Chapter 11

Efficient Forward Error Correction Decoder Design for High-Speed Optical Networking

Bo Yuan, Li Li and Zhongfeng Wang

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48291

1. Introduction

Due to the fast development of Internet, the traffic load of data network has increased dramatically in past decades. Accordingly, optical network, as the major carrier for data transmission, needs to increase its capacity to meet the increasing data rate requirement. As stated in the white paper of Optical Internetworking Forum (OIF) (see [1]), the rapid growth of data flow demands optical network to double its capacity every 12-18 months. As a result, this critical requirement pushes various types of optical transmission systems to improve their delivered data rate at the same time.

Generally, in high-speed optical communication, the increase on data rate usually comes with the increase on signal bandwidth and sampling rate. In this case, due to the sensitiveness of optical and electronic devices, the additive transmission noise will inevitably increase as well. Therefore, how to increase the throughput of optical network without loss of robustness is an essential task when designing modern high-speed optical network.

To date, various techniques have been employed to enhance the quality of data transmission in optical network. Among those approaches, Forward error correction (FEC), or commonly called error correction coding (ECC), is viewed as the most cost-effective solution, and has been widely adopted in many industrial optical transmission systems. Many specific FEC codes, including Reed-Solomon (RS), BCH, and LDPC codes, are proposed in different industrial standards for error correction in physical layer. Among them, RS code is the earliest and most widely used FEC code in optical communication. For example, RS (255, 239) was the first generation FEC code for submarine fiber-optical transmission in [2], and its code rate is still the standard parameter for frame design. In addition, for Ethernet network such as 10GBase-LR in [3], Reed-Solomon (255, 239) code is also the standard FEC code.

There are several reasons for the wide application of RS code. First, modern long-distance optical network, especially long-haul network, is very high-speed system (10Gbps and beyond). For other promising FEC codes, such as LDPC or Turbo code, the corresponding decoding throughput usually can not meet such stringent requirement on data rate, or with the penalty of very high hardware complexity. Instead, RS decoder can achieve such high throughput with affordable hardware resource. Second, for local optical network, such as Ethernet network, the real-time response is an important metric for system design. Compared with its counterpart, RS code has the particular advantage on low decoding latency. Therefore, RS codes are widely employed in modern optical transmission system and are believed to play an important role in next generation optical networks.

Considering the importance of RS code, its efficient implementation is quite important for the optical transmission system. A low-complexity high-speed RS encoding/decoding system will improve the overall performance significantly. Particularly, since RS decoding is the most complex procedure in the RS-based FEC system, efficient RS decoder design should be well-studied. Therefore, targeted to different level of optical communication ranging from short-distance Ethernet network to long-haul backbone system, this chapter fully introduces efficient VLSI design of RS decoder. In addition, to meet the requirement of 100Gbps era, this chapter also discusses some new FEC schemes for ultra high-speed application (beyond 100Gbps).

The chapter is organized as follows. Section 2 reviews the RS decoding. The low-complexity high-speed RS decoders for short-distance network are discussed in Section 3. Section 4 analyzes performance-improved RS burst-error decoder for medium-distance system. Some recent FEC schemes targeted to 100Gbps long-haul network are introduced in Section 5. Section 6 draws the conclusion.

2. Review of RS decoding

According to the coding theory in [4], the procedure for decoding RS code contains three main steps: **syndrome computation** (SC), **key equation solving** (KES) and **Chien search & error evaluation** (CSEE). Therefore, the decoding procedure of RS code is summarized as below:

Step 1. (Syndrome computation): For an (n, k) RS code defined over GF(2^m) whose primitive element is α in reference [4], let $\mathbf{C}(x)$ and $\mathbf{R}(x)$ be the transmitted and received codeword polynomial respectively, and then assumes $\mathbf{R}(x) = \mathbf{C}(x) + \mathbf{E}(x)$, where $\mathbf{E}(x)$ is the error polynomial which reflects the errors induced by transmission channel noise. Then, the syndrome polynomial $\mathbf{S}(x)$ is computed as follows:

$$\mathbf{S}(x)=s_0+s_1x+s_2x^2+\ldots+s_{2t-1}x^{2t-1}, \text{ where } s_i=R(\alpha^{i+1}) \text{ and } t=(n-k)/2. \quad (1)$$

The architecture of SC block of an example RS (255, 239) decoder is shown in Fig. 1. Here $\mathbf{R}(x)=r_{n-1}x^{n-1}+r_{n-2}x^{n-2}+\ldots+r_1x+r_0$ is serially transmitted to SC block with the sequence of r_{n-1} , r_{n-2} ,…,r_0. Every partial syndrome is calculated with shown multiply-accumulate circuits (MAC) in every clock cycle. After n=255 clock cycles, the $2t$=16 syndromes are computed and serially transmitted to the next KES block.

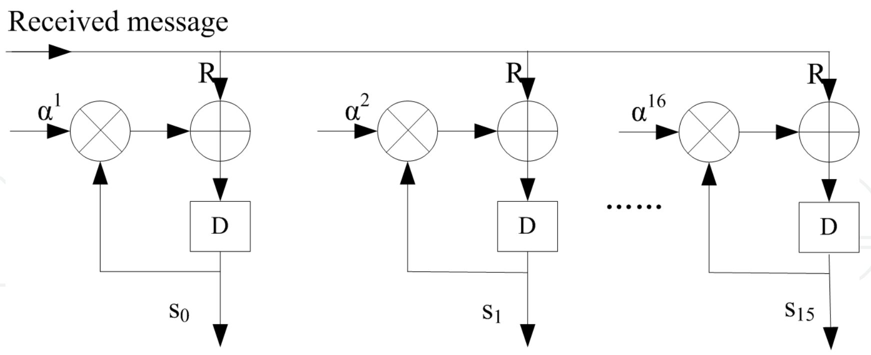


Figure 1. The block diagram of syndrome computation for example RS (255, 239) code.

Step 2. (Key equation solving): With the help of inputted $\mathbf{S}(x)$, in this step, Key equation solver (KES) block will calculate error evaluator polynomial $\mathbf{\Omega}(x)$ and error locator polynomial $\mathbf{\Lambda}(x)$ by solving key equation: $\mathbf{\Lambda}(x)\mathbf{S}(x)\equiv\mathbf{\Omega}(x) \bmod x^{2t}$. This part is the most important step in the whole RS decoding procedure, which usually dominates the performance of the overall decoder. Therefore, in this chapter, we focus on the algorithm and architecture optimization of KES block.

Generally, Berlekamp-Massey (BM) algorithm or modified Euclidean (ME) algorithm can be employed to solve key equation. To data, many efforts have addressed for efficient VLSI implementation of the above two algorithms. In [5], BM algorithm was reformulated as RiBM with the same regular architecture format compared with conventional ME algorithm in [6] and [7], and a folded BM algorithm based on RiBM was introduced in [8]. Reference [6] and [7] implemented conventional ME algorithm with systolic and recursive architecture. In Section 3 and Section 4, based on the above efforts, some improved KES algorithms and their corresponding hardware implementations will be discussed for efficient RS decoder design.

Step 3. (Chien search & error evaluation): After KES block finishes its computation for the current codeword, the calculated error locator polynomial $\mathbf{\Lambda}(x)$ and the error evaluator polynomial $\mathbf{\Omega}(x)$ will be outputted to CSEE block to generate the error positions and magnitudes.

Chien search is a widely employed approach to look for error position. Its basic idea is simple but efficient: If $\Lambda(\alpha^{-i})$=0 for current i, it indicates that the i-th symbol of the received codeword is wrong and needs to be corrected. After obtaining the position of error, the following Forney algorithm is applied to determine the error value:

$$Y_i = -\left.\frac{\Omega(x)}{x\Lambda'(x)}\right|_{x=\alpha^{-i}} \tag{2}$$

where Y_i is the error magnitude for the i-th erroneous symbol.

Based on the above described Chien search and Forney algorithm, the architecture of CSEE block for example RS (255, 239) code is illustrated in Fig. 2. It consists of several unit cells (shown in Fig. 2(a)). Both of the sub-blocks that carry out Chien search and Forney algorithm consist of these basic cells. In the beginning, λ_i, and ω_i, (represented by U_i in the figure), as the coefficients of $\mathbf{\Lambda}(x)$ and $\mathbf{\Omega}(x)$, are parallel loaded into these basic cells (enable=1). Then, during the next 255 cycles, those basic cells will carry out multiply iteratively. Fig. 2(b) is the overall architecture for CSEE block. Once a zero is detected in Chien search, the corresponding error magnitude will be computed via executing the above Forney algorithm.

