Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A pipelined Reed-Solomon decoder based on a modified step-by-step algorithm*

Xing-ru PENG[1], Wei ZHANG[1], Yan-yan LIU‡[2]

([1]*School of Electronic and Information Engineering, Tianjin University, Tianjin 300072, China*)

([2]*College of Electronic Information and Optical Engineering, Nankai University, Tianjin 300071, China*)

E-mail: tjupengxr@tju.edu.cn; tjuzhangwei@tju.edu.cn; lyytianjin@nankai.edu.cn

Received Sept. 20, 2015; Revision accepted Feb. 16, 2016; Crosschecked Aug. 8, 2016

Abstract: We propose a pipelined Reed-Solomon (RS) decoder for an ultra-wideband system using a modified step-by-step algorithm. To reduce the complexity, the modified step-by-step algorithm merges two cases of the original algorithm. The pipelined structure allows the decoder to work at high rates with minimum delay. Consequently, for RS(23,17) codes, the proposed architecture requires 42.5% and 24.4% less area compared with a modified Euclidean architecture and a pipelined degree-computationless modified Euclidean architecture, respectively. The area of the proposed decoder is 11.3% less than that of the previous step-by-step decoder with a lower critical path delay.

Key words: Reed-Solomon codes, Step-by-step algorithm, Ultra-wideband, Pipelined structure
http://dx.doi.org/10.1631/FITEE.1500303 **CLC number:** TN79

1 Introduction

Reed-Solomon (RS) code is one of the most frequently used forward-error-correcting codes. Because of the excellent performance in correcting burst errors and random errors, RS codes are widely used in many digital storage and communication systems, such as the vestigial sideband system, satellite, mobile communications, and multiband orthogonal frequency division multiplexing (MB-OFDM) ultra-wideband (UWB) systems (Batra *et al.*, 2004; Das *et al.*, 2013). For UWB systems, RS(23,17) codes, which are shorted version of RS(255,249) codes, are adopted to protect the important header information. The most well-known decoding methods for RS codes are based on the Berlekamp-Massey (BM) algorithm (Berlekamp, 1968; Sarwate and Shanbhag, 2001; Wu, 2015) and the modified Euclidean (ME) algorithm (Lee, 2003; Baek and Sunwoo, 2006; Guo and Gai, 2014). These decoding methods share three main steps: syndrome calculator (SC), key equation solver (KES), and Chien search and error evaluator (CSEE).

Massey (1965) presented a totally different decoding algorithm, which was known as a step-by-step (SBS) algorithm. Since a great number of iterations need to be performed on each symbol, the conventional step-by-step algorithm is very complex. Using the properties of temporarily changed syndrome matrices, a new step-by-step algorithm for RS codes was presented (Chen *et al.*, 2000). The algorithm can directly determine whether the received symbol is erroneous or not and immediately find the corresponding error value without solving the key equations. Based on this algorithm, pipelined structures were presented (Chen *et al.*, 2003; Chen and Tasi, 2007). However, determinants need to be calculated for each symbol, which leads to high computational and hardware complexity. Using a new method to

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61474080) and the Program for New Century Excellent Talents in University, China

ORCID: Yan-yan LIU, http://orcid.org/0000-0001-8488-5480

calculate the determinants of temporarily changed syndrome matrices, a simplified parallel step-by-step decoding algorithm was proposed (Liu *et al.*, 2007), which significantly reduces the computational complexity.

The step-by-step algorithm is especially suitable for RS codes with small error-correction capacity, such as the RS(23,17) codes described above, whose error-correction capacity is three. However, Chen *et al.* (2000) and Liu *et al.* (2007) took two different ways to decode when the error number is equal to t or not, which involve a lot of redundant operations. Hence, the complexity of the step-by-step algorithm can be further reduced by adopting a unified strategy to decode those two situations. There are two main contributions in this study. First, a modified step-by-step algorithm is proposed, which merges the two cases and hence reduces the complexity significantly. Second, since the SC block takes a long time, rather than reducing the latency efficiently, the parallel algorithm (Liu *et al.*, 2007) increases the hardware requirements. In this paper, a pipelined architecture is proposed. This architecture does not introduce extra latency while reducing hardware complexity, which increases the hardware utilization efficiency significantly.

2 Original step-by-step decoding algorithm

For RS(n,k) codes defined in the Galois field GF(2^m), t is the error-correcting capacity which satisfies $n-k=2t$. The $2t$ syndromes are defined as S_i (i=1, 2, ..., $2t$), and a $k\times k$ syndrome matrix $\boldsymbol{L}_k$ is established as

$$\boldsymbol{L}_k=\begin{bmatrix} S_1 & S_2 & \cdots & S_k \\ S_2 & S_3 & \cdots & S_{k+1} \\ \vdots & \vdots & & \vdots \\ S_k & S_{k+1} & \cdots & S_{2k-1} \end{bmatrix}. \tag{1}$$

Suppose the error number is v. If $v<k$, then $\det(\boldsymbol{L}_k)=0$; otherwise, $\det(\boldsymbol{L}_k)\neq 0$ ($\det(\cdot)$ is the determinant of the syndrome matrix).

