Received: 20 December 2019 | Revised: 15 May 2020 | Accepted: 20 October 2020

IET Circuits, Devices & Systems

DOI: 10.1049/cds2.12026

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH PAPER

Reduced complexity hard- and soft-input BCH decoding with applications in concatenated codes

Jürgen Freudenberger | **Daniel Nicolas Bailon** | **Malek Safieh**

Institute for System Dynamics (ISD), HTWG Konstanz, University of Applied Sciences, Konstanz, Germany

Correspondence

Jürgen Freudenberger, Institute for System Dynamics (ISD), HTWG Konstanz, University of Applied Sciences, Konstanz, Germany.
Email: jfreuden@htwg-konstanz.de

Funding information

Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie, Grant/Award Number: 16ES1045

Abstract

Error correction coding for optical communication and storage requires high rate codes that enable high data throughput and low residual errors. Recently, different concatenated coding schemes were proposed that are based on binary BCH codes with low error correcting capabilities. In this work, low-complexity hard- and soft-input decoding methods for such codes are investigated. We propose three concepts to reduce the complexity of the decoder. For the algebraic decoding we demonstrate that Peterson's algorithm can be more efficient than the Berlekamp–Massey algorithm for single, double, and triple error correcting BCH codes. We propose an inversion-less version of Peterson's algorithm and a corresponding decoding architecture. Furthermore, we propose a decoding approach that combines algebraic hard-input decoding with soft-input bit-flipping decoding. An acceptance criterion is utilised to determine the reliability of the estimated codewords. For many received codewords the stopping criterion indicates that the hard-decoding result is sufficiently reliable, and the costly soft-input decoding can be omitted. To reduce the memory size for the soft-values, we propose a bit-flipping decoder that stores only the positions and soft values of a small number of code symbols. This method significantly reduces the memory requirements and has little adverse effect on the decoding performance.

1 | INTRODUCTION

Concatenated codes using BCH codes of moderate length, with low error correcting capability have recently been applied for error correction in optical communication as well as in storage systems. Such coding systems require very high throughput with hard-input decoding, but some applications also demand efficient soft-input decoding algorithms. These requirements can be met by product codes, half-product codes, staircase codes, or generalized concatenated codes. For instance, product code constructions based on BCH codes were proposed in [1–3]. Moreover, generalized concatenated codes (GCC) with inner BCH codes were investigated in [4–8]. Hardware architectures for such codes were proposed for instance in [9–13]. Similarly, implementations for fast decoding of staircase codes require fast BCH decoding [14, 15].

Due to the required code rates, BCH codes that can only correct single, double, or triple errors are used. The decoding of the concatenated codes typically uses multiple rounds of BCH decoding. Hence, the achievable throughput depends strongly on the speed of the BCH decoder. Moreover, BCH codes that correct only two or three errors are used in random access memory (RAM) applications [16–18], that need high data throughput and a very low decoding latency. In this work, we propose efficient methods for hard- and soft-input decoding for single, double, or triple error correcting BCH codes. We propose three concepts to reduce the complexity of the decoder hardware.

First, we consider the algebraic hard-input decoding. Algebraic BCH decoding consists of three steps: syndrome calculation, calculation of the error location polynomial, and the Chien search which determines the error positions. For BCH codes of moderate length (over Galois fields $GF(2^6)$, …, $GF(2^{12})$), the syndrome calculation and the Chien search can be performed in parallel structures that calculate all syndrome values and all error positions within a single clock cycle, whereas the calculation of the error location polynomial is often performed using the Berlekamp–Massey algorithm (BMA) which requires several iterations. Alternatively, decoders based on Peterson's algorithm [19] were proposed in [9,

10, 20, 21]. Both algorithms determine the error location polynomial. However, we demonstrate that for BCH codes with small error correcting capabilities Peterson's algorithm can be more efficient in terms of decoding complexity than the BMA. We propose an inversion-less version of Peterson's algorithm and a corresponding decoding architecture for single, double, and triple error correcting BCH codes. This decoder is faster than the fully parallel BMA at a comparable circuit size.[1]

Next, we investigate a hybrid decoding approach that combines algebraic hard-input decoding of binary block codes with soft-input decoding. In particular, an acceptance criterion is utilised which determines the reliability of a candidate codeword. This acceptance criterion was proposed in [8] for decoding concatenated codes with binary inner codes and outer Reed–Solomon (RS) codes. In particular, an error and erasure decoding of the outer RS codes was considered in [8], where the acceptance criterion improves the decoding performance. We demonstrate that this acceptance criterion can also be used as a stopping rule to reduce the decoding complexity. For many received codewords the stopping criterion indicates that the hard-decoding result is sufficiently reliable, and the costly soft-input decoding can be omitted. This hybrid decoding approach can enhance the decoder throughput for GC codes as proposed in [13].

Finally, we consider the hardware costs for the soft-input decoder. The hardware size of the soft-input decoder proposed in [13] is dominated by the memory for the soft-values. In order to reduce the memory requirements, we propose a bit-flipping decoder that stores only the positions and soft values of the least reliable code symbols. The number of stored symbols depends on the minimum Hamming distance of the code. For high rate BCH codes, this method significantly reduces the memory requirements and has little adverse effect on the decoding performance.