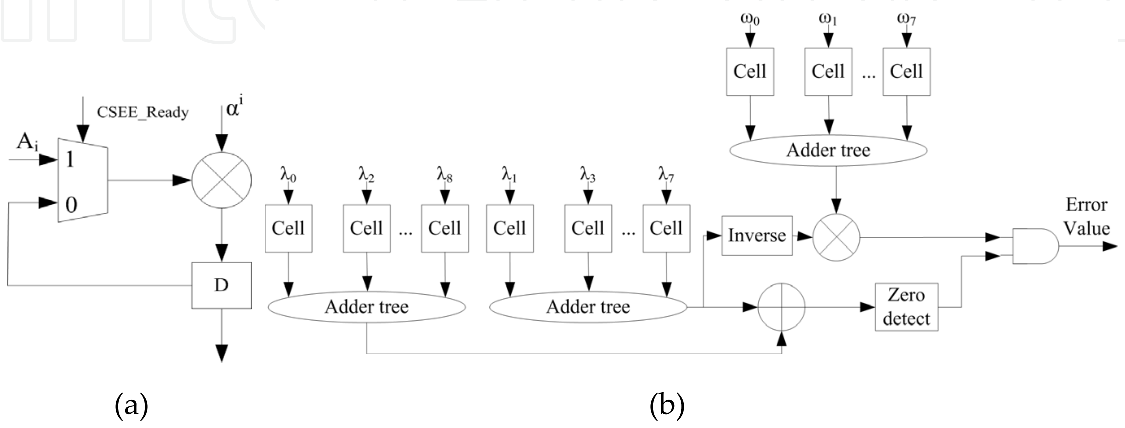


Figure 2. (a) The diagram of CSEE cell. (b) The block diagram of CSEE.

The overall architecture of RS decoding is summarized in Fig. 3.

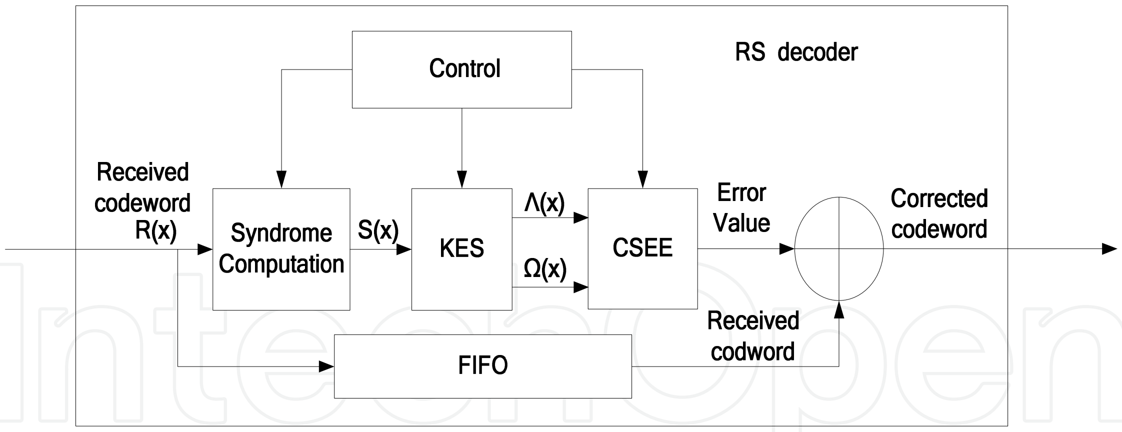


Figure 3. The overall architecture of RS decoder.

As mentioned in previous paragraph, since KES is the dominating step in the whole RS decoding, Section 3 and 4 will focus on the algorithm and architecture optimization of KES block.

3. Low-complexity high-speed RS decoders for short-distance network

For short-distance optical transmission, such as 10GBase-LR, since the noise rendered from transmission distance is quite limited, the requirement of coding gain is not as strict as long-distance backbone network (which will be discussed later). Therefore, as discussed in Section 1, Reed-Solomon (255, 239) code is widely used in this kind of network due to its high code rate and good error correction capability.

Although coding gain is not the major concern in this scenario, because of limited hardware resource, in order to implement efficient RS decoder, the designers have to consider the challenge of achieving high data rate with low hardware complexity. Accordingly, optimization of RS decoding architectures is necessary for high-efficiency hardware implementation.

In this section, based on the two main RS decoding algorithms, the improved ME-based and BM-based decoders are introduced.

3.1. rDCME-based RS decoder

In traditional ME algorithm, the inherent degree computation and systolic architecture renders large consumption of area and power (see [6] and [7]), which is not suitable for the discussed application. To reduce the unnecessary degree computation, DCME algorithm was introduced in [9]. By generating internal switch and shift signals, the DCME algorithm can achieve the same function as ME algorithm without degree computation.

DCME algorithm

```
1:  The initial conditions:
2:  $R_0 = xS(x)$   $Q_0 = x^{2t}$   $L_0 = x$   $U_0 = 0$
3:   $i = 0$
4:  Iteration phase:
5:    while $i < 2t-1$
6:       generate leading coefficients of $R_i$ and $Q_i$: $a_i$ and $b_i$
7:       generate switch ($sw$) and polynomial computation ($pc$) signals by control FSM
8:       if ($sw$=1) then switch $R_i$ and $Q_i$;
9:                     switch $L_i$ and $U_i$;
10:                    switch $a_i$ and $b_i$;
11:      if ($pc$=1) then
12:           $R_{i+1} = b_i R_i + a_i Q_i$
13:           $Q_{i+1} = Q_i$ (delay a clock cycle)
14:           $L_{i+1} = b_i L_i + a_i U_i$
15:           $U_{i+1} = U_i$ (delay a clock cycle)
16:      else polynomial of $R_i$ is shifted by one degree
17:      $i = i + 1$
18: Output $L_i$ as error locator polynomial and $R_i$ as error value polynomial
```

It is needed to point out that the initialization of DCME and ME is different due to the consideration of the design of the following introduced FSM.

Fig. 4 shows the FSM for generating control signals. In each iteration, there are two possible states: S0 and S1. S0 represents the case when both of a_i and b_i are nonzero; otherwise the state of FSM is S1. The different combinations of the current and previous states will determine control signals in the current iteration.

When the leading coefficients of R_i and Q_i are both nonzero, it denotes that polynomial computation can be carried out (*pc*=1), otherwise shift operation would be performed (*pc*=0) to reduce leading coefficient (and in this case the leading coefficient must be zero, the details will be shown in next paragraph). In each iteration, the possible shift operation would be executed once at most. The whole shift process would not stop until both of a_i and b_i are nonzero, which means the degrees of R_i and Q_i are equal again. And in that case KES block starts executing polynomial computation. So it is clear that in each iteration the algorithm would perform only shift operation or only polynomial computation operation.

It should be pointed out that after every polynomial computation, if being carried out, the original leading coefficient of R_{i+1} must be zero due to the arithmetic character of $R_{i+1} = b_iR_i + a_iQ_i$. Different from the leading coefficients referred in above paragraph, this kind of leading zero is a "false" leading coefficient which will cause logic errors in next iteration. (For example, after polynomial computation if R_{i+1} is represented by 0, 0, 0, α^2, α^3, the "false" leading coefficient is the first zero, and the real representation of R_{i+1} should be0, 0, α^2, α^3.) So in every possible polynomial computation process, the designed rDCME KES block has automatically eliminated this kind of leading zero with the aid of "*start*" signal in hardware design (Fig.5): the coefficients which arrive simultaneously with "*start*" signal are selected as the leading coefficients. So once polynomial computations are finished, by delaying Q_{i+1}, U_{i+1} and *start* signal one more clock cycle, the "false" leading zero is eliminated, and degR_{i+1} is one less than deg R_i or equal to it (this condition happens when the previous iteration's actual input is xR_i brought by initial input R_0=$xS(x)$). Then a_i and b_i represent the real leading coefficients respectively.

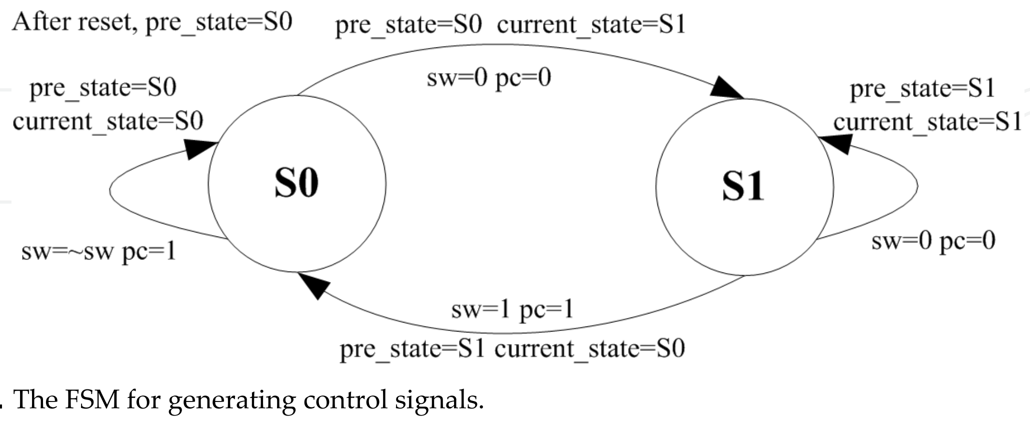


Figure 4. The FSM for generating control signals.

If the previous state and the current state are both S0, it indicates that the polynomial computation is able to be executed in the two successive iterations. So *pc*=1 since the current operation is polynomial computation. Due to the fact the previous state is S0, after the previous polynomial computation and the degree reduction, degR_i is one smaller than deg

R_{i-1} or equal to it. So deg R_i is the same with degQ_i or one smaller than degQ_i. These two possible conditions occur successively and the switch signal (*sw*) alternates successively (*sw*=~*sw*).

If the previous state is S0 and the current state is S1, KES block would process shift operation to eliminate leading zero (*pc* is set to 0) in the current iteration, because S1 shows leading coefficient is zero. *sw* is also 0 because switch operation always be carried out with polynomial operation.