By adding a nonzero element β to the jth ($j=0,\ 1,\ \ldots,\ n$) symbol of the received codeword, new syndromes are obtained, defined as $S_i(\beta,j)$ and $S_i(\beta,j)=S_i+\beta\cdot\alpha^{ij}$, where α is the primitive element of the Galois field. Using the new syndromes, the temporarily changed syndrome matrix $\boldsymbol{L}_k(\beta,j)$ is established and its determinant is calculated as

$$\det(\boldsymbol{L}_k(\beta,j))=\det(\boldsymbol{L}_k)+\beta\sum_{x=1}^{v}\alpha^{(2x-1)j}\det(\boldsymbol{L}_k^{xx}), \tag{2}$$

where $\boldsymbol{L}_k^{xx}$ is the submatrix of $\boldsymbol{L}_k$ obtained by deleting the xth row and xth column of $\boldsymbol{L}_k$.

If the error number is v ($0<v\leq t$), define $H_{v,j}$ and $H_{v+1,j}$ as

$$\begin{cases} H_{v,j}=\sum\limits_{x=1}^{v}\alpha^{(2x-1)j}\det(\boldsymbol{L}_v^{xx}), \\ H_{v+1,j}=\sum\limits_{x=1}^{v+1}\alpha^{(2x-1)j}\det(\boldsymbol{L}_{v+1}^{xx}). \end{cases} \tag{3}$$

If $H_{v,j}=0$, the jth symbol must be correct. However, if $H_{v,j}\neq 0$, we cannot determine whether the symbol is correct or not. However, if $H_{v+1,j}\neq 0$, the jth symbol must be correct; otherwise, the jth symbol must be wrong. Supposing the jth symbol is wrong, the error value is calculated by $\beta=\det(\boldsymbol{L}_v)/H_{v,j}$, which satisfies $\det(\boldsymbol{L}_v(\beta,j))=0$. In the situation where $v<t$, $H_{v+1,j}$ is calculated to determine whether the jth symbol is erroneous. Nevertheless, if $v=t$, $H_{t+1,j}$ cannot be calculated directly, since S_{2t+1} remains unknown. In this case, $H_{t,j}$ is calculated first. If $H_{t,j}=0$, the jth symbol must be correct; otherwise, $\det(\boldsymbol{L}_{t+1})$ and β are calculated. By adding β to the jth symbol, the error number may reduce to $t-1$ or increase to $t+1$. Hence, the value of $\det(\boldsymbol{L}_{t+1}(\beta,j))$ can be used to determine the error number. If $\det(\boldsymbol{L}_{t+1}(\beta,j))=0$, it means that the number of errors reduces to $t-1$ and thus the jth symbol is erroneous and the error value is β. Otherwise, the jth symbol is correct. Since $\det(\boldsymbol{L}_t(\beta,j))=0$, the value of $\det(\boldsymbol{L}_{t+1}(\beta,j))$ is independent of S_{2t+1}. Therefore, the value of S_{2t+1} can be set to any symbol in GF(2^m). The flowchart of the step-by-step algorithm is illustrated in Fig. 1.

3 Modified step-by-step algorithm

To make the parallel algorithm (Liu *et al.*, 2007) suitable for a pipelined structure, some modifications are required. The value of $H_{v,j}$ is calculated by Eq. (3). Define $\alpha^{(2x-1)j}\det(\boldsymbol{L}_{v,0}^{xx})=\det(\boldsymbol{L}_{v,j}^{xx})$. Thus, $\det(\boldsymbol{L}_{v,j}^{xx})=\alpha^{2x-1}\det(\boldsymbol{L}_{v,j-1}^{xx})$. Consequently, $H_{v,j}$ can be calculated as $H_{v,j}=\sum\limits_{x=1}^{v}\det(\boldsymbol{L}_{v,j}^{xx})=\sum\limits_{x=1}^{v}\alpha^{2x-1}\det(\boldsymbol{L}_{v,j-1}^{xx})$, which implies that the