The remaining article is organised as follows. In Section 2, we propose an inversion-less version of Peterson's algorithm for triple error correcting BCH codes and a pipelined decoder architecture. The bit-flipping decoding is described in Section 3, where we also analyse the stopping rule. In Section 4, we discussed applications of the proposed decoding concepts for concatenated codes and present an approach to reduce the memory requirements for soft values. Performance results and conclusions are provided in Sections 5 and 6, respectively.

2 | ALGEBRAIC BCH DECODER

In this section, we suggest an inversion-less version of Peterson's algorithm for triple error correcting BCH codes. This algorithm is more efficient than the decoders employing Galois field inversion [10, 20]. Moreover, the proposed algorithm provides more flexibility regarding the hardware implementation and enables pipelining to speed up the decoding. A decoding architecture for such a pipelined architecture is presented. Furthermore, we present conditions for decoding failure detection with extended BCH codes. If the actual number of errors exceeds the error correction capability of the BCH code, the decoder may fail to determine a valid codeword. However, such failures can be detected and this failure detection can be utilised for the decoding of concatenated codes [23].

2.1 | Peterson's algorithm

In this section, we briefly review Peterson's algorithm and introduce the notations. The received vector is $r(x) = v(x) + e(x)$, where $v(x) = v_0 + v_1x + \cdots + v_{n-1}x^{n-1}$ is a codeword of length n and $e(x) = e_0 + e_1x + \cdots + e_{n-1}x^{n-1}$ is the error vector. $S_1, S_2, \ldots, S_{2t-1}$ denote the syndrome values which are defined as

$$S_i = r(\alpha^i) = e(\alpha^i), \tag{1}$$

where α is the primitive element of the Galois field $GF(2^m)$. For binary BCH codes, the following relation holds

$$S_{2i} = S_i^2. \tag{2}$$

Let ν be the actual number of errors and t the error correcting capability of the BCH code. The coefficients of the error location polynomial $\sigma(x) = \sigma_0 + \sigma_1x + \cdots + \sigma_tx^\nu$ satisfy a set of equations called Newton's identities. In matrix form these equations are

$$\mathbf{A}_\nu\mathbf{\Delta}_\nu = \mathbf{S}_\nu. \tag{3}$$

$\mathbf{A}_i$ = is the $(i \times i)$ matrix

$$\mathbf{A}_i = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ S_2 & S_1 & 1 & 0 & \ldots & 0 \\ S_4 & S_3 & S_2 & S_1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ S_{2i} & S_{2i-1} & S_{2i-2} & \ldots & & \end{pmatrix}. \tag{4}$$

$\sigma_0 = 1$ and Δ_i denote the vector of coefficients:

$$\mathbf{\Delta}_i = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_i \end{pmatrix}. \tag{5}$$

Moreover, $\mathbf{S}_i$ is the syndrome vector:

$$\mathbf{S}_i = \begin{pmatrix} -S_1 \\ -S_2 \\ \vdots \\ -S_{2i+1} \end{pmatrix}. \tag{6}$$

[1]The material in this paper was presented in part at the International ITG Conference on Systems, Communications and Coding, 2019 [22].

Note that the matrix $\mathbf{A}_i$ is singular for $i > \nu$. Hence, Peterson's algorithm first calculates the number of errors ν. Starting with $i = t$ the determinant $D_i = \det(\mathbf{A}_i)$ is calculated. If $D_i = 0$ then the algorithm reduces the matrix $\mathbf{A}_i$ until $D_i = \det(\mathbf{A}_i) \neq 0$ holds and Equation (3) can be solved.

Finally, the Chien search determines the error positions by searching for the roots of the error location polynomial. The calculation of $\sigma(\alpha^i)$ for $i = 0, \ldots, n-1$ can be conducted in parallel using simple logic operations [12, 16].

2.2 | Calculating the error location polynomial for single, double, and triple errors

For single, double, and triple errors the following direct solutions of the Newton's identities follow [24, 25]:

$$\sigma(x) = 1 + S_1 x \quad \text{for } \nu = 1 \tag{7}$$

$$\sigma(x) = 1 + S_1 x + \frac{S_3 + S_1^3}{S_1} x^2 \quad \text{for } \nu = 2 \tag{8}$$

$$\begin{aligned} \sigma(x) &= 1 + S_1 x + \frac{S_1^2 S_3 + S_5}{S_3 + S_1^3} x^2 \\ &+ \left(S_1^3 + S_3 + S_1 \frac{S_1^2 S_3 + S_5}{S_3 + S_1^3} \right) x^3 \text{ for } \nu = 3 \end{aligned} \tag{9}$$

These solutions are used in [10, 20, 26] for decoding BCH codes. The main difference between [10, 20] is the implementation of the Galois field inversion in Equation (9). For instance, in [10], a parallel hardware implementation is proposed. This architecture requires only four Galois field multipliers, but additionally a Galois field inversion is required. The complexity and the throughput of this architecture are determined by the inversion. For the Galois field $GF(2^{10})$ the size of the inversion is about twice the size of a multiplier and the length of the critical path is four times longer than that of a multiplier. In [20], the inversion is implemented using a look-up table, which is only efficient for small Galois fields, because the table size is of order $O(m2^m)$. Even for moderate Galois field sizes, for example, $m = 8, \ldots, 12$, such look-up tables are costly. In particular, if multiple instances of the decoder are required. Moreover, omitting inversion reduces hardware complexity and speeds up the calculation.

