algorithm is very effective for correcting combination of burst errors and random errors (mode-1), while sLBC and rEEC algorithms are well-suited for single burst (mode-2) and random errors and erasures (mode-3) correction. By observing the three algorithms, it can be founded that they share many common or similar computation steps. Based on this interesting similarity, a unified hybrid decoding(UHD) architecture that is capable of correcting these three different types of errors pattern (or called as three work modes) will be given in this section. Fig.3 shows the overall architecture of UHD decoder. Three types of lines illustrate data flows for different work modes: solid line formode-1, dashed line for mode-2 and dotted line for mode-3. Different blocks are used to process different steps.

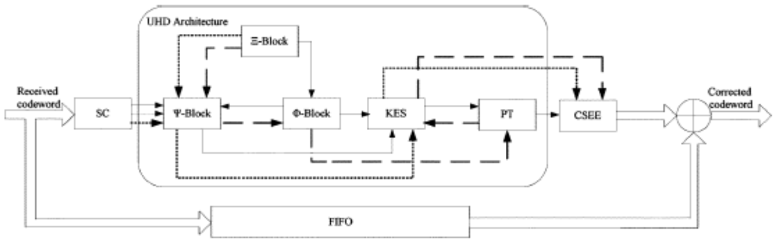


Fig.3 Unified Hybrid Decoder

4.1 Syndrome Calculator:

The syndrome is the result of a parity check performed on *R*(*x*) to determine whether *R*(*x*) is a valid member of the codeword set. If in fact *R*(*x*) is a member, the syndrome *S* has value 0. Any nonzero value of *S* indicates the presence of errors. Similar to the binary case, the syndrome *S* is made up of *n* - *k* symbols, {*Si*} (i = 1, …, *n* - *k*). Thus, for this (7, 3) R-S code, there are four symbols in every syndrome vector; their values can be computed from the received polynomial, *R*(*x*). The computation of a syndrome symbol can be described as follows:

$$S_i = R(\alpha^i) \qquad i = 1, 2....(n-k) \qquad (4.1)$$

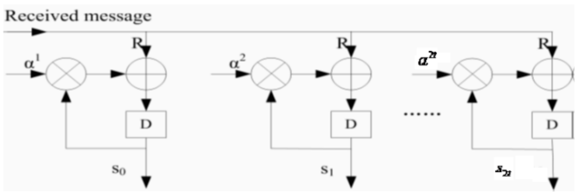


Fig.4 Syndrome Calculator

4.2 $\Xi -$ Block Architecture

$\Xi -$ block is used to process steps A1, B1, or C1 in different work modes. No matter which work mode is selected, the computation of $\Xi(x)$ is always carried out as follows:

$$\Xi(x) = \prod_{i=0}^{c} (1 - A_i x) = 1 + \xi_1(x) + \xi_2 x^2 + \ldots + \xi_c x^c \qquad (4.2)$$

where A_i denotes $\alpha^{i-(2t-2\beta)}, \alpha^{i-(2t-1)}$ or Z_{i-1} , and c denotes $2t - 2\beta, 2t - 1, \rho$.

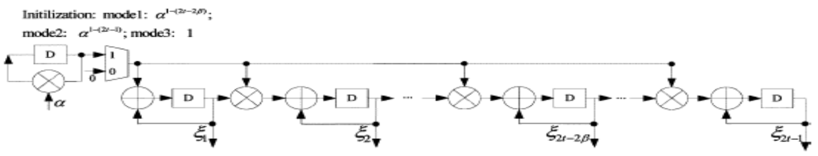

Initilization: mode1: $\alpha^{1-(2t-2\beta)}$;
mode2: $\alpha^{1-(2t-1)}$; mode3: 1
D
1
0
α
D
D
D
D
ξ_1
ξ_2
$\xi_{2t-2\beta}$
ξ_{2t-1}

Fig. 5 $\Xi -$ Block diagram

By inputting A_i to the block serially, it can be found that can be implemented as shown in Fig 5. In Fig 5, once the required work mode is selected, the left-most register is initialized as a specific value. Then after certain number of clock cycles that depends on the selected mode, each accumulate unit computes its corresponding coefficient of $\Xi(x)$. Note that if in mode-3 the decoder detects that the current received symbol is not erasured, the input 0 of multiplexer will be selected.