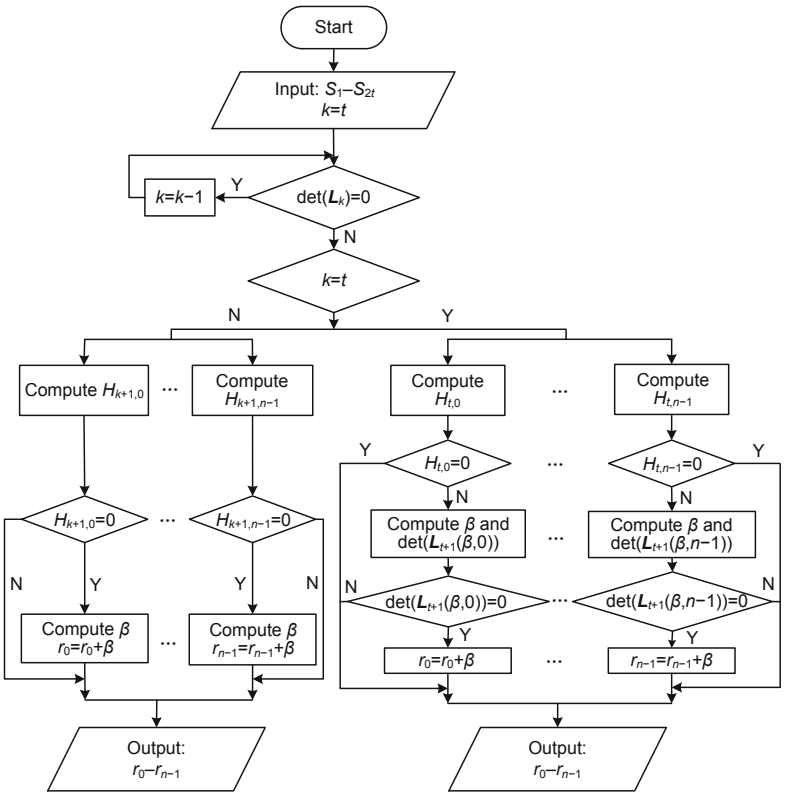

Fig. 1 Flowchart of the step-by-step algorithm

calculation for the jth symbol can be obtained from the foregoing symbol.

The previous step-by-step algorithms take two different ways to decode when the error number is equal to t or not. Therefore, two different blocks are needed for the decoder. However, when decoding each received codeword, only one block is working while the other one is idle, which results in high hardware complexity. Consequently, a unified strategy is proposed as follows to decode both situations. For t-error correcting codes, suppose the error number is v ($0 \le v \le t$). For the jth symbol, $H_{v,j}$ and $H_{v+1,j}$ are calculated first. As described above, the nonexistent S_{2t+1}, which may be needed to calculate $H_{v+1,j}$, can be set to 0. Therefore, alternative-error-value β and $\det(\boldsymbol{L}_{v+1}(\beta,j))$ can be calculated, regardless of whether v equals t or not. If $\det(\boldsymbol{L}_{v+1}(\beta,j))=0$, the jth symbol is erroneous and the error value is β; otherwise, the jth symbol is correct.

Based on the properties described above, a modified step-by-step algorithm is proposed. First, the syndrome values S_i ($i=1,\ 2,\ \ldots,\ 2t$) are calculated. Second, $\det(\boldsymbol{L}_k)$ ($k=0,\ 1,\ \ldots,\ t+1$) and $\det(\boldsymbol{L}_k^{xx})$ ($k=0,\ 1,\ \ldots,\ t+1;\ x=0,\ 1,\ \ldots,\ k$) are calculated. Since the value of S_{2t+1} is set to 0, the calculation associated with S_{2t+1} can be ignored. Then $\det(\boldsymbol{L}_k)$ ($k=0,\ 1,\ \ldots,\ t$) are consecutively tested to determine the error number v. Once v is determined, $\det(\boldsymbol{L}_v), \det(\boldsymbol{L}_{v+1}), \det(\boldsymbol{L}_{v,0}^{xx})$ ($x=0,\ 1,\ \ldots,\ t$), and $\det(\boldsymbol{L}_{v+1,0}^{xx})$ ($x=0,\ 1,\ \ldots,\ t+1$) are selected and saved. Third, from r_0 to r_{n-1}, $H_{v,j}$ and $H_{v+1,j}$ are calculated by summing $\det(\boldsymbol{L}_{v,j}^{xx})$ and $\det(\boldsymbol{L}_{v+1,j}^{xx})$, respectively. Thereafter, each $\det(\boldsymbol{L}_{v,j}^{xx})$ and $\det(\boldsymbol{L}_{v+1,j}^{xx})$ are multiplied by α^{2x-1} to update for the calculation of the next symbol. After the calculation of $H_{v,j}$, an alternative-error-value is calculated by $\beta=\det(\boldsymbol{L}_v)/H_{v,j}$. Finally, $\det(\boldsymbol{L}_{v+1}(\beta,j))$ is calculated to determine the error value. The modified step-by-step algorithm is presented in Algorithm 1 and the flowchart is shown in Fig. 2.

4 Pipelined step-by-step decoding architecture for UWB systems

Since the error-correction capacity is small, the RS(23,17) codes adopted by the MB-OFDM

Algorithm 1 Modified step-by-step algorithm

1: Calculate the syndrome values by $S_i = r(\alpha^i)$ $(i = 1,\ 2,\ \ldots,\ 2t)$, and simply set $S_{2t+1} = 0$.