In the following, we propose an algorithm for triple errors that omits the Galois field inversion similar to the approach in [26] that considers double errors. First, we consider the case for single and double errors. Note that the roots of the error location polynomial do not change, if we multiply all coefficients with a non-zero factor. For instance, multiplying the right hand side of Equation (8) with $S_1 \neq 0$, we obtain

$$\sigma(x) = S_1 + S_1^2 x + D_2 x^2 \tag{10}$$

for $\nu = 2$ with the determinant

$$D_2 = S_3 + S_1^3. \tag{11}$$

Note that for $\nu = 1$ and $\nu = 2$, S_1 is non-zero. For a single error in position i we have $S_1 = \alpha^i \neq 0$. Similarly, for two errors in positions i and j, we have $S_1 = \alpha^i + \alpha^j \neq 0$, because $\alpha^i \neq \alpha^j$. Equation (10) is also a solution for $\nu = 1$, because $D_1 = S_1 \neq 0$ and $D_2 = 0$ holds for a single error.

Next, we consider the case $\nu \geq 2$. For $\nu = 2$ and $\nu = 3$, we have $D_2 \neq 0$ [20]. To see that, consider $\nu = 2$. We have $S_1 = \alpha^i + \alpha^j$ and $S_3 = \alpha^{3i} + \alpha^{3j}$. Hence,

$$\begin{aligned} S_1^3 + S_3 &= (\alpha^i + \alpha^j)^3 + \alpha^{3i} + \alpha^{3j} \\ &= \alpha^{i+2j} + \alpha^{2i+j} \neq 0 \text{ for } i \neq j. \end{aligned} \tag{12}$$

Similarly, for $\nu = 3$, we have $S_1 = \alpha^i + \alpha^j + \alpha^k$ and $S_3 = \alpha^{3i} + \alpha^{3j} + \alpha^{3k}$. Consequently,

$$\begin{aligned} S_1^3 + S_3 &= (\alpha^i + \alpha^j + \alpha^k)^3 + \alpha^{3i} + \alpha^{3j} + \alpha^{3k} \\ &= \alpha^{i+2j} + \alpha^{i+2k} + \alpha^{j+2k} + \\ &\quad \alpha^{j+2i} + \alpha^{k+2i} + \alpha^{k+2j}. \end{aligned} \tag{13}$$

The last term is the determinant of the following matrix:

$$\begin{pmatrix} 1 & \alpha^i & \alpha^{2i} \\ 1 & \alpha^j & \alpha^{2j} \\ 1 & \alpha^k & \alpha^{2k} \end{pmatrix}. \tag{14}$$

This matrix has full rank, because the columns are linearly independent. Hence, $D_2 \neq 0$ holds for $\nu = 3$.

Now, multiplying the right hand side of Equation (9) by D_2, we obtain a solution for $\nu =$ three:

$$\sigma(x) = D_2 + S_1 D_2 x + \delta_2 x^2 + D_3 x^3 \tag{15}$$

with

$$\delta_2 = S_1^2 S_3 + S_5 \tag{16}$$

and the determinant

$$D_3 = S_1(S_2 S_3 + S_1 S_4) + S_3^2 + S_1 S_5. \tag{17}$$

Using Equations (1), (11) and (16), we obtain

$$\begin{aligned} D_3 &= S_1(S_1^2 S_3 + S_5) + S_1^6 + S_3^2 \\ &= S_1 \delta_2 + D_2^2. \end{aligned} \tag{18}$$

Algorithm 1 Inversion-less Peterson algorithm

calculate D_2, δ_2, D_3
if $D_3 = = 0$ **then**
  return $\sigma(x) = S_1 + S_1^2 x + D_2 x^2$
else
  return $\sigma(x) = D_2 + S_1 D_2 x + \delta_2 x^2 + D_3 x^3$

The decoding procedure is summarised in Algorithm 1. This algorithm can easily be adapted to the decoding of single and double error correcting codes, for example setting $D_3 = 0$ for double error correcting BCH codes. This is important for the decoding of GC codes that use nested inner codes [12, 13], where the error correcting capability increases from level to level.

In [13, 23], concatenated codes based on extended BCH codes were presented, where the additional parity bit for the BCH codes can be used for failure detection after the algebraic BCH decoding. This failure detection can improve the decoding performance of the outer RS codes with error and erasure decoding.