If the previous state and the current state are both S1, it indicates that the two successive iterations are both in shifting operations. Similar with the above condition, *sw* and *pc* are both set to 0.

If the previous state is S1 and the current state is S0, the polynomial computation would be executed (*pc*=1) in the current iteration. Since in the previous iteration R_i is in shift operation (Q_i is never in shift operation because of its character in polynomial computation, it is also guaranteed by rDCME's initial conditions), actual degree of R_i must be smaller than Q_i, so *sw* is set to 1.

After $2t$=16 iterations, the rDCME KES block stops and outputs error value polynomial $R(x)$=$\Omega(x)$ and error locator polynomial $L(x)$=$\Lambda(x)$.

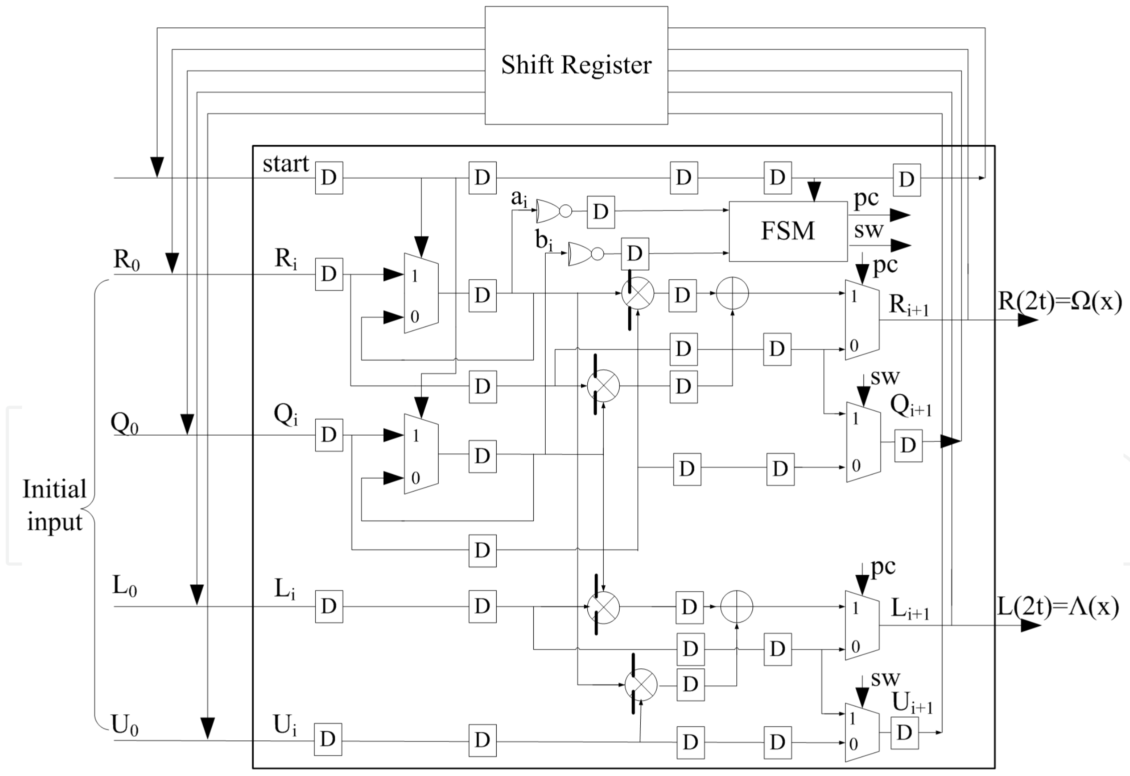


Figure 5. The block diagram of KES.

Fig. 5 shows the detailed architecture of rDCME algorithm. The KES block is designed with single PE. It is commonly known that recursive architecture usually can not be pipelined due to data dependency. And recursive architecture is always a bottleneck for high-speed. However, in the rDCME architecture, these disadvantages can be avoided. A 11-stages ($2t$-5=11) shifter registers are used to store the last iteration results and feedback to the next iteration for avoiding dependency between successive iterations: At the end of each iteration, the leading coefficients of five updated inputs (R, Q, L, U and *start*) are just stored back into the leftmost registers of shift registers and ready to be updated in the next iteration. Because during the computation procedure the whole iteration process of KES block is a close loop, the property of leading coefficients' in-time arrival makes dependency between iterations be avoided and logical validity guaranteed. Furthermore, because the former SC block takes n clock cycles to output one codeword, the PEs in conventional systolic DCME architecture in [10] and [11] are idle in the most of processing time and at the same time it occupies a large amount of chip area. So the multi-stages pipeline can be employed in the area efficient recursive KES block with valid logic and only a little data processing rate degradation. Note that in Fig. 5 the multipliers are pipelined.

Architect.	**rDCME**	**pDCME in [10]**	**DCME in [11]**	**PrME in [7]**
Tech.(μm)	0.18	0.13	0.25	0.13
PE	1	2t	3t+2	1
SC	2900	2900	2900	2900
KES	11400	46200	21760	17000
CSEE	4100	4100	4100	4100
Total gates	18400	53200	28760	24000
fmax(MHz)	640	660	200	625
Throughput (Gb/s)	5.1	5.3	1.6	5

Table 1. Implementation results and comparisons

Table 1 presents performance comparisons between the rDCME RS decoder and other existing RS decoders. It can be observed that the rDCME decoder has very low hardware complexity and high throughput. Compared with the existing ME architectures, the total gate count of the rDCME architecture is reduced by at least 30.4%. Therefore, the hardware efficiency is improved at least 1.84 times, which means under the same technology condition our design would be much more area-efficient compared with other existing RS decoder designs for multi-Gb/s optical communication systems.

3.2. PI-iBM-based RS decoder

Besides ME algorithm, BM algorithm is another main decoding approach for RS codes. An important and inevitable disadvantage of traditional iBM/RiBM algorithms is the high cost of area or iteration time for computing error value polynomial $\mathbf{\Omega}(x)$. In iBM architecture stated in [12], one third of total iteration time or half of hardware complexity is employed to compute $\mathbf{\Omega}(x)$; in RiBM architecture stated in [5], one third of processing elements (PE) are utilized to calculate and store $\mathbf{\Omega}(x)$. Therefore, the calculation of $\mathbf{\Omega}(x)$ impedes further performance improvement of current BM architectures.

The PI-iBM algorithm employs simplified Forney algorithm to compute error values. Simplified Forney algorithm, presented in [13] and [14], replaces $\Omega(x)$ with scratch polynomial B(x) as follows:

$$Y_i = \left. \frac{\lambda_0 \delta}{x B(x) \Lambda^{'}(x)} \right|_{x=\alpha^{-i}}$$

In each iteration, scratch polynomial $\mathbf{B}(x)$, discrepancy δ, error locator polynomial $\mathbf{\Lambda}(x)$ and its coefficient λ_0 are simultaneously updated. After completing iteration, KES block outputs them to CSEE block for calculating error values Y_i. So the computation of $\mathbf{\Omega}(x)$ is completely eliminated, which enables KES block to reduce a large amount of extra computation circuitry and iteration time.

Furthermore, in order to reduce hardware complexity significantly without sacrificing throughput per unit area, pipeline interleaving techniques in [15] is employed in the PI-iBM algorithm and architecture proposed in [16].

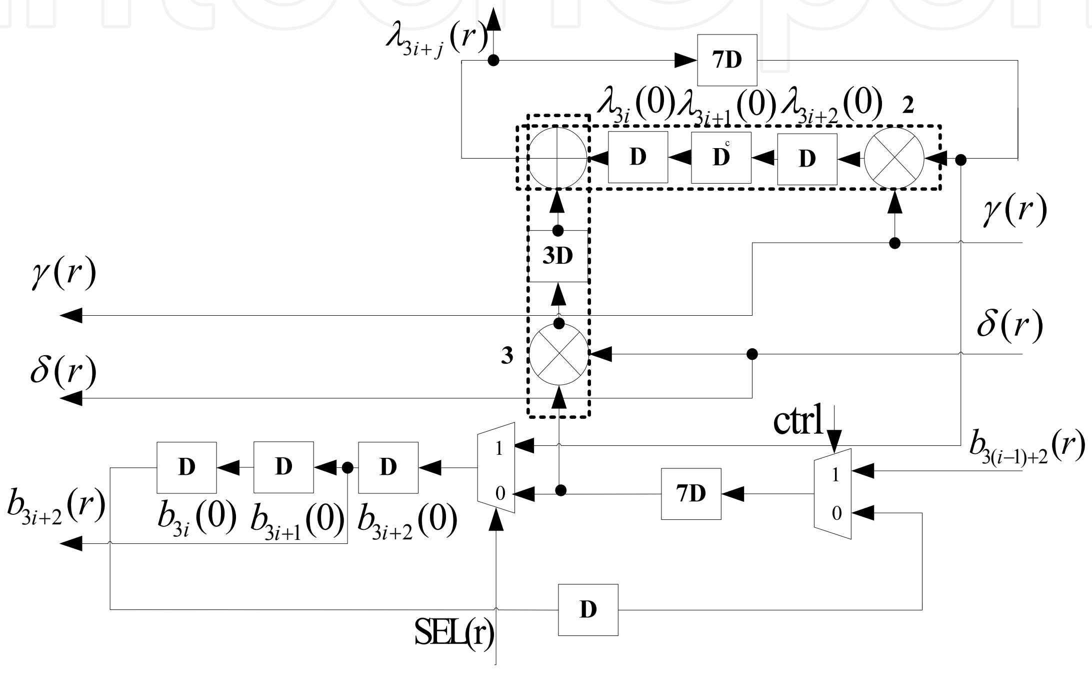
As depicted in the following PI-iBM algorithm, interleaving factor g is a crucial factor to design overall architecture. In practical RS (n, k, t) codes, such as (255, 239, 8) code, t=8 is a common value. So in this paper we set both p and g as 3 for demonstrating PI-iBM architecture.