2: Calculate $\det(\boldsymbol{L}_k)$ and $\det(\boldsymbol{L}_k^{xx})$ $(k = 0,\ 1,\ \ldots,\ t+1;\ x = 0,\ 1,\ \ldots,\ k)$. Then determine the error number v. Thereafter, select and save $\det(\boldsymbol{L}_v), \det(\boldsymbol{L}_{v+1}), \det(\boldsymbol{L}_{v,0}^{xx})$ $(x = 0,\ 1,\ \ldots,\ t)$, and $\det(\boldsymbol{L}_{v+1,0}^{xx})$ $(x = 0,\ 1,\ \ldots,\ t+1)$.

3: For each symbol r_j $(j = 0,\ 1,\ \ldots,\ n)$

3.1: Calculate

$$H_{v,j} = \sum_{x=1}^{t} \det(\boldsymbol{L}_{v,j}^{xx});$$

$$H_{v+1,j} = \sum_{x=1}^{t+1} \det(\boldsymbol{L}_{v+1,j}^{xx});$$

$$\beta = \det(\boldsymbol{L}_v)/H_{v,j};$$

$$\det(\boldsymbol{L}_{v,j+1}^{xx}) = \alpha^{(2x-1)} \cdot \det(\boldsymbol{L}_{v,j}^{xx})\quad (x = 1,\ 2,\ \ldots,\ t);$$

$$\det(\boldsymbol{L}_{v+1,j+1}^{xx}) = \alpha^{(2x-1)} \cdot \det(\boldsymbol{L}_{v+1,j}^{xx})\quad (x = 1,\ 2,\ \ldots,\ t+1).$$

3.2: Calculate $\det(\boldsymbol{L}_{v+1}(\beta, j))$ by $\det(\boldsymbol{L}_{v+1}(\beta, j)) = \det(\boldsymbol{L}_{v+1}) + \beta \cdot H_{v+1,j}$. If $\det(\boldsymbol{L}_{v+1}(\beta, j)) = 0$, then $r'_j = r_j + \beta$; otherwise, $r'_j = r_j$.

4: Finish

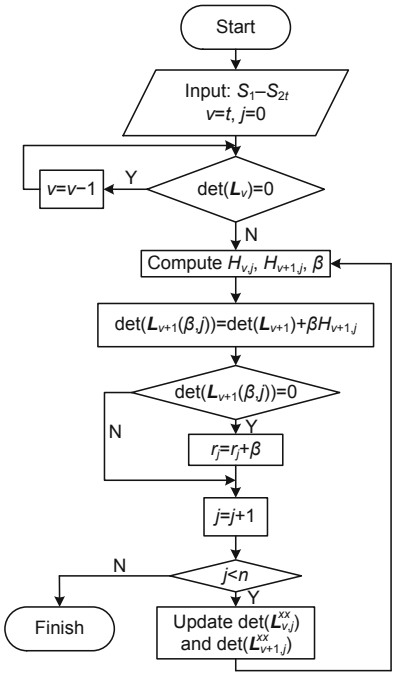

Fig. 2 Flowchart of the modified step-by-step algorithm

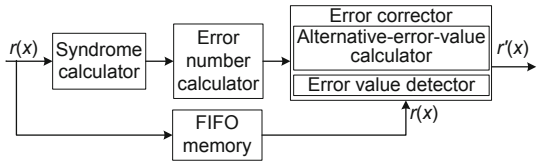

Fig. 3 Block diagram of the pipelined step-by-step decoder

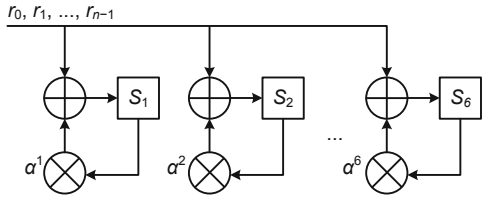

Fig. 4 Syndrome calculation block

UWB system are quite suitable for the step-by-step decoding algorithm. Based on the modified step-by-step algorithm above, a pipelined decoder architecture is proposed in this section. Fig. 3 shows the block diagram of the pipelined decoder, which comprises an SC, an error number calculator (ENC), and an error corrector (EC). The first-input first-output (FIFO) memory is used to store the received codeword.

As shown in Fig. 4, the SC block is used to generate the $2t$=6 syndromes. It takes n=23 clock cycles to calculate the syndromes. Shorted from the RS(255,249) codes, the symbols as well as the arithmetic of RS(23,17) codes are defined in GF(2^8). Consequently, the multipliers used in the SC block are constant multipliers in GF(2^8), which are much simpler than regular multipliers.