The proposed decoding approach can be applied to extended BCH codes. Table 1 summarises the conditions for a valid syndrome S_0 of the parity check for extended BCH codes, where X denotes unused conditions. For instance, if the determinant D_3 is non-zero, the algebraic decoder assumes that three errors have occurred. An error pattern with an odd number of errors implies $S_0 = 1$. Hence, $D_3 \neq 0$ and $S_0 = 0$ correspond to an uncorrectable error pattern and a decoding failure can be detected. Similarly, the condition $D_2 \neq 0$ and $D_3 = 0$ indicates two errors, which implies $S_0 = 0$. Consequently, a failure is detected for $D_2 \neq 0$, $D_3 = 0$, and $S_0 = 1$.

2.3 | Hardware architecture

In this section, we present a hardware architecture for the proposed algebraic decoding algorithm and compare its speed (critical path length) with the BMA. Note that the critical path length and the circuit size is dominated by the Galois field multipliers and Galois field inversion. For $GF(2^m)$, the size of a bit-parallel multiplier grows with order $O(m^2)$ and the critical path with $O(m)$. The Galois field inversion is often implemented using Fermat's little theorem, which requires only a single multiplier and a squaring operation, but $m - 1$ clock cycles [27]. Hence, the total number of basic logic operations per inversion is of the order $O(m^3)$. On the other hand, the addition and squaring operations over $GF(2^m)$ are of the order $O(m)$ with a critical path length $O(1)$. Consequently, these two operations are neglected in the following discussion.

Algorithm 1 can be implemented performing all operations in parallel. Such a parallel implementation requires four multipliers and has a critical path length of two multipliers. It is more efficient than the implementation proposed in [10], which uses three multipliers and one inversion. The logic for the inversion is about twice the size of a multiplier (for $GF(2^{10})$) and the critical path length of the inversion is equivalent to four multiplications. The total critical path in [10] has a length that is equivalent to six multiplications. Hence, at a smaller size the proposed algorithm has a significantly shorter critical path.

TABLE 1 Conditions for the parity check for extended BCH codes

ν	S_0	D_1	D_2	D_3
0	0	0	0	0
1	1	$\neq 0$	0	0
2	0	X	$\neq 0$	0
3	1	X	X	$\neq 0$

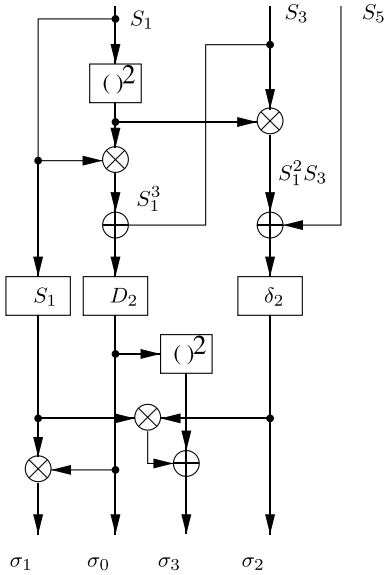

FIGURE 1 Hardware architecture of the decoder pipeline

The Galois field inversion is an atomic operation which limits the efficiency of pipelining, whereas the proposed algorithm enables pipelined architectures that can speed up the decoding. Figure 1 presents a decoder pipeline (without control logic). In the first stage the variables D_2 and δ_2 are calculated according to Equations (11) and (16), respectively. The syndrome S_1 is stored, as it is required for the calculation of D_3 in the second stage. In the second pipeline stage D_3 is determined according to Equation (18). The final error location polynomial is provided as $\sigma_0 = D_2$, $\sigma_1 = S_1 D_2$, $\sigma_2 = \delta_2$, and $\sigma_3 = D_3$.

The pipeline requires four multipliers and additional registers (three registers of width m bits) to store intermediate results. The pipeline reduces the critical path length to a single multiplication. Hence, the pipelined architecture doubles the throughput compared with the structure without pipeline. Note that the fully parallel BMA also has a critical path length

Module	Number of LUT	Number of flip-flops	Throughput
Proposed decoder $m = 8$	145	24	$500 \cdot 10^6$
Multiplier $m = 12$	71	–	$333 \cdot 10^6$
Proposed decoder $m = 12$	318	36	$333 \cdot 10^6$
Parallel BMA $m = 12$	426	84	$111 \cdot 10^6$

Abbreviations: BMA, Berlekamp–Massey algorithm; FPGA, field-programmable gate array.

TABLE 2 Results for the FPGA implementation for the proposed algorithm

of a single multiplication. However, the parallel BMA requires $2t$ multipliers, at least $2t$ registers and t iterations [28, 29]. Hence, the proposed architecture is smaller and about three times as fast as the parallel BMA.