The PI-iBM architecture consists of two blocks: pipeline interleaving error locator update (PI-ELU) block and pipeline interleaving discrepancy computation (PI-DC) block. As it is illustrated in Fig. 6, PI-ELU block is designed to execute Step3 for updating polynomials. Fig. 6(a) shows the internal architecture of the i-th PE. Initial values of upper and leftmost registers are shown in the figure and other registers are initialized to zero. For the i-th PE, in each iteration 10 cycles are required to update the stored coefficients of $\mathbf{\Lambda}(x)$ and $\mathbf{B}(x)$, meanwhile "ctrl" signal is set to be "1 0 0 0 0 0 0 0 0 0". At the beginning of r-th iteration, $b_{3i}(r)$, $b_{3i+1}(r)$, $b_{3i+2}(r)$ are stored in the leftmost three registers with $\lambda_{3i}(r\text{-}1)\gamma(r\text{-}1)$, $\lambda_{3i+1}(r\text{-}1)\gamma(r\text{-}1)$, $\lambda_{3i+2}(r\text{-}1)\gamma(r\text{-}1)$ in the upper three registers, then they are shifted in the upper and lower loops to be updated. During the first 3 cycles, $\lambda_{3i}(r)$, $\lambda_{3i+1}(r)$, $\lambda_{3i+2}(r)$ are successively computed and outputted to PI-DC block for calculating discrepancy $\delta(r)$ (Step 1). After current iteration is completed, $b_{3i}(r+1)$, $b_{3i+1}(r+1)$, $b_{3i+2}(r+1)$ and $\lambda_{3i}(r)\gamma(r)$, $\lambda_{3i+1}(r)\gamma(r)$, $\lambda_{3i+2}(r)\gamma(r)$ are just fed back to the initial registers which stored them at the beginning. The two dashed rectangles indicate that the critical path between lower multiplier and adder has been 3-stage fine-grain pipelined; the path between upper multiplier and adder is tackled in the same way.

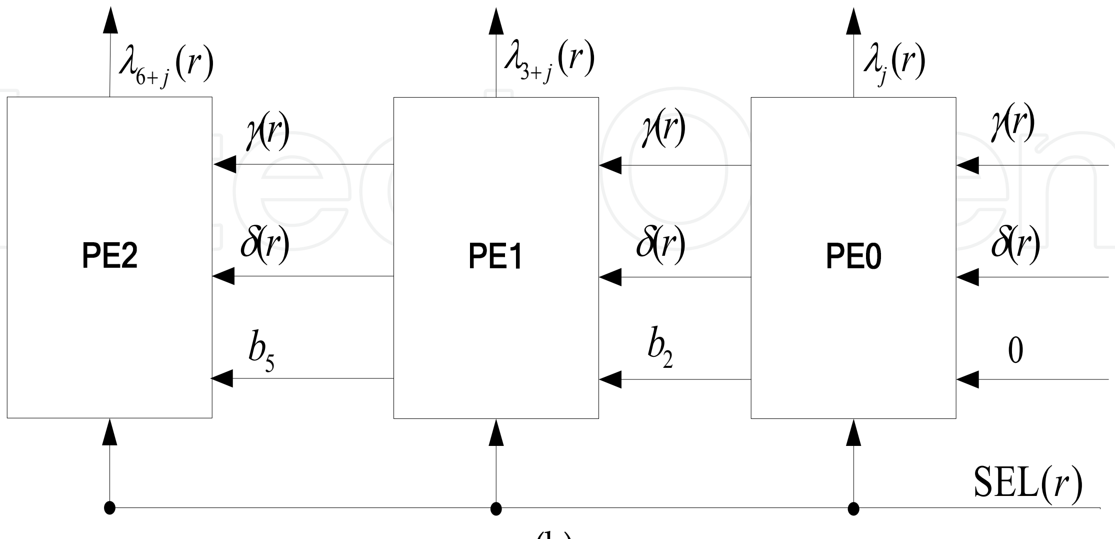
In addition, PI-DC block mainly implements the function of updating discrepancy $\delta(r)$ (Step1). A low-complexity and high-speed architecture of PI-DC block is shown in Fig. 7. As shown in Fig. 7, 2t-1 syndromes are serially sent to PI-DC block and shifted in the upper t+1 registers **every 10 cycles.** The initialization of leftmost register is S_0 while other registers are initialized to zero. In each iteration "ctrl 1"signal is set to be "0 0 0 0 0 0 1 0 0 0". In the first 5 cycles of each iteration, input λ_j, λ_{3+j}, λ_{6+j} and corresponding syndromes selected by multiplexers are multiplied by three 3-stage pipelined multipliers (shown by dashed lines). At the end of 6-th cycle accumulator circuit computes $\delta(r)$ and outputs it to control block for updating $\gamma(r)$ and SEL(r). Passing another register which cuts path between PI-ELU and PI-DC in control block, the three signals are fed back to PI-ELU block. In the overall architecture of PI-iBM (Fig. 8), it takes 7 cycles to calculate and output $\delta(r)$ (PI-DC block), and another 3 cycles is the cost for calculating new coefficients (PI-ELU block), so the total time for one iteration is 10 cycles.

```
The PI-iBM Algorithm
Initialization and Input:
  Let $t+1 = p \times g$, where $g$ is the coefficient of pipeline interleaving;
  $\lambda_0(0) = b_0(0) = 1$,
  $\lambda_l(0) = b_l(0) = 0$ for $l = 1, 2, ..., t$;
  $k(0) = 0$, $\gamma(0) = 1$;
  $S(x) = S_0 + S_1x + S_2x^2 + ... + S_{2t-1}x^{2t-1}$.
 Iteration Process:
 for $r = 0$ step 1 until $2t$ - 1 do
  Begin
   Step1: $\delta(r) = S_r\lambda_0(r) + S_{r-1}\lambda_1(r) + ... + S_{r-t}\lambda_t(r)$;
   Step2: If $\delta(r) \neq 0$ and $k(r) \geq 0$ then $a = 1$; else $a = 0$;
   Step3: for $j = 0$ step 1 until $g-1$ do
         Begin
          Step3.1: $\lambda_{gi+j}(r+1) = \gamma(r)\lambda_{gi+j}(r) - \delta(r)b_{gi+j-1}(r)$ ;
          Step3.2: $b_{gi+j}(r+1) = \begin{bmatrix} \lambda_{gi+j}(r) & b_{gi+j-1}(r) \end{bmatrix} \begin{bmatrix} a \\ - \\ a \end{bmatrix}$ for $i = 0,1,...p-1$;
         End
   Step4: $\begin{bmatrix} \gamma(r+1) \\ k(r+1) \end{bmatrix} = \begin{bmatrix} \delta(r) & \gamma(r) \\ -k(r)-1 & k(r)+1 \end{bmatrix} \begin{bmatrix} a \\ - \\ a \end{bmatrix}$;
  End
 Output: $\Lambda(x) = \lambda_0(2t) + \lambda_1(2t)x + \lambda_2(2t)x^2 + ... + \lambda_t(2t)x^t$;
         $B(x) = b_0(2t) + b_1(2t)x + b_2(2t)x^2 + ... + b_t(2t)x^t$.
```

Table 2 gives the implementation results of PI-iBM decoder and also lists some other designs. From this table we can find that the PI-iBM architecture deliver very high throughput with relatively low hardware complexity: the total throughput rate and throughput per unit area in the PI-iBM design are at least 200% more than those existing works. To achieve data rates from 10 Gb/s to 100 Gb/s, PI-iBM decoder has the lowest hardware complexity. If 65 nm CMOS technology is used in the implementation, the throughput of our design can be increased significantly. Thus the current designs can fit well for 10 Gb/s-40 Gb/s optical communication systems. For 100 Gb/s applications, we may need two to three independent hardware copies of the designs. However, the PI-iBM architecture will remain to have the lowest hardware complexity compared with existing designs. In short, the PI-iBM decoder is very area-efficient for very high-speed optical applications.



(a)



(b)

Figure 6. The diagram of PI-ELU block. (a) The internal architecture of the *i*-th PE. (b) The overall architecture of PI-ELU block.