4.3 $\Phi -$ Block Architecture

1) Steps A2.1 and B3.1 are implemented in $\Phi -$ block as shown in the Fig.6 For these steps, the common operation is multiply-accumulate for each coefficient of the polynomial. Only a slight difference exists in step B3.1, it is a Chien Search-like step, hence an extra adder tree is required to verify the validity of current received symbol. Notice that $\Phi -$ block will be idle in mode-3. In mode-1, ξ_i , as the coefficients of $\Xi(x)$, are inputted into each multiply-accumulate unit for iterated multiplication. For each *l* in step A2.1, since $\Phi(x) = \Xi(\alpha^l x) = 1 + \phi_1 x + \phi_2 x^2 + \ldots\ldots\phi_{2t-2\beta} x^{2t-2\beta}$ should be maintained within 2*t*+3 cycles, 3:1 multiplexers are introduced to help the lower registers keep the coefficients of current $\Phi(x)$ during the above time

interval. The value of *l* will increase by 1 every 2*t*+3 cycles. Once *l* increases by 1, after 1 cycle, the lower registers will output ϕ_i to the Ψ -block.

2) In mode-2, Γ_i as the coefficients of $\Gamma(x)$, are selected to be inputted instead of ξ_i . Then by employing adder tree and zero detect, it takes *n* cycles for $\Phi(x)$ -block to find the roots of $\Gamma(x)$ (step B3.1).

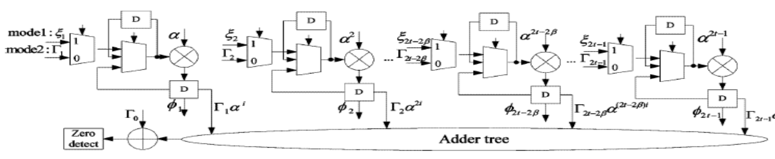


Fig. 6 Φ – Block diagram

4.4 Ψ -Block Architecture

Ψ -block is used to execute steps A2.2, B2, and C2. Actually, the inherent nature of steps A2.2, B2, and C2 is the multiplication of two polynomials. This operation can be implemented as shown in Fig. 7. In Fig. 7, the initial values of registers are all set to 0, and the operating procedures for three work modes are introduced as follows:

1) For mode-1, the coefficients of $\Phi(x)$ are serially inputted into Ψ -block. After $2t-2\beta+1$ cycles, ψ_i , as the coefficients of $\Psi(x)$, are stored in the registers.
2) For mode-2, being different from mode-1 and mode-3, the coefficients of $\Xi(x)$ are concurrently fed into Ψ -block, and then after only 1 cycle, Γ_i , as the coefficients of $\Gamma(x)$, are calculated and stored in the registers.
3) For mode-3, similar with mode-1, the coefficients of $\Xi(x)$ are serially inputted into Ψ -block. After $\rho+1$ cycles, ψ_i , as the coefficients of $\Psi(x)$, are stored in the registers.

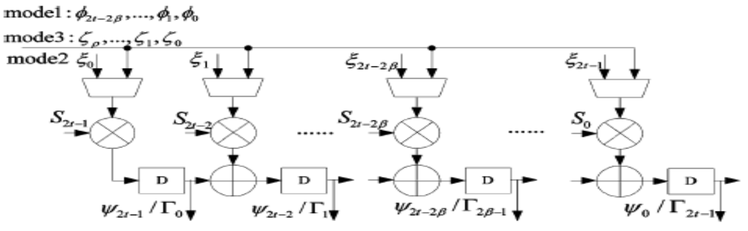


Fig. 7 Ψ - Block diagram

4.5 KES Block Architecture

In UHD decoder, KES block is employed to carry out steps A2.4, B4, B5, and C4. Fig. 8 presents the overall architecture of KES block and the internal structure of its two types of processing elements (PE), PE0 and PE1.As shown in Fig. 8(a), the KES block consists of PE0's and PE1's. The detailed operating scheme is presented as follows.