To verify the above size considerations, the proposed decoding algorithm has been implemented on a field-programmable gate array (FPGA) in Verilog. Table 2 contains results for the Xilinx Virtex-7 FPGA, where throughput is presented in terms of codewords per second. The fundamental building blocks of an FPGA are flip-flops and the look-up tables (LUT). The size of the logic is represented by the number of LUT. Table 2 represents data for $m = 8$ and $m = 12$. Both decoders require $3m$ flip-flops for the registers. The size of the decoder for $m = 12$ is dominated by the four multipliers which require about 90% of the logic. The speed of the decoders is determined by the achievable clock frequency f_{clk}. The circuit for $m = 8$ achieves a clock frequency $f_{clk} = 500$ MHz, that is a throughput of $500 \cdot 10^6$ BCH codewords per second, because one codeword is processed per clock cycle. The latency is two clock cycles, that is 4 ns. Moreover, the decoder for $m = 8$ is about 1.5 times faster than the decoder for $m = 12$ which confirms that the critical path length is of order $O(m)$. Similarly, the ratio of 2.2 for the logic size for $m = 12$ and $m = 8$ agrees well with the estimate $O(m^2)$ for the circuit size. The results for the parallel BMA are estimates based on the required number of multipliers and registers. The actual size will be higher. The BMA requires three clock cycles per BCH codeword. Hence, the achievable throughput is $111 \cdot 10^6$ BCH codewords per second at a clock frequency of $f_{clk} = 333$ MHz.

3 | SOFT-INPUT DECODING

In the section, we discuss soft-input decoding of BCH codes based on a bit-flipping procedure. Bit-flipping decoding of binary block codes was pioneered by chase [30]. Chase introduced reliability information-based decoding procedures that generate a list of candidate codewords by flipping bits in the received word. The test patterns for the bit-flipping are based on the least reliable positions of the received word. For instance, the p-Chase algorithm decodes the 2^p binary n-tuples obtained by considering all possible values in the p least reliable positions [31]. The algebraic hard-input decoding is applied for each test pattern. At the end, the most likely codeword is selected from this list of candidates.

In [32, 33], acceptance criteria for bit-flipping decoding were proposed that aim on stopping the decoding, once the ML codeword is found. Such stopping rules aim on achieving near-ML decoding performance and reducing the complexity compared to chase decoding. We consider a similar acceptance criterion for each estimated codeword. This criterion aims on exploiting error and erasure decoding of the outer RS codes in a concatenated coding scheme [8]. This scheme uses the p-Chase algorithm and the acceptance criterion to determine the reliability of the final codeword candidate. This acceptance criterion enables a trade-off between the error and the erasure probability. It was shown in [8], that the bit-flipping combined with error and erasure decoding can significantly improve the soft-input decoding performance compared with Kaneko's rule [32]. It can even outperform ML decoding of the BCH code in combination with outer RS decoding, because the acceptance criterion can efficiently detect unreliable codewords.

In contrast to [8, 13, 31], we do not use a fixed number of decoding cycles. We propose a hybrid decoding approach that combines algebraic hard-input decoding of binary BCH codes with bit-flipping decoding. In particular, we use the acceptance criterion to determine the reliability of each candidate codeword. For many received words the stopping criterion indicates that the hard-decoding result is sufficiently reliable, and the costly bit-flipping decoding can be omitted. In cases where bit-flipping decoding is required, we propose a procedure with a maximum number of decoding cycles, and apply a decision rule to each candidate. The decoder can stop after each decoding round if the codeword fulfils the acceptance criterion. If no reliable codeword is found after the final decoding round an erasure is declared.

3.1 | Kaneko's acceptance criterion

Kaneko's rule is a sufficient condition that the ML codeword is found. However, we show that Kaneko's rule can be too restrictive. First, we briefly introduce notation and explain review Kaneko's acceptance criterion. Afterwards we discuss the new acceptance criterion.

Consider a binary linear code $\mathcal{B}(n,k,d)$, where n is the codeword length, k is the dimension, and d denotes the minimum Hamming distance. We denote the Hamming distance between two binary vectors **a**, **b** by d_{H}(a), (b). We assume transmission of codewords $\mathbf{x}=(x_1,\ldots,x_n)\in\mathcal{C}$ over an additive white Gaussian noise (AWGN) channel with binary phase shift keying. Let **y** be the received vector. The hard decision vector $\widehat{\mathbf{y}}=(\widehat{y}_1,\ldots,\widehat{y}_n)$ is obtained as

$$\widehat{y}_i=\begin{cases}1, & y_i\le 0\\ 0, & y_i>0\end{cases} \tag{19}$$

Furthermore, we use the vector of soft-values $\mathbf{z}=(z_1\ldots, z_n)$ with $z_i=|y_i|$. The soft-values **z** are sorted in ascending order $\widetilde{\mathbf{z}}=(\widetilde{z}_1,\ldots,\widetilde{z}_n)$, such that $|\widetilde{z}_1|\le|\widetilde{z}_2|\le\ldots\le|\widetilde{z}_n|$.