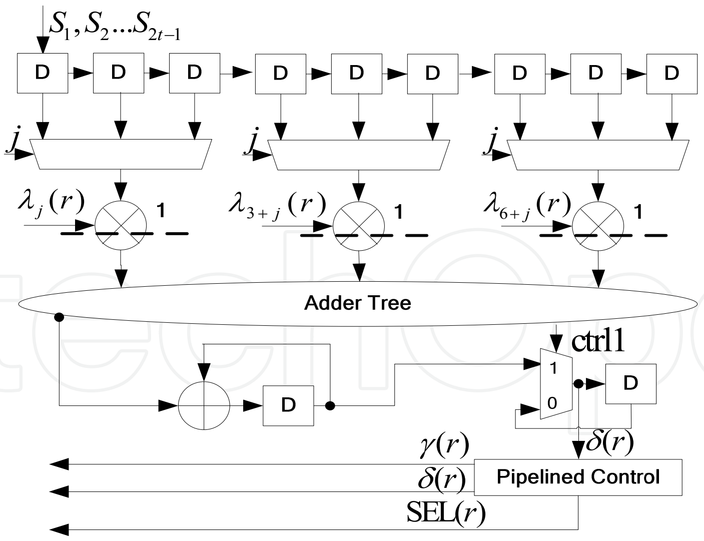


Figure 7. The diagram of PI-DC block.

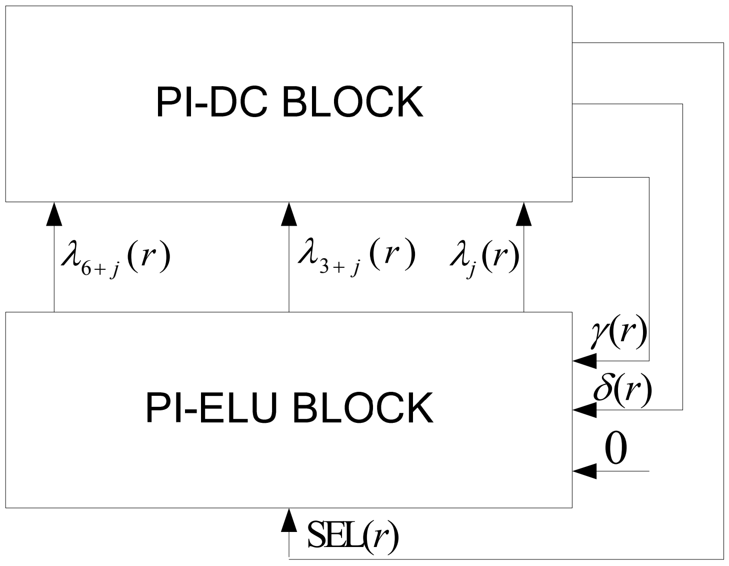


Figure 8. The diagram of overall PI-iBM architecture.

Design	Tech. (µm)	Total of gates	fmax (MHz)	Throughput (Gb/s)	Latency (cycles)
Retimed iBM in [17]	0.18	8423	654	5.23	288
Multi-mode RiBM in [18]	0.18	9566	400	3.20	128
Systolic ME in [19]	0.13	102500	770	6.16	80
Folded ME in [7]	0.13	17000	625	5.00	256
PI-iBM	0.18	10951	980	12.5	160

Table 2. Implementation results and comparisons

4. High-performance low-complexity RS burst-error decoders for mediate distance network

For mediate distance optical network, such as Metro Ethernet network, traditional RS decoding can not provide enough coding gain for the data transmission in this scenario. Instead, enhanced FEC scheme should be employed for improved error-correcting capability. Notice that in this kind of systems, long burst error is the major error pattern in transmission procedure; therefore, burst-error decoding algorithm and architecture are attractive solutions for this case. In this chapter, we introduce efficient burst-error-correcting RS decoder to meet the requirement in this type of application.

4.1. Reformulated inversionless burst-error correcting (RiBC) algorithm

As excellent Maximum Distance Separable (MDS) code [20], RS code is very effective in correcting long burst errors. However, previous RS burst decoding algorithms in [21] and [22] are infeasible for hardware implementation due to their high computation complexity. In [20], Wu 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, this approach can correct a long burst of errors with length up to $2t$-1-2β plus a maximum of β random errors. Here β is a pre-chosen parameter that determines the specific error correcting capability. In this case, the miscorrection probability is upper bounded by $(n\text{-}2f)(n\text{-}f)^{\beta}2^{m(\beta+f\text{-}2t)}$.

Although the approach in [20] has reduced computation complexity, it still contains inversion operation and long data path, which impedes its efficient VLSI implementation, therefore, the algorithm in [20] was reformulated to the RiBC algorithm. The RiBC algorithm is a kind of list decoding algorithm. 8 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 l^* of 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 $(n\text{-}2f)(n\text{-}f)^{\beta}2^{m(\beta+f\text{-}2t}$.

The RiBC algorithm is targeted for correcting burst error plus some random errors. By observing step2.3 and step 2.4, it can be founded that both of them are quite similar to the essential update equations in RiBM algorithm (see [5]). Therefore, it inspires us that both of the RiBC algorithm for burst-error correction and RiBM for random error correction can be implemented one the same hardware. Furthermore, considering single burst error correcting algorithm in [20] is a specific instance of RiBC algorithm with β=0, so it can also be implemented on the RiBC architecture. Accordingly, a unified hybrid RS decoder, which can be configured to the above three types of error correcting mode, is introduced in the next subsection.

Reformulated Inversionless Burst-Error Correcting (RiBC) Algorithm

```
$f$-length ($f < 2t$ - $2\beta$) burst of errors plus maximum $\beta$ random errors ):
Input: Syndromes $S_0, S_1, S_2, ..., S_{2t-1}$;
 step1: Compute $\Xi(x) = (1-\alpha^{1-(2t-2\beta)}x)(1-\alpha^{2-(2t-2\beta)}x)...(1-\alpha^{-1}x)(1-x)$
                 $= 1 + \xi_1 x + \xi_2 x^2 + ... + \xi_{2t-2\beta} x^{2t-2\beta}$;
 step2: For $l = 0$ Step 1 Until $n-1$ do
        step2.1: Compute $\Phi(x) = \Xi(\alpha^l x) = 1 + \phi_1 x + \phi_2 x^2 + ... + \phi_{2t-2\beta} x^{2t-2\beta}$;
        step2.2: Compute $\Psi(x) = \psi_0 + \psi_1 x + ... + \psi_{2t-1} x^{2t-1}$, where $\psi_i = \sum_{j=0}^{2t-2\beta} \phi_j S_{i+2t-2\beta-j}$;
        step2.3: Initialize $\Lambda^{(0)}(x) = \lambda_0^{(0)} + \lambda_1^{(0)} x + ... + \lambda_{2t-2}^{(0)} x^{2t-2} = \Phi(x)$;
                 $B^{(0)}(x) = b_0^{(0)} + b_1^{(0)} x + ... + b_{2t-2}^{(0)} x^{2t-2} = \Phi(x)$;
                 $\tilde{\Lambda}^{(0)}(x) = \tilde{\lambda}_0^{(0)} + \tilde{\lambda}_1^{(0)} x + ... + \tilde{\lambda}_{2t-2}^{(0)} x^{2t-2} = 1$;
                 $\tilde{B}^{(0)}(x) = \tilde{b}_0^{(0)} + \tilde{b}_1^{(0)} x + ... + \tilde{b}_{2t-2}^{(0)} x^{2t-2} = 1$;
                 $\tilde{\Delta}^{(0)}(x) = \tilde{\delta}_0^{(0)} + \tilde{\delta}_1^{(0)} x + ... + \tilde{\delta}_{2t-1}^{(0)} x^{2t-1} = \Psi(x)$;
                 $\tilde{\Theta}^{(0)}(x) = \tilde{\theta}_0^{(0)} + \tilde{\theta}_1^{(0)} x + ... + \tilde{\theta}_{2t-1}^{(0)} x^{2t-1} = \Psi(x)$;
                 $\tilde{\Delta}^{*(0)}(x) = \tilde{\delta}_0^{*(0)} + \tilde{\delta}_1^{*(0)} x + ... + \tilde{\delta}_{2t-1}^{*(0)} x^{2t-1} = S(x)$;
                 $\tilde{\Theta}^{*(0)}(x) = \tilde{\theta}_0^{*(0)} + \tilde{\theta}_1^{*(0)} x + ... \tilde{\theta}_{2t-1}^{*(0)} x^{2t-1} = S(x)$;
                 $\gamma^{(0)} = 1, k^{(0)} = 0$;
        step2.4: For $r = 0$ Step 1 Until $2\beta - 1$ do
                 step2.4.1: Compute $\tilde{\delta}_i^{(r+1)} = \gamma^{(r)} \tilde{\delta}_{i+1}^{(r)} - \tilde{\delta}_0^{(r)} \tilde{\theta}_i^{(r)}$;
                                    $\tilde{\delta}_i^{*(r+1)} = \gamma^{(r)} \tilde{\delta}_{i+1}^{*(r)} - \tilde{\delta}_0^{(r)} \tilde{\theta}_i^{*(r)}$;
                                    $\lambda_i^{(r+1)} = \gamma^{(r)} \lambda_i^{(r)} - \tilde{\delta}_0^{(r)} b_{i-1}^{(r)}$;
                                    $\tilde{\lambda}_i^{(r+1)} = \gamma^{(r)} \tilde{\lambda}_i^{(r)} - \tilde{\delta}_0^{(r)} \tilde{b}_{i-1}^{(r)}$;
                 step2.4.2: If $\tilde{\delta}_0^{(r)} \neq 0$ and $k^{(r)} \geq 0$ then $a = 1$; else $a = 0$;
                 step2.4.3: $\begin{bmatrix} b_i^{(r+1)} \\ \tilde{b}_i^{(r+1)} \\ \tilde{\theta}_i^{(r+1)} \\ \tilde{\theta}_i^{*(r+1)} \\ \gamma^{(r+1)} \\ k^{(r+1)} \end{bmatrix} = \begin{bmatrix} \lambda_i^{(r)} & b_{i-1}^{(r)} \\ \tilde{\lambda}_i^{(r)} & \tilde{b}_{i-1}^{(r)} \\ \tilde{\delta}_{i+1}^{(r)} & \tilde{\theta}_i^{(r)} \\ \tilde{\delta}_{i+1}^{*(r)} & \tilde{\theta}_i^{*(r)} \\ \tilde{\delta}_0^{(r)} & \gamma^{(r)} \\ -k^{(r)} - 1 & k^{(r)} + 1 \end{bmatrix} \begin{bmatrix} a \\ \bar{a} \end{bmatrix}$
 step3: Track the longest consecutive $\tilde{\Lambda}^{(2\beta)}(x)$ that are identical, recorded the last
    element $l^*$ of the consectuive $l$'s, then the overall error locator polynoimal
    $\Lambda^*(x) = \Lambda^{(2\beta)}(x)$ at the $l^*$-th outer loop.
    The overall evaluator polynomial $\Omega^*(x)$ is corresponding $\tilde{\Delta}^{(2\beta)}(x)$ at he $l^*$-th outer loop.
Output: $\Lambda^*(x)$, $\Omega^*(x)$
```