1) For mode-1, in the *r* th iteration, each register in $PE0_i/PE1_i$ stores the corresponding coefficients of different polynomials. For each outer iteration, it takes 2β cycles to compute $\lambda_i^{(2\beta)}$ and $\tilde{\delta}_i^{(2\beta)}$ as the coefficients of $\Lambda^{(2\beta)}(x)$ and $\tilde{\Delta}^{(2\beta)}(x)$ .Meanwhile, $\lambda_i^{(2\beta)}$ will also be computed and outputted into PT block to track the longest consecutive $\Lambda^{(2\beta)}(x)$ that are identical.
2) For mode-2, as aforementioned, KES block is arranged to carry out steps B4 and B5. Accordingly, both of the initial values in registers and input signals are different from those in mode-1, and they are operated based on the following schedule:
   i. First, 2*t*-1 PE0's compute step B4 $\Lambda^*(x)=\Xi(\alpha^{s+2t-2}x)$. In each PE0$_i$, the second uppermost register is initialized with ξ_{i+1} , in addition *Ctrl* and $\delta_0^{(r)}$ are always set to 1 and 0, respectively. Then after $s+2t-2$ cycles, these registers in group A just store the coefficients of polynomial $\Lambda^*(x)$.
   ii. The successive step C5 $\omega_i=\Lambda_0^*S_i+\Lambda_1^*S_{i-1}+.....+\Lambda_i^*S_0$ for *i* =0,1,..2*t*-1 is carried out by 2*t* PE1's. In each PE1, the uppermost register group B is initialized with 0,while the initial value in the

third uppermost register group C is S_{2t-1-i} ,meanwhile *Ctrl* and $\gamma^{(r)}$ are always set to 0 and 1. Additionally in the *r*-th cycle, $\delta_0^{(r)}$ is set to Λ^*_{2t-1-r} for $0 \le r \le 2t-1$ then after 2*t* cycles registers in group B store the coefficients of $\Omega^*(x)$.

3) For mode-3, the rEEC algorithm can be directly carried out by KES block by replacing $\tilde{\delta}_i^{(r)}$ by $\delta_i^{(r)}$ in each $PE1_i$'s. Note that in this case only half of the hardware component of each PE0/PE1 is utilized. Therefore it can be derived that the total throughput would be improved by twice if two independent codeword are inputted.

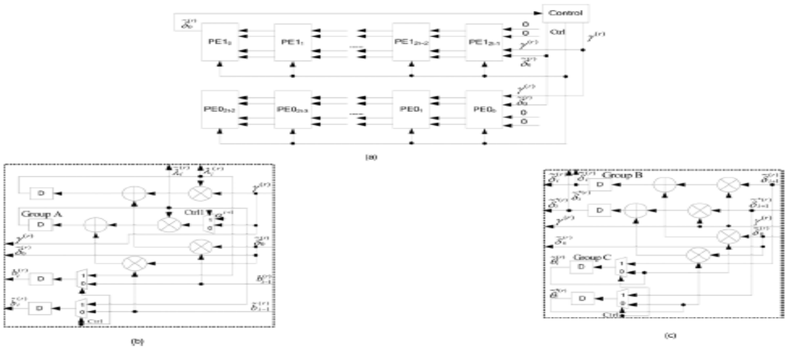


Fig. 9(a) Overall architecture of K ES block. (b) The block diagram of $PE0_i$ (c) The block diagram of $PE1_i$

4.6 Position Track (*PT*) Block Architecture

PT block is used to track the longest consecutive polynomials that are identical (step A3) or position of roots (step B3.2 and B3.3)