The ENC block is used to calculate the determinant of syndrome matrices and determine the error number, whose functional block diagram is given in Fig. 5. The determinant calculator is shown in Fig. 6 with critical path delay $T_{\text{mult}} + 2T_{\text{add}}$, where T_{mult} and T_{add} are delays of the multiplier and adder, respectively.

Since t=3 for RS(23,17), the maximum dimension of the syndrome matrices is 4, whose determinants can be calculated directly. As S_7 is set to 0, the calculation associated with S_7 can be ignored. Moreover, using the intermediate values of $\det(\boldsymbol{L}_i)$, the complexity of calculating $\det(\boldsymbol{L}_i^{xx})$ can be significantly reduced. For instance, $S_1 \cdot S_5$ has been calculated during calculating $\det(\boldsymbol{L}_4)$ and $S_3 \cdot S_3$ has been calculated during calculating $\det(\boldsymbol{L}_3)$. Hence, the calculation of $\det(\boldsymbol{L}_3^{22}) = S_1 \cdot S_5 + S_3 \cdot S_3$ needs only one adder. After that, $\det(\boldsymbol{L}_k)$ are tested to

determine the error number. When the error number v is determined, the ENC block also selects and saves $\det(\boldsymbol{L}_v), \det(\boldsymbol{L}_{v+1}), \det(\boldsymbol{L}^{xx}_{v,0})$ $(x=0,\ 1,\ \ldots,\ t)$, and $\det(\boldsymbol{L}^{xx}_{v+1,0})$ $(x=0,\ 1,\ \ldots,\ t+1)$. Moreover, if $x>v$ or $x>v+1$, $\det(\boldsymbol{L}^{xx}_{v,0})$ or $\det(\boldsymbol{L}^{xx}_{v+1,0})$ is set to 0.

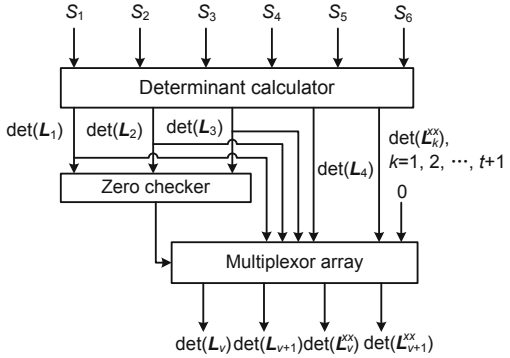

Fig. 5 Block diagram of the error number calculator

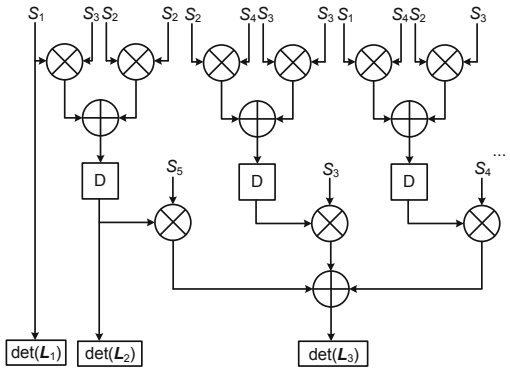

Fig. 6 Determinant calculator

Thereafter, a symbol-by-symbol error correction method is used. The EC block consists of an alternative-error-value calculator (AEVC) and an error-value detector (EVD) block. The AEVC block computes the alternative-error-value β and $H_{v+1,j}$ simultaneously. As shown in Fig. 7a, during each clock cycle, $\det(\boldsymbol{L}^{xx}_{v,j})$ $(x=1,\ 2,\ \ldots,\ t)$ are summed up to calculate $H_{v,j}$ and multiplied by α^{2x-1} to calculate $\det(\boldsymbol{L}^{xx}_{v,j+1})$ $(x=1,\ 2,\ \ldots,\ t)$. After that, β is calculated by $\beta=\det(\boldsymbol{L}_v)/H_{v,j}$. A similar architecture for calculating H_{v+1} is given in Fig. 7b. The critical path delay of AEVC block is T_{mult}.

After obtaining the values of β and $H_{v+1,j}$, the EVD block computes $\det(\boldsymbol{L}_{v+1}(\beta,j))$ by $\det(\boldsymbol{L}_{v+1}(\beta,j)) = \det(\boldsymbol{L}_{v+1}) + \beta\cdot H_{v+1,j}$. According to the modified step-by-step algorithm, if $\det(\boldsymbol{L}_{v+1}(\beta,j))=0$, the error value equals β; otherwise, the error value equals 0. When the error value is selected by a multiplexor, error correction is carried out afterwards. The architecture of the EVD block is shown in Fig. 8 and its critical path delay is $T_{\mathrm{EVD}}=T_{\mathrm{mult}}+T_{\mathrm{add}}$.