4.2 Unified hybrid decoding (UHD) architecture

The overall UHD architecture is shown in Fig. 9. Here different blocks are used to process different steps in algorithm. Since excluding KES and PT blocks, other blocks are quite straightforward to be implemented; in this section we only introduce the architectures of KES and PT blocks and focus the discussion for the case of RiBC work mode. Interested readers can refer to [23] for the introduction of other blocks and other modes.

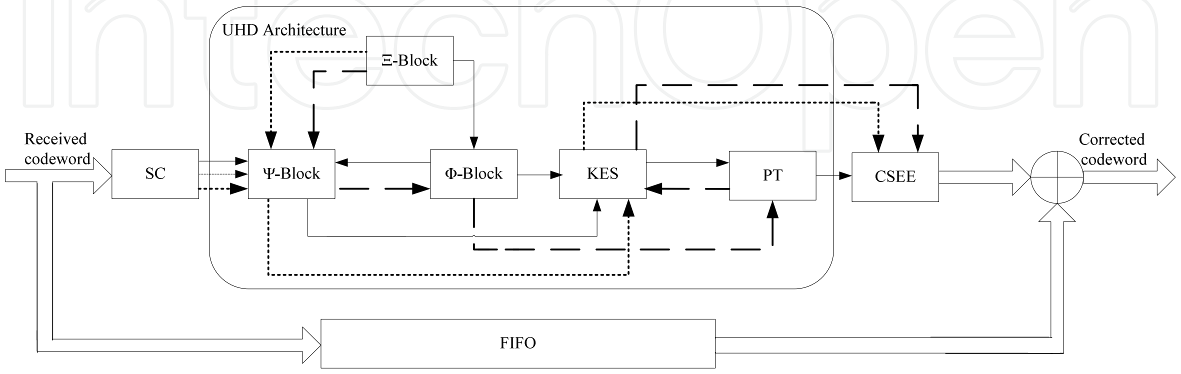


Figure 9. The overall architecture of the UHD decoder. Three types of lines illustrate data flows for different work modes: solid line (mode-1) for burst combined with random error correction RiBC algorithm, dashed line (mode-2) for only burst-error correction and dotted line (mode-3) for only random error correction.

4.2.1. KES block architecture

For RiBC algorithm, KES block is employed to carry out steps 2.4. Fig. 10 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. 10(a), the KES block consists of 2*t*-1 PE0's and 2*t* PE1's. In the *r*-th iteration, each register in PE0$_i$/PE1$_i$ stores the corresponding coefficients of different polynomials (Fig. 10(b) (c)). 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, $\tilde{\lambda}_i^{(2\beta)}$ will also be computed and outputted into PT block to track the longest consecutive $\tilde{\Lambda}^{(2\beta)}(x)$ that are identical.

4.2.2. Position track (PT) block architecture

PT block is used to track the longest consecutive polynomials that are identical (step 3). Fig. 11 illustrates the architecture of PT block. The input $\tilde{\lambda}_i^{(2\beta)}$, $\lambda_i^{(2\beta)}$ and $\tilde{\delta}_i^{(2\beta)}$ from 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 of current *longest* continuously identical $\tilde{\Lambda}^{(2\beta)}(x)$. Control signals *shift* and *equal* are generated from the signal generation schedule. After *l* reaches n, $\lambda_i(longest)$ and $\tilde{\delta}_i(longest)$ are outputted as the coefficients of overall error locator polynomial $\Lambda^*(x)$ and overall error evaluator polynomial $\Omega^*(x)$.

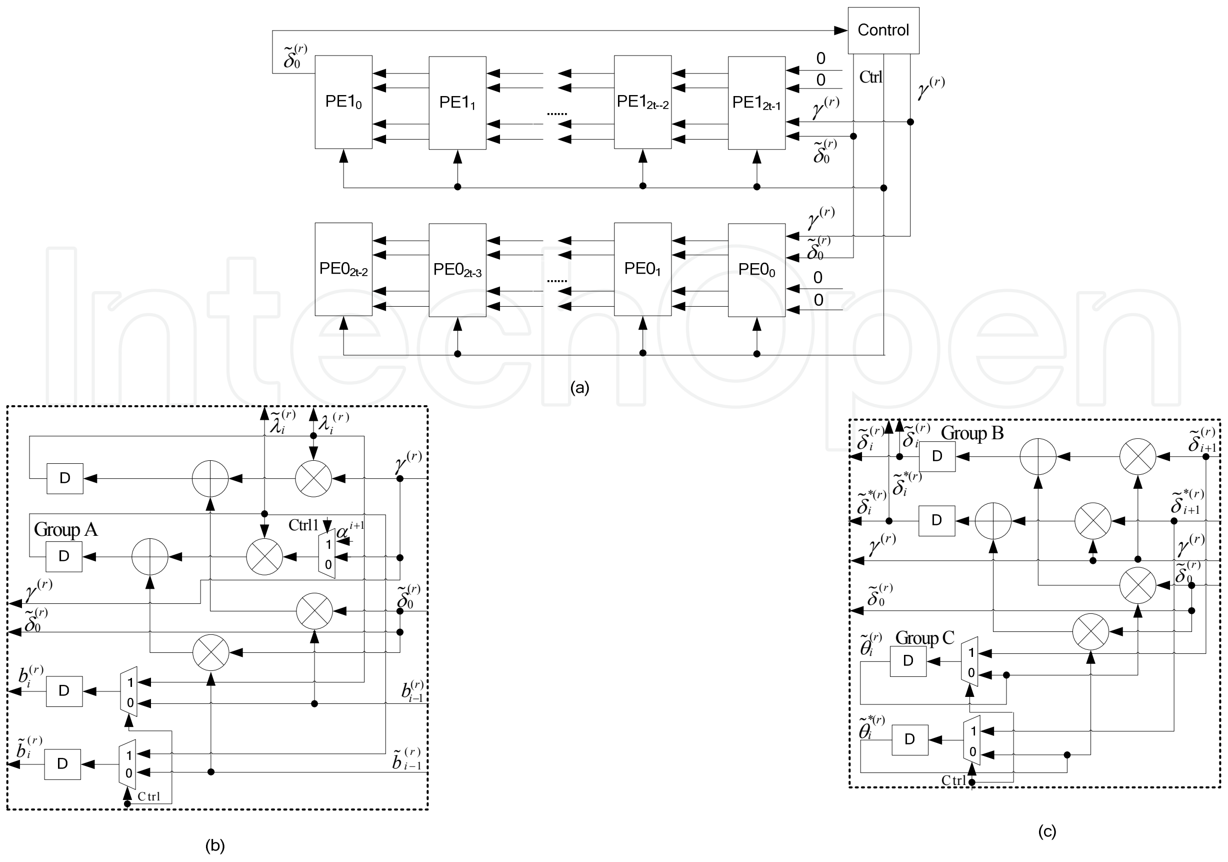


Figure 10. (a) The overall architecture of KES block. (b) The block diagram of PE0*i*. (c) The block diagram of PE1*i*.