1) Fig.10 illustrates the architecture of the PT block for mode1 .The inputted $\tilde{\lambda}_i^{2\beta}, \lambda_i^{2\beta}$ and $\tilde{\delta}_i^{2\beta}$ from the KES block at the *l* th outer iteration are denoted as $\tilde{\lambda}_i(l), \lambda_i(l)$ and $\tilde{\delta}_i(l)$. In addition $\tilde{\lambda}_i\ (temp)$ represents $\tilde{\lambda}_i(l-1)$ while $\tilde{\lambda}_i\ (store)$ are the coefficients of current continuously identical $\tilde{\Lambda}^{2\beta}(x)$. Moreover, $\tilde{\lambda}_i\ (longest)$ stores the coefficients current longest continuously identical $\tilde{\Lambda}^{2\beta}(x)$. After *l* reaches *n* $\tilde{\lambda}_i\ (longest)$ and $\tilde{\delta}_i\ (longest)$ are outputted as the coefficients of overall error locator polynomial $\Lambda^*(x)$ and overall error evaluator polynomial $\Omega^*(x)$.
2) In mode 2 since finding the roots of $\Gamma(x)$ has been implemented in Φ -block there is no need for PT block to carry out this function, but it is used to track the longest consecutive roots of $\Gamma(x)$.

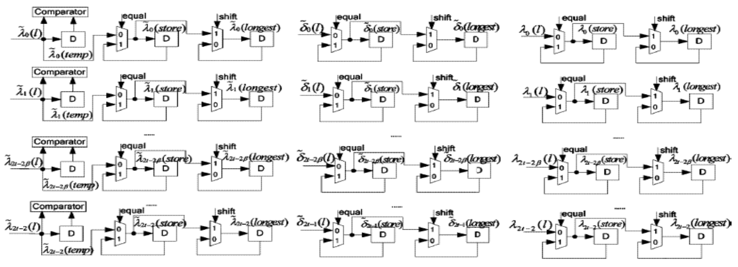
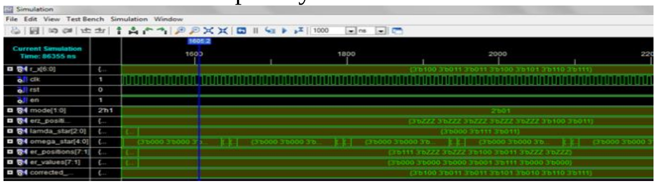


Fig. 11 Architecture of PT block in mode 1

V. SIMULATION RESULTS AND DESIGN SUMMARY

5.1 Simulation Result

The Simulation is done in the three modes of a (7, 3) RS Decoder for the received signal *R*(*x*) = {100 011 011 10<u>0 101</u> 110 111} with the 4th and 5th corrupted symbols.



```
Current Simulation
Time: 86355 ns
r_x[6:0]          {...   {3'b100 3'b011 3'b011 3'b100 3'b101 3'b110 3'b111}
clk               1
rst               0
en                1
mode[1:0]         2'h1   2'b01
erz_positi...     {...   {3'bZZZ 3'bZZZ 3'bZZZ 3'bZZZ 3'bZZZ 3'b100 3'b011}
lamda_star[2:0]   {...   {3'b000 3'b111 3'b011}
omega_star[4:0]   {...   {3'b000 3'b000 3'b...
er_positions[7:1] {...   {3'b111 3'bZZZ 3'bZZZ 3'b100 3'b011 3'bZZZ 3'bZZZ}
er_values[7:1]    {...   {3'b000 3'b000 3'b000 3'b001 3'b111 3'b000 3'b000}
corrected_...     {...   {3'b100 3'b011 3'b011 3'b101 3'b010 3'b110 3'b111}
```

Fig. 12 Simulation Result in mode 1(RiBC)

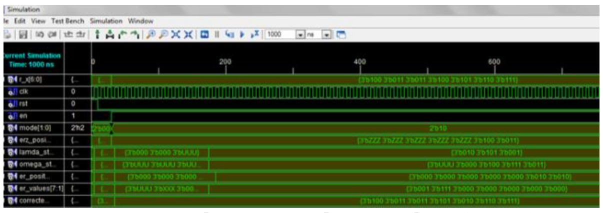


Fig. 13 Simulation Result in mode 2 (sLBC)