The proposed criterion is similar to Kaneko's acceptance criterion for bit-flipping decoding [32] and uses the same metric. The bit-flipping metric between a codeword **x** and the received vector **y** is defined as

$$l(\mathbf{y},\mathbf{x})=\sum_{x_i\neq\widehat{y}_i}z_i. \tag{20}$$

Let $m=d_{\mathrm{H}}(\mathbf{x}',\widehat{\mathbf{y}})$ be the Hamming distance between the codeword $\mathbf{x}'$ and $\widehat{\mathbf{y}}$. Furthermore, let $\mathbf{x}_0$ denote the codeword found from a hard-input decoder with Hamming distance $m_0=d_{\mathrm{H}}(\mathbf{x}_0,\widehat{\mathbf{y}})$. Kaneko's acceptance criterion

$$l(\mathbf{x}',\mathbf{y})\le\sum_{i=1}^{d-\lfloor\frac{m+m_0}{2}\rfloor}\widetilde{z}_i \tag{21}$$

is a sufficient condition for $\mathbf{x}'$ being the ML codeword [32].

3.2 | Acceptance criterion

In order to motivate the stopping rule and the hybrid decoding approach, we analyse Kaneko's acceptance criterion and some properties of the bit-flipping metric. Let $p(\mathbf{y}|\mathbf{x})$ be the conditional density function for receiving **y** after transmitting the codeword **x** over the AWGN channel.

Lemma 1 *For the AWGN channel we have*

$$l(\mathbf{y},\mathbf{x}'')-l(\mathbf{y},\mathbf{x}')=\log\frac{p(\mathbf{y}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x}'')}, \tag{22}$$

where $\mathbf{x}'$ and $\mathbf{x}''$ are two codewords.

Proof From the definition in Equation (20) follows:

$$l(\mathbf{y},\mathbf{x}'')-l(\mathbf{y},\mathbf{x}')=\sum_{x_i''\neq\widehat{y}_i}z_i-\sum_{x_i'\neq\widehat{y}_i}z_i. \tag{23}$$

Cancelling all terms in both sums with $x_i'=x_i''$ results in

$$l(\mathbf{y},\mathbf{x}'')-l(\mathbf{y},\mathbf{x}')=\sum_{x_i''\neq\widehat{y}_i\wedge x_i''\neq x_i'}z_i-\sum_{x_i'\neq\widehat{y}_i\wedge x_i''\neq x_i'}z_i. \tag{24}$$

The AWGN channel is memoryless. Hence, the probability of receiving the vector **y** given the codeword **x** is

$$p(\mathbf{y}|\mathbf{x})=\prod_{i=1}^{n}p(y_i|x_i), \tag{25}$$

where $p(y_i|x_i)$ is the conditional density function for receiving y_i given the symbol x_i. Using this factorization, we obtain

$$\begin{aligned}\log\frac{p(\mathbf{y}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x}'')} &= \sum_{i=1}^{n}\log p(y_i|x_i')-\sum_{i=1}^{n}\log p(y_i|x_i'') \\ &= \sum_{x_i'\neq x_i''}\log p(y_i|x_i')-\log p(y_i|x_i'') \qquad (26)\\ &= \sum_{x_i'\neq x_i''}\log\frac{p(y_i|x_i')}{p(y_i|x_i'')}, \qquad (27)\end{aligned}$$

where (26) follows from cancelling all terms with $x_i'=x_i''$. For $x_i'\neq x_i''$, we have

$$\log\frac{p(y_i|x_i')}{p(y_i|x_i'')}=\begin{cases}z_i, & x_i'=\widehat{y}_i\\ -z_i, & x_i'\neq\widehat{y}_i\end{cases} \tag{28}$$

Using Equations (24) and (27), we obtain Equation (22).

Using Lemma 1, we can derive some interesting properties of the bit-flipping metric. In the following proposition, $\mathbf{x}'$ denotes the ML codeword and $\mathbf{x}''$ the second best codeword.

Proposition 1

(a) The bit-flipping metric is a maximum likelihood metric.
(b) If the hard decision vector $\widehat{\mathbf{y}}$ is a codeword, then $\mathbf{x}'=\widehat{\mathbf{y}}$ is the ML codeword.
(c) Let $m=d_{\mathrm{H}}(\mathbf{x}',\widehat{\mathbf{y}})$ be the Hamming distance between $\mathbf{x}'$ and $\widehat{\mathbf{y}}$. Then the metrics of the best and second best codeword satisfy

$$l(\mathbf{x}',\mathbf{y}) \ge \sum_{i=1}^{m}\widetilde{z}_i \tag{29}$$

$$l(\mathbf{x}'',\mathbf{y}) \ge \sum_{i=1}^{d-m}\widetilde{z}_i \tag{30}$$

Proof a) follows directly from Lemma 1. The bit-flipping metric can be used for maximum likelihood decoding, because $l(\mathbf{y},\mathbf{x}')\le l(\mathbf{y},\mathbf{x}'')$ implies $p(\mathbf{y}|\mathbf{x}')\ge p(\mathbf{y}|\mathbf{x}'')$.

(b) follows from the fact that the bit-flipping metric is non-negative and $l(\widehat{\mathbf{y}}, \mathbf{y}) = 0$.