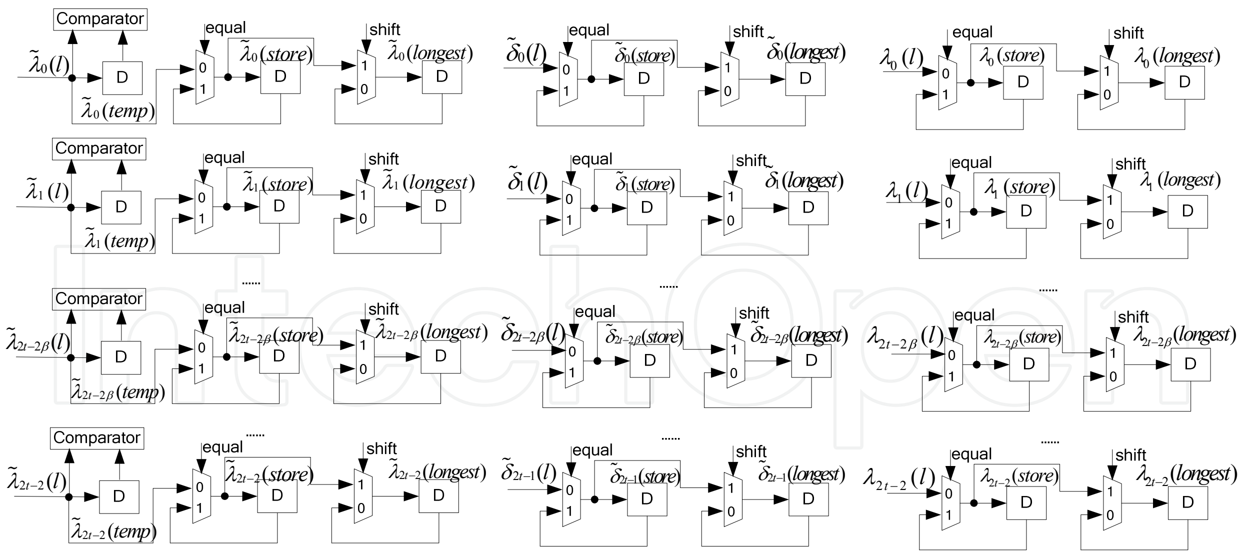

Comparator
equal
shift
D
$\tilde{\lambda}_0(l)$
$\tilde{\lambda}_0(temp)$
$\tilde{\lambda}_0(store)$
$\tilde{\lambda}_0(longest)$
$\tilde{\delta}_0(l)$
$\tilde{\delta}_0(store)$
$\tilde{\delta}_0(longest)$
$\lambda_0(l)$
$\lambda_0(store)$
$\lambda_0(longest)$
$\tilde{\lambda}_{2t-2}(l)$
$\tilde{\lambda}_{2t-2}(temp)$
$\tilde{\lambda}_{2t-2}(store)$
$\tilde{\lambda}_{2t-2}(longest)$

Figure 11. The architecture of PT block for mode-1.

Table 3 presents the comparison between UHD and RiBM decoder. Here for the example RS (255, 239) code, *n*=255, *t*=8 and *m*=8. The hardware complexity is estimated based on the work in [24]. Although the area requirement of the UHD decoder is about 1.7 times of that of the RiBM decoder, the UHD decoder can achieve significantly enhanced burst-error

```
Track Control Signal Generation Schedule for RiBC Algorithm
Initiliazation: $l_1 = 0,\ l_2 = 1$;
Step S1: Input: $\tilde{\lambda}_i^{(2\beta)}$ for current $l$, denote them as $\tilde{\lambda}_i(l)$ for $i = 0,1...2t-2\beta$;
Step S2: If $\tilde{\lambda}_i(l) = \tilde{\lambda}_i(temp)$ for all $i = 0,1...2t-2\beta$ then $equal = 1,\ l_2 = l_2 + 1$;
          else $equal = 0,\ l_2 = l_1$;
Step S3: If $l_2 > l_1$ then $shift = 1$;
          else $shift = 0$;
Step S4: If $shift = 1$ then $l_1 = l_2$;
          else $l_1$ remains;
Step S5: Output $equal,\ shift$;
Step S6: Goto Step S1
```

Architecture		**UHD**	**RiBM**
Total gates(# of XOR gates)		34308	18968
		44392 (two codewords)	
Critical path (# of gates)		10	8
(Mode-1) (*f*=11, *β*=1)	**Latency**	4846	Unavailable to decode
	Throughput (Normalized)	1	
	Miscorrection Probability	$(n\text{-}2f)(n\text{-}f)^{\beta}2^{m(\beta+f-2t)}$ $=1.32*10^{-5}$	
(Mode-2) (*f*=12)	**Latency**	542 (the worst case)	Unavailable to decode
	Throughput (Normalized)	8.9~16.8	
	Miscorrection Probability	$(n\text{-}2f)2^{m(f-2t)}$ $=5.38*10^{-8}$	
(Mode-3) ($t \le 8$, 2t+ρ≤16)	**Latency**	255	255
	Throughput (Normalized)	19	23.8
		38 (two codewords)	
	Miscorrection Probability	0	0

Table 3. Comparisons of performance on hardware and error correction capability.

correcting capability. In the channel environments that likely generate long burst of errors (*f*>8), the traditional RiBM decoder fails to decode the codewords for its limited error correcting capability, while UHD decoder can be still effective. In short, the UHD design provides an efficient and attractive unified solution for multi-mode RS decoding in optical applications that demands enhanced error correcting capability.

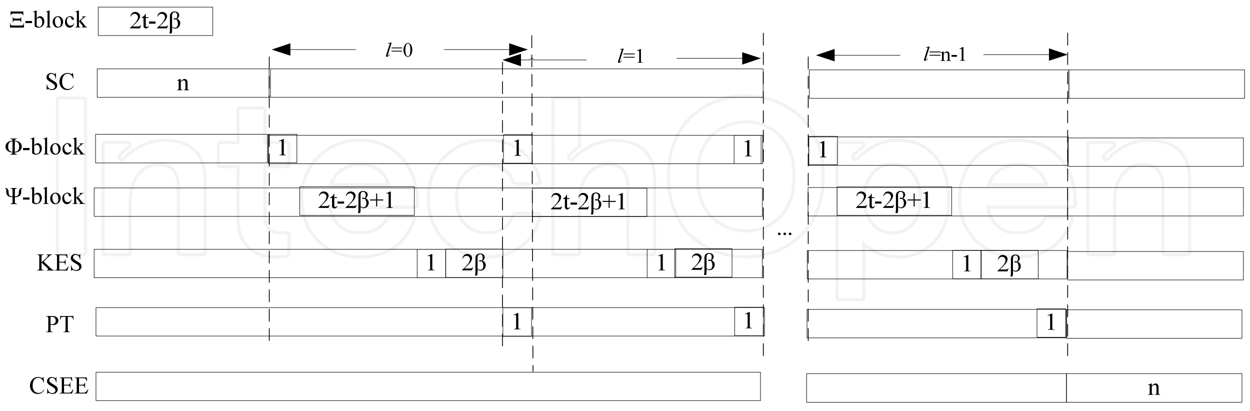


Figure 12. The timing charts for RiBC architecture.

5. Ultra high performance FEC schemes for long-haul network

For long-haul optical transport networks (OTN), because the performance loss mainly results from long distance transmission, the requirement on coding gain is very strict. This requirement even gets more and more strict when optical backbone networks enter 100Gbps era. Based on OIF whitepaper, the new FEC schemes applied in 100G long-haul systems should achieve waterfall performance at very low BER region. Meanwhile the other requirements for OTN FEC such as capable of achieving very high speed and having very low error floor still remain. Therefore, 100Gbps era puts more challenges on FEC schemes. In this section, some recent FEC schemes targeted to 100Gbps applications are introduced.

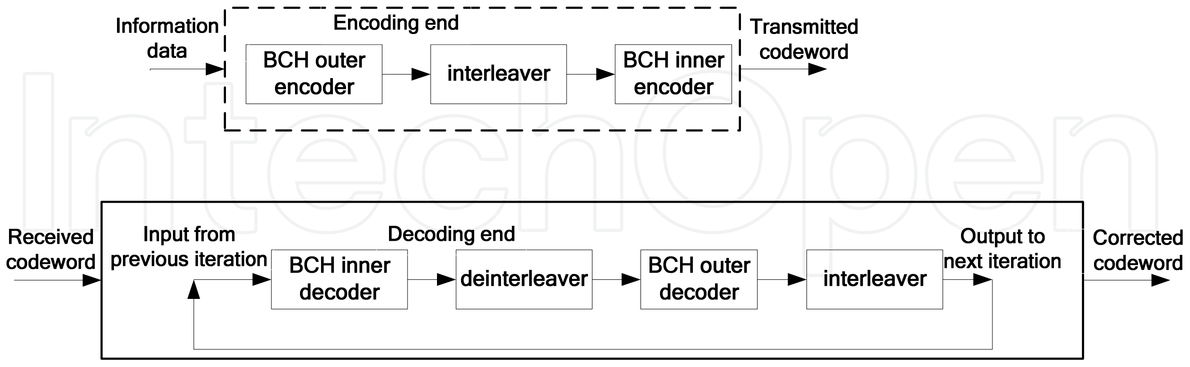


Figure 13. The hard-decision product BCH scheme.

5.1. Hard-decision BCH product codes

One candidate for 100Gbps application is BCH-based binary product codes such as the one presented in [25]. The component BCH (992, 960) and (987, 956) codes are constructed carefully over $GF(2^{10})$ for hardware amenity, which have the 3-error correction capability, therefore its decoder design can be developed based on simple PGZ algorithm in [4]. The simulation results show the performance of this FEC scheme can be very close to the Shannon limit.