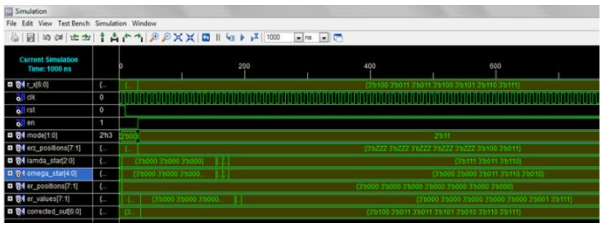
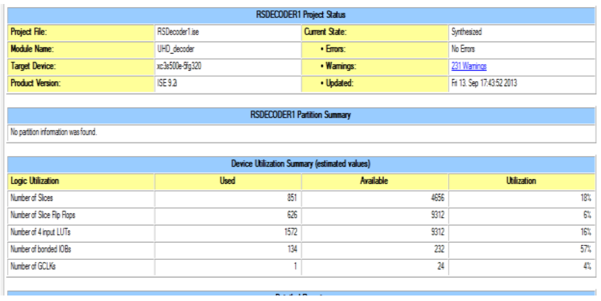


Fig. 14 Simulation Result in mode 3 (rEEC)

5.2 Design Summary

The Fig.15 gives the detailed design summary of the UHD decoder.



RSDECODER1 Project Status			
Project File:	RSDecoder1.ise	**Current State:**	Synthesized
Module Name:	UHD_decoder	**• Errors:**	No Errors
Target Device:	xc3s500e-5fg320	**• Warnings:**	231 Warnings
Product Version:	ISE 9.2i	**• Updated:**	Fri 13. Sep 17:43:52 2013

RSDECODER1 Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	851	4656	18%
Number of Slice Flip Flops	626	9312	6%
Number of 4 input LUTs	1572	9312	16%
Number of bonded IOBs	134	232	57%
Number of GCLKs	1	24	4%

Fig.15 Design Summary of the UHD decoder

VI. CONCLUSION

In this brief a Unified Hybrid Decoder architecture that can support three different modes so that it can correct three different types of error pattern is implemented. Also the simulation results and design summary for a (7,3) RS Decoder is shown.

REFERENCES

[1] E. R. Berlekamp, Algebraic Coding Theory. New York: McGraw-Hill, 1968. (revised ed.—Laguna Hills, CA: Aegean Park, 1984).
[2] R. E. Blahut, Theory and Practice of Error-Control Codes. Reading, MA: Addison-Wesley, 1983.
[3] E. Dawson and A. Khodkar, “Burst error-correcting algorithm for Reed-Solomon codes,” Electron. Lett., vol. 31, pp. 848–849, 1995.
[4] D. V. Sarwate and N. R. Shanbhag, “High-speed architectures for Reed-Solomon decoders,” IEEE Trans. Very Large Scale Integr.(VLSI) Syst., vol. 9, no. 5, pp. 641–655, Oct. 2001.
[5] Z. Wang and J. Ma, “High-speed interpolation architecture for soft-decision decoding of Reed-Solomon codes,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 14, no. 9, pp. 937–950, Sep. 2006.
[6] Y. Wu, “Novel burst error correcting algorithms for Reed-Solomon codes,” in Proc. IEEE Allerton Conf. Commun., Control, Comput., 2009, pp. 1047–1052.
[7] L. Yin, J. Lu, K. B. Letaief, and Y. Wu, “Burst-error-correcting algorithm for Reed-Solomon codes,” Electron. Lett., vol. 37, no. 11, pp. 695–697, May 2001.
[8] T. Zhang and K. K. Parhi, “On the high-speed VLSI implementation of errors-and-erasures correcting Reed-Solomon decoders,” in Proc. ACM Great Lake Symp. VLSI (GLVLSI), 2002, pp. 89–93.
[9] X. Zhang and J. Zhu, “High-throughput interpolation architecture for algebraic soft-decision Reed-Solomon decoding,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 3, pp. 581–591, Mar. 2010.