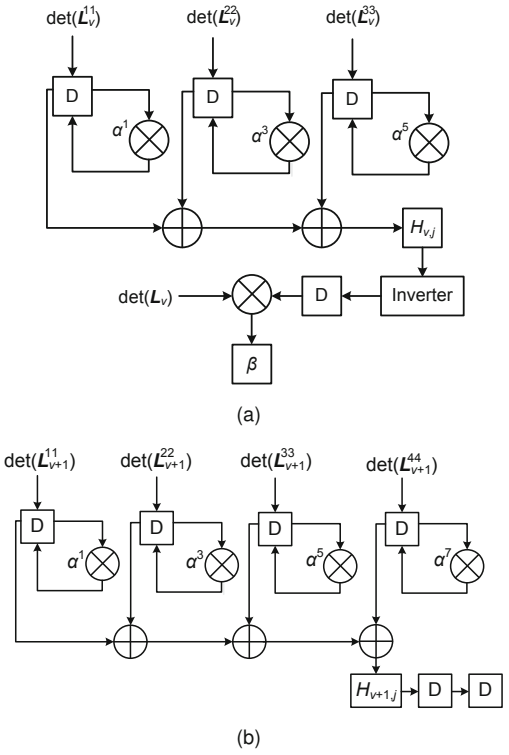

Fig. 7 Architecture of alternative-error-value calculator (a) and $\boldsymbol{H_{v+1}}$ calculator (b)

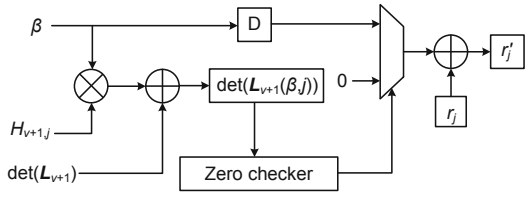

Fig. 8 The error-value detector

5 Analyses

5.1 Hardware and latency

The hardware requirements without the FIFO memory for each part of the proposed decoder are illustrated in Table 1. Since the hardware complexities of the ME architecture (Lee, 2003), pipelined

degree-computationless modified Euclidean (pDCME) architecture (Lee and Lee, 2008), and the previous step-by-step decoder (Chen *et al.*, 2003; Chen and Tasi, 2007) are related to t, the corresponding hardware requirements for RS(23,17) codes can be easily obtained. Table 2 shows the hardware comparisons between the proposed pipelined architecture and the existing RS decoders. In GF(2^8), by applying the composite field arithmetic, a regular multiplier can be implemented by 64 XOR gates and 48 AND gates while an inverter consists of 121 XOR gates and 36 AND gates. The gate number of a constant multiplier is determined by the constant multiplicand, and its average area is 10 times that of an XOR. Each AND or OR gate requires 3/4 of the area of an XOR, and each 2-to-1 multiplexor (Mux) or memory cell has an area equal to that of an XOR. Moreover, the area of each register (Reg) is about three times that of an XOR (Zhu *et al.*, 2009; Zhang and Zhu, 2010; García-Herrero *et al.*, 2011). As shown in Table 2, for RS(23,17) codes, the proposed architecture involves $(10\,090-5798)/10\,090 = 42.5\%$ and $(7674-5798)/7674 = 24.4\%$ less area than the ME architecture and the pDCME architecture, respectively. Moreover, the decoders (Chen *et al.*, 2003; Chen and Tasi, 2007) contain three determinant calculators, which need more multipliers and adders. Consequently, the area of the proposed decoder is $(6536-5798)/6536=11.3\%$ less than that of the previous step-by-step decoder.

The critical path delay in the proposed decoder is $T_{\text{mult}}+2T_{\text{add}}$, which exists in the determinant calculator. However, the critical path delay of the previous step-by-step decoder is as large as $T_{\text{mult}}+3T_{\text{add}}$, which exists during calculating $\det(N'^{j}_{t+1})$. As described above, the latency of the SC block is $n = 23$ clock cycles. The ENC and EC blocks take 4 clock cycles and $n + 5 = 28$ clock cycles, respectively. Hence, the decoder can be divided into two pipelining stages: the SC block and ENC block are in one pipelining stage, while the EC block is in another pipelining stage. The decoding latency is determined by the longest pipelining stage latency. Therefore, the latency of the proposed decoder is 28 clock cycles. In addition, the decoding timing schedule is shown in Fig. 9.

To make the advantage of the proposed architecture more obvious, hardware requirements of decoders for RS(255,251) codes are also considered, whose critical path delay is $T_{\text{mult}}+T_{\text{add}}$. As shown in Table 3, the number of the required XOR gates of the proposed architecture is 61.3%, 49.1%, or 9.97% less than that of ME, pDCME, or the previous SBS architecture, respectively.