To prove (c), we first consider the case where $\widehat{\mathbf{y}}$ is a codeword, that is, $m = 0$ and $l(\mathbf{x}', \mathbf{y}) = 0$. In this case the Hamming metric $d_H(\mathbf{x}'', \widehat{\mathbf{y}}) = d_H(\mathbf{x}'', \mathbf{x}')$ is at least d. Consequently the metric $l(\mathbf{x}'', \mathbf{y})$ is the sum over at least d values z_i. This sum is lower bounded by

$$l(\mathbf{x}'', \mathbf{y}) \geq \sum_{i=1}^{d} \widetilde{z}_i$$

due to the ordering of the soft-values.

Now let $\mathbf{x}'$ be a codeword with Hamming distance m to the vector $\widehat{\mathbf{y}}$. In this case, the codeword $\mathbf{x}''$ differs in at least $d - m$ positions from $\widehat{\mathbf{y}}$. Hence the lower bound in Equation (30) follows. Similarly, Equation (29) follows from the ordering of the soft-values.

From inequality (30), it is seen that

$$l(\mathbf{x}', \mathbf{y}) \leq \sum_{i=1}^{d-m} \widetilde{z}_i \tag{31}$$

is a sufficient condition for $\mathbf{x}'$ being the ML codeword. In fact this is a special case of Kaneko's acceptance criterion according to (21). Kaneko's acceptance criterion is a sufficient condition for $\mathbf{x}'$ being the ML codeword. However, Equation (21) is not a necessary condition. In some cases Kaneko's rule is too restrictive which is shown in the next proposition.

Proposition 2 *Let $\boldsymbol{x}$ be a candidate codeword with $m = d_H(\mathbf{x}, \widehat{\mathbf{y}})$ $\boldsymbol{x}$ can only satisfy Kaneko's acceptance criterion according to Equation (21), if*

$$m < \frac{2d + 1}{3} \tag{32}$$

holds.

Proof The lower bound in Equation (29) depends on the Hamming distance m of the ML codeword. From this lower bound it is observed that Kaneko's acceptance criterion can only be fulfilled, if

$$m \leq d - \lfloor \frac{m + m_0}{2} \rfloor \tag{33}$$

holds. To obtain an upper bound on m, we assume that m takes on the maximum value:

$$m = d - \lfloor \frac{m + m_0}{2} \rfloor . \tag{34}$$

Now we bound the term $d - \lfloor \frac{m+m_0}{2} \rfloor$. Due to $\lfloor \frac{m+m_0}{2} \rfloor > \frac{m+m_0}{2} - 1$ we have

$$m < d - \frac{m + m_0}{2} + 1. \tag{35}$$

From Proposition 1 follows that $m_0 \geq 1$ if $\widehat{\mathbf{y}}$ is not a codeword. Hence, we have

$$m < d - \frac{m}{2} + \frac{1}{2}. \tag{36}$$

From which we obtain the upper bound in Equation (32).

It was shown by chase in [30] that bit-flipping decoding can achieve the near-ML performance and can correct up to $d - 1$ errors. Consequently, the metric of the ML codeword can be the sum over $m = d - 1$ reliability values which is much larger than that indicated in Equation (32).

In order to improve the detection rate, we use the acceptance criterion proposed in [8]:

$$l(\mathbf{x}', \mathbf{y}) \leq \sum_{i=1}^{M} \widetilde{z}_i - T, \tag{37}$$

where M and T are parameters that enable a trade-off between the decoding performance and the number of test patterns. The acceptance criterion is based on Yamamoto and Itoh's decision rule [34], which is a simplification of the Forney's optimal criterion on the trade-off between erasure and error probability [35]. It was shown in [8] that this criterion is a good estimation of the acceptance criterion proposed by Yamamoto and Itoh. However, to obtain the best possible trade-off parameters, these parameters must be determined by Monte Carlo simulation.

We utilise the acceptance criterion (37) for a hybrid decoding approach. In the first decoding step, the syndrome is checked. According to Proposition 1, we stop the decoding if the received vector is a valid codeword. In the second stage, algebraic hard-input decoding is applied based on the hard decision vector $\widehat{\mathbf{y}}$. This step may result in an estimated codeword or a decoding fail as mentioned in Section 2. For instance, the error location polynomial indicates an odd number of errors and the parity bit indicates an even number. Hence, the solution of the error location polynomial is inconsistent. In case of a coding failure, we proceed with bit-flipping decoding. Inconsistent solutions are not considered as candidate codewords. After all algebraic decoding steps the reliability of the estimated codeword is tested according to Equation (37). The decoding is stopped if the criterion is fulfiled. Otherwise, the decoding process enters the next decoding step. In contrast to [8], we test the estimated codeword after each decoding iteration.

4 | GC CODES

In order to demonstrate that the proposed decoding methods achieve a significant reduction of the decoder complexity, we consider the decoding of GC codes with BCH codes as component codes. First, we review the GC construction and its parameters. A detailed discussion can be found in [5, 13, 36, 37]. Afterwards we design a GC code for comparison with a polar code construction proposed in [38] for flash memories. In the next section, we present simulation results for this code with different decoding strategies of the inner BCH codes.