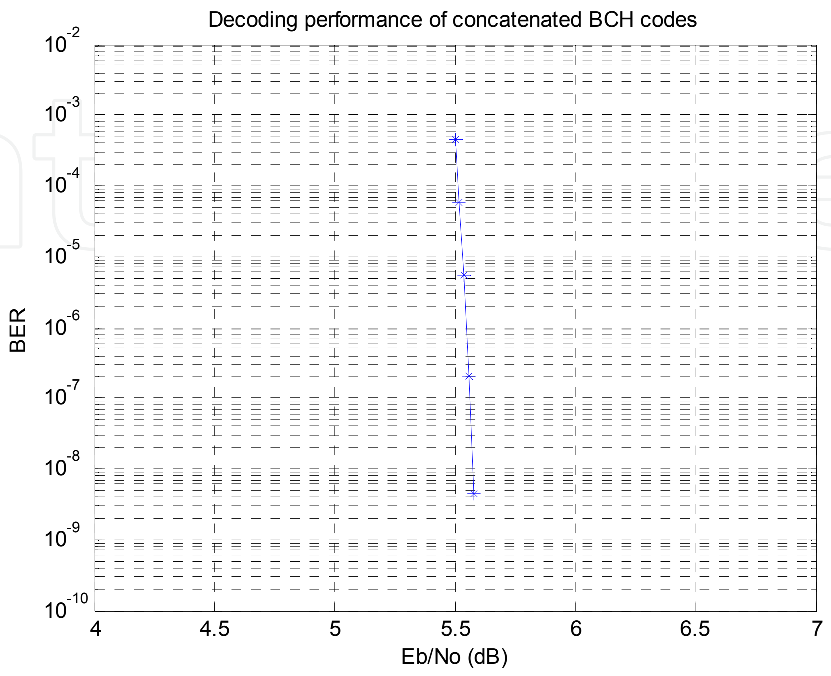


Figure 14. Decoding performance of product BCH codes based on maximum 7 iterations.

5.2. Some other soft-decision based concatenated codes

The above binary product scheme is based on hard-decision. If soft information is available in the system, soft-decision decoding approach can work with the product codes to enhance the overall decoding performance. Fig. 15 illustrates a LDPC code concatenated with BCH-based product code for long-haul network systems in [26]. In this scheme, LDPC code is used as inner code and BCH-based product code is used as outer code. Some other soft-decision based concatenated FEC scheme such as RS code concatenated with LDPC coding system in [27] can also provide significant coding gain for targeted ultra high-speed optical communication.

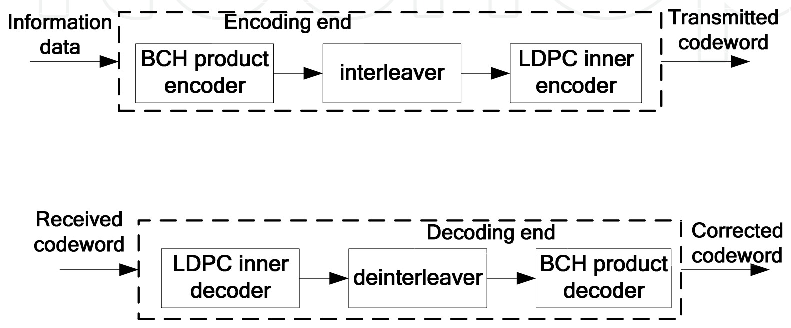


Figure 15. Product BCH-LDPC concatenated scheme in [26].

6. Conclusion

With the evolution of optical network, the employed FEC scheme has been developed in several generations. The requirement on high data rata and large coding gain is always challenging the design for efficient FEC decoder. In this chapter, targeted to different types of optical transmission networks, ranging from local Ethernet to long-haul backbone system, different FEC solutions with efficient VLSI implementations are discussed. For short-distance networks, two kinds of area-efficient high-speed RS decoders are analyzed for the scenario. For mediate distance networks, which require some tradeoff between decoding performance and hardware efficiency, the introduced RS burst-error decoder can be employed to meet such requirement. For long-haul systems, which have stringent requirement on decoding performance, some candidate FEC schemes targeted to the future 100Gbps era are discussed. In summary, these various FEC architectures and schemes are good candidates for their specific targeted optical transmission applications.

Author details

Bo Yuan
University of Minnesota-Twin Cities, USA

Li Li
Nanjing University, China

Zhongfeng Wang
Broadcom Corporation, USA

7. References

[1] 100G Forward Error Correction White Paper, OIF, OIF-FEC-100G-01.0, May. 2010.
[2] Forward Error Correction for high bit-rate DWDM Submarine System, Telecommunication Standardization Section, International Telecom Union, ITU-T G. 975.1, Feb. 2004.
[3] H. -Y. Hsu, A.-Y. Wu and J. -C. Yeo. Area-efficient VLSI design of Reed-Solomon decoder for 10Gbase-LX4 optical communication systems. IEEE Transactions on Circuits and Systems II 2006; 53(11) 1245-1249.
[4] E. R. Berlekamp. Algebraic coding theory. Newyork: McGraw-Hill; 1968.
[5] D. V. Sarwate and N. R. Shanbhag. High-speed architectures for Reed-Solomon decoders. IEEE Transactions on VLSI Systems 2001; 9(5) 641-655.
[6] H. M. Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen and I. S. Reed. A VLSI design of a pipeline Reed-Solomon decoder. IEEE Transactions on Computer 1985; 34(5) 393-403.
[7] H. Lee. A High-speed low-complexity Reed-Solomon decoder for optical communications. IEEE Transactions on Circuits and Systems II 2005; 52(8) 461-465.

[8] K. Seth, K. N. Viswajith, S. Srinivasan, and V. Kamakoti. Ultra folded high-speed architectures for Reed-Solomon decoders: proceeding of International. Conference on VLSI Design, 3-7 Jan. 2006.

[9] B. Yuan, Z. Wang, L. Li, M. Gao, J. Sha and C. Zhang. Area-efficient Reed-Solomon decoder design for optical communications. IEEE Transactions on Circuits and Systems II 2009; 56(6) 469-473.

[10] S. Lee, H. Lee, J Shin and Je-Soo Ko. A high-speed pipelined degree-computationless modified Euclidean algorithm architecture for Reed-Solomon decoders: proceeding of IEEE International. Sympousium on. Circuits and System, 27-30 May 2007.

[11] J. H. Baek and M. H. Sunwoo. New degree computationless modified Euclid's algorithm and architecture for Reed-Solomon decoder. IEEE Transactions on VLSI Systems 2006; 14(8) 915-920.

[12] I. S. Reed, M. T. Shih, and T. K. Truong. VLSI design of inverse-free Berlekamp-Massey algorithm. proceeding of IEE, part. E 1991; 138(5) 295-298.

[13] T. Horiguchi, High-speed Decoding of BCH Codes Using a New Error-evaluation Algorithm, Electronics and Communications in Japan Part 3 1989; 72(12) 63-71.

[14] Z. Yu and W. Feng. Efficient high-speed Reed-Solomon dcoder, U.S. Patent 7 322 004, Jan 22, 2008.

[15] K. K. Parhi. VLSI digital signal processing systems: Design and implementation. MA: Wiley; 1999.

[16] B. Yuan, L. Li, J. Sha, and Z. Wang. Area-efficient RS decoder design for 10-100 Gb/s applications: proceeding of IEEE International. Sympousium on. Circuits and System, 24-27 May 2009.

[17] S. Rizwan. Retimed decomposed serial Berlekamp-Massey architecture for high-speed Reed-Solomon decoding: proceeding of International Conference on VLSI Design, 4-8 Jan. 2008.

[18] M. –D. Shieh, Y. –K. Lu, S. –M. Chung, and J. –H. Chen. Design and implementation of efficient Reed-Solomon decoders for multi-mode applications: proceeding of IEEE International. Sympousium on. Circuits and System, 21-24 May 2006.

[19] H. Lee. High-speed VLSI architecture for parallel Reed-Solomon decoder. IEEE Transactions on VLSI Systems 2003; 11(2) 288-294.

[20] Y. Wu. Novel burst error correcting algorithms for Reed-Solomon codes. proceeding of IEEE Allerton Conference on Communication, Control and Computing, Sep. 30 - Oct. 2 2009.

[21] E. Dawson and A. Khodkar. Burst error-correcting algorithm for Reed-Solomon codes. Electronics Letters 1995; 31(11) 848-849.

[22] L. Yin, J. Lu, K. B. Letaief and Y. Wu. Burst-error-correcting algorithm for Reed-Solomon codes. Electronics Letters 2001; 37(11) 695-697.

[23] L. Li, B. Yuan, Z. Wang, J. Sha, H. Pan and W. Zheng. Unified Architecture for Reed-Solomon Decoder Combined with Burst-Error Correction. IEEE Transactions on VLSI Systems, to appear.

[24] X. Zhang and J. Zhu. High-throughput interpolation architecture for algebraic soft-decision Reed-Solomon decoding. IEEE Transactions on Circuits and Systems-I 2010; 57(3) 581-591.

[25] Z. Wang, C-J. Chen, K. Xiao, H. Jiang, J. R. Fife, and S. Bhoja. Communication device employing binary product coding with selective additional cyclic redundancy check(CRC) therein. U.S. Patent 12 726 062, Mar. 17, 2010.

[26] Z. Wang, B. Shen, K. Xiao and J. R. Fife. Communication device employing LDPC coding with Reed-Solomon and/or binary product coding. U.S. Patent 12 726 219, Mar. 17, 2010.

[27] T. Mizuochi, Y. Konishi, Y. Miyata, T. Inoue, K. Onohara, S. Kametani, T. Sugihara, K. Kubo, H. Yoshida, T. Kobayashi, and T. Ichikawa. Experimental demonstration of concaenated LDPC and RS codes by FPGAs emulation. IEEE photonics technology letter 2009; 21(18) 1302 – 1304.