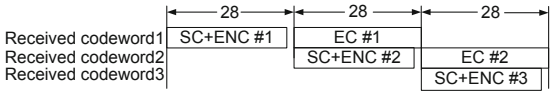

Fig. 9 Timing schedule of the proposed decoder

Table 1 Hardware requirement for the proposed decoder

Component	Number of multipliers	Number of adders	Number of multiplexors	Number of registers	Number of constant multipliers	Number of invertors
SC	0	48	0	48	6	0
ENC	36	184	152	264	0	0
EC	2	56	8	112	7	1
Total	38	288	160	424	13	1

SC: syndrome calculator; ENC: error number calculator; EC: error corrector

Table 2 Hardware requirement and comparisons of decoders for RS(23,17) codes

Architecture	Number of multipliers	Number of adders	Number of multiplexors	Number of registers	Number of constant multipliers	Number of invertors	Total number of XORs
ME*	25	280	984	2016	13	1	10 090
pDCME**	25	184	488	1408	13	1	7674
Previous SBS***	45	352	96	400	24	1	6536
Proposed	38	288	160	424	13	1	5798

* Lee (2003). ** Lee and Lee (2008). *** Chen *et al.* (2003); Chen and Tasi (2007). ME: modified Euclidean; pDCME: pipelined degree-computationless modified Euclidean; SBS: step-by-step

Table 3 Hardware requirement and comparisons of decoders for RS(255,251) codes

Architecture	Number of multipliers	Number of adders	Number of multiplexors	Number of registers	Number of constant multipliers	Number of invertors	Total number of XORs
ME*	17	184	664	1360	9	1	6866
pDCME**	17	120	328	944	9	1	5218
Previous SBS***	18	144	48	216	16	1	2948
Proposed	14	120	104	264	9	1	2654

* Lee (2003). ** Lee and Lee (2008). *** Chen *et al.* (2003); Chen and Tasi (2007). ME: modified Euclidean; pDCME: pipelined degree-computationless modified Euclidean; SBS: step-by-step

5.2 Computational complexity and performance

The previous pipelined step-by-step architectures (Chen *et al.*, 2003; Chen and Tasi, 2007) need to calculate $\det(\boldsymbol{N}_k^0)$, whose computational complexity is $\sum_{i=1}^{t} O(i)$ ($O(i)$ is the number of operations required to calculate the determinant of an $i \times i$ matrix). Moreover, $\det(\boldsymbol{M}_k^j)$ and $\det(\boldsymbol{N}_k^{'j})$ must be calculated for each symbol, and the computational complexity is $n \cdot \sum_{i=1}^{t-1} O(i) + n \cdot \sum_{i=1}^{t+1} O(i) = n \cdot O(t+1) + n \cdot O(t) + 2n \cdot \sum_{i=1}^{t-1} O(i)$. Therefore, the total computational complexity of the previous architectures is $n \cdot O(t+1) + (n+1) \cdot O(t) + (2n+1) \cdot \sum_{i=1}^{t-1} O(i)$. In the proposed architecture, the determinants of syndrome matrices are calculated and saved first instead of being calculated for each symbol. Thus, the computational complexity of the proposed architecture is reduced to $O(t+1) + (t+2) \cdot O(t) + (t+1) \cdot O(t-1) + \sum_{i=1}^{t-2} O(i)$. Since $n \gg t$, the complexity of computation can be greatly reduced.

Specifically, for the RS(23,17) codes, the computational complexity of the previous architecture is $23 \cdot O(4) + 24 \cdot O(3) + 47 \cdot O(2) + 47 \cdot O(1)$ while that of the proposed architecture is only $O(4) + 5 \cdot O(3) + 4 \cdot O(2) + O(1)$.

Moreover, as described in Section 3, the proposed algorithm is more suitable for hardware implementation than the simplified step-by-step algorithm (Liu *et al.*, 2007). Although the computational complexities of these two algorithms are the same, the decoding process of the proposed algorithm is much simpler, since a unified decoding strategy is adopted. In addition, the parallel algorithm is modified to a recursive algorithm, which is more suitable for pipelined structure; hence, the hardware utilization is significantly improved.

Bit error rate (BER) is an indicator to measure the performance of decoding algorithms. BM, ME, and SBS algorithms are hard-decision decoding (HDD) algorithms, which can correctly decode codes with fewer than t errors; hence, they have the same decoding performance. Simulations are also carried out for short RS codes and long RS codes using ME, SBS, and the modified SBS algorithms. Simulation results are shown in Fig. 10. It can be observed that the modified SBS algorithm can always achieve the same performance as the ME and SBS algorithms. That is to say, the modified SBS algorithm can significantly improve the hardware utilization without performance degradation.

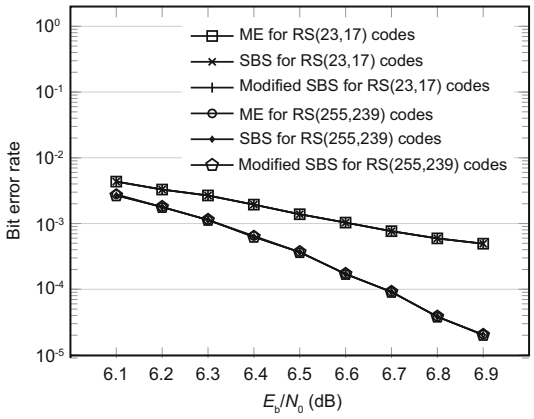

Fig. 10 Simulation results for RS codes decoding schemes

6 Conclusions

A modified step-by-step algorithm and a novel pipelined decoder for RS(23,17) codes are proposed in this paper. With modification, the computational complexity is significantly reduced.

In addition, much less area is needed for this decoder compared with the ME architecture and the pDCME architecture. As a result, the low computational and hardware complexities make the proposed decoder suitable for the UWB system.

References

Baek, J.H., Sunwoo, M.H., 2006. New degree computationless modified Euclid algorithm and architecture for Reed-Solomon decoder. *IEEE Trans. VLSI Syst.*, **14**(8):915-920. http://dx.doi.org/10.1109/TVLSI.2006.878484

Batra, A., Balakrishnan, J., Dabak, A., *et al.*, 2004. Multi-band OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a. IEEE P802.15-03/268r2.

Berlekamp, E.R., 1968. Algebraic Coding Theory. McGraw-Hill, New York.

Chen, T.C., Tasi, M.H., 2007. Hardware implementation of a high-speed (32, 24, 4) RS decoder. *Chung Hua J. Sci. Eng.*, **5**(4):21-27.

Chen, T.C., Wei, C.H., Wei, S.W., 2000. Step-by-step decoding algorithm for Reed-Solomon codes. *IEE Proc. Commun.*, **147**(1):8-12. http://dx.doi.org/10.1049/ip-com:20000149

Chen, T.C., Wei, C.H., Wei, S.W., 2003. A pipeline structure for high-speed step-by-step RS decoding. *IEICE Trans. Commun.*, **E86-B**(2):847-849.

Das, A.S., Das, S., Bhaumik, J., 2013. Design of RS(255,251) encoder and decoder in FPGA. *Int. J. Soft Comput. Eng.*, **2**(6):391-394.

García-Herrero, F., Valls, J., Meher, P.K., 2011. High-speed RS(255, 239) decoder based on LCC decoding. *Circ. Syst. Signal Process.*, **30**(6):1643-1669. http://dx.doi.org/10.1007/s00034-011-9327-4

Guo, W., Gai, W., 2014. Area-efficient recursive degree computationless modified Euclid's architecture for Reed-Solomon decoder. Proc. IEEE Int. Conf. on Electron Devices and Solid-State Circuits, p.1-2. http://dx.doi.org/10.1109/EDSSC.2014.7061134

Lee, H., 2003. High-speed VLSI architecture for parallel Reed-Solomon decoder. *IEEE Trans. VLSI Syst.*, **11**(2):288-294. http://dx.doi.org/10.1109/TVLSI.2003.810782

Lee, S., Lee, H., 2008. A high-speed pipelined degree-computationless modified Euclidean algorithm architecture for Reed-Solomon decoders. *IEICE Trans. Fundament. Electron. Commun. Comput. Sci.*, **E91-A**(3):830-835.

Liu, X., Lu, C., Cheng, T.H., *et al.*, 2007. A simplified step-by-step decoding algorithm for parallel decoding of Reed-Solomon codes. *IEEE Trans. Commun.*, **55**(6):1103-1109. http://dx.doi.org/10.1109/TCOMM.2007.898703

Massey, J., 1965. Step-by-step decoding of the Bose-Chaudhuri-Hocquenghem codes. *IEEE Trans. Inform. Theory*, **11**(4):580-585. http://dx.doi.org/10.1109/TIT.1965.1053833

Sarwate, D.V., Shanbhag, N.R., 2001. High-speed architectures for Reed-Solomon decoders. *IEEE Trans. VLSI Syst.*, **9**(5):641-655. http://dx.doi.org/10.1109/92.953498

Wu, Y., 2015. New scalable decoder architectures for Reed-Solomon codes. *IEEE Trans. Commun.*, **63**(8):2741-2761. http://dx.doi.org/10.1109/TCOMM.2015.2445759

Zhang, X., Zhu, J., 2010. High-throughput interpolation architecture for algebraic soft-decision Reed-Solomon decoding. *IEEE Trans. Circ. Syst. I*, **57**(3):581-591. http://dx.doi.org/10.1109/TCSI.2009.2023935

Zhu, J., Zhang, X., Wang, Z., 2009. Backward interpolation architecture for algebraic soft-decision Reed-Solomon decoding. *IEEE Trans. VLSI Syst.*, **17**(11):1602-1615. http://dx.doi.org/10.1109/TVLSI.2008.2005575