4.1 | GC encoding

The GC codeword is organised in an $n_b \times n_a$ matrix as depicted in Figure 2. All component codes are constructed over GF(2^m). The parameters n_a and n_b denote the lengths of the outer codes $\mathcal{A}^{(l)}$ and inner codes $\mathcal{B}^{(l)}$, respectively. The encoding starts with the outer codes. The rows of the codeword matrix are protected by L Reed-Solomon (RS) codes of length n_a, where L denotes the number of levels. m elements of each column represent one symbol from the Galois field GF(2^m). Hence, m rows of the matrix belong to a codeword of an outer code $\mathcal{A}^{(l)}, l = 0\ldots, L-1$. Note that the code rate of the outer codes increases from level to level. The outer codes protect Lm rows of the matrix. The code dimensions are $k_b^{(0)} = Lm, k_b^{(1)} = (L-1)m, \ldots, k_b^{(L-1)} = m$. The remaining $n_b - Lm$ rows are used for the redundancy of the inner codes or for information bits without outer encoding. Let k_u denote the number of rows without outer encoding. Then, the dimension of the GC code is

$$k = m \sum_{l=0}^{L-1} k_{a,j} + k_u \cdot n_a. \tag{38}$$

After the outer encoding, the columns of the codeword matrix are encoded with binary BCH codes $\mathcal{B}^{(l)}$ of length n_b. In the simulations, we consider only extended BCH codes as inner codes. Each column of the codeword matrix is the sum of L codewords of nested extended BCH codes.

$$\mathcal{B}^{(L-1)} \subset \mathcal{B}^{(L-2)} \subset \ldots \subset \mathcal{B}^{(0)} \tag{39}$$

Hence, a higher level code is a sub-code of its predecessor, where the higher levels have higher error correcting capabilities, that is, $t_{b,L-1} \geq t_{b,L-2} \geq \ldots \geq t_{b,0}$, where $t_{b,l}$ is the error correcting capability of level l. The codeword of the jth column is the sum of L codewords.

$$\mathbf{b}_j = \sum_{l=0}^{L-1} \mathbf{b}_j^{(l)}. \tag{40}$$

These codewords $\mathbf{b}_j^{(l)}$ are formed by encoding the symbols $a_{j,l}$ with the corresponding sub-code $\mathcal{B}^{(l)}$, where $a_{j,l}$ is the j-th symbol (m bits) of the outer code $\mathcal{A}^{(l)}$. For this encoding $(L-l-1)m$ zero bits are prefixed onto the symbol $a_{j,l}$. Note that the every column $\mathbf{b}_j$ of the matrix is a codeword of $\mathcal{B}^{(0)}$, because of the linearity of the nested codes.

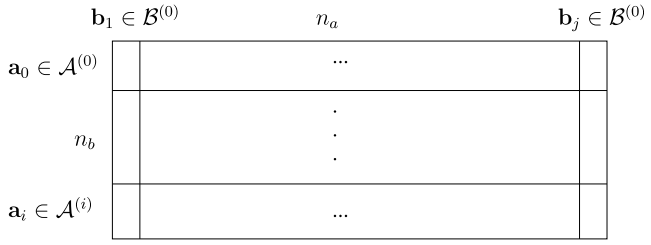

FIGURE 2 Codeword matrix

Example 1 In this example, we construct a code for a comparison with a polar code [38] for applications in flash memories. The comparison will be discussed in Section 5. The polar code is suitable for a block length of 1 kilobyte. Similarly, a GC code with code rate $R = 0.875$ and a code length of approx. 8192 is constructed. This code is designed according to the rules proposed in [8] to achieve a word error probability 10^{-5} at a channel bit error rate of 0.008. All component codes are constructed over GF(2^7).

The outer RS codes have a length of $n_a = 89$. The inner codes are binary extended BCH codes of length $n_b = 92$. For this GC code, we use $L = 12$ levels, where only the first six levels are protected by outer RS codes. Table 3 summarises the parameters of the GC code, where we use the same RS code from level 6 to level 11. To reduce the complexity, only the first three decoding levels use soft-decoding. The fourth level is protected by the code of the third level. The last six levels do not require error correction of the outer decoder, because the inner decoder guarantees the required word error probability.

In Table 3, $k_{b,l}$ and $d_{b,l}$ are the dimension and minimum Hamming distance of the binary inner codes at level l. $k_{a,l}$ and $d_{a,l}$ are the dimension and minimum Hamming distance of the outer RS codes, respectively. Overall, the code has dimension $k = m\sum_{l=0}^{L-1} k_{a,j} = 7168$, length $n = n_a \cdot n_b = 8188$, and rate $R = k/n = 0.8754$.

TABLE 3 Parameters of a generalized concatenated code of length 8188 and rate $R = 0.875$

level l	$k_{b,l}$	$d_{b,l}$	$k_{a,l}$	$d_{a,l}$
0	84	4	65	25
1	77	6	81	9
2–3	70	8	85	5
4	56	12	87	3
5	49	14	87	3
6-11	42	16	89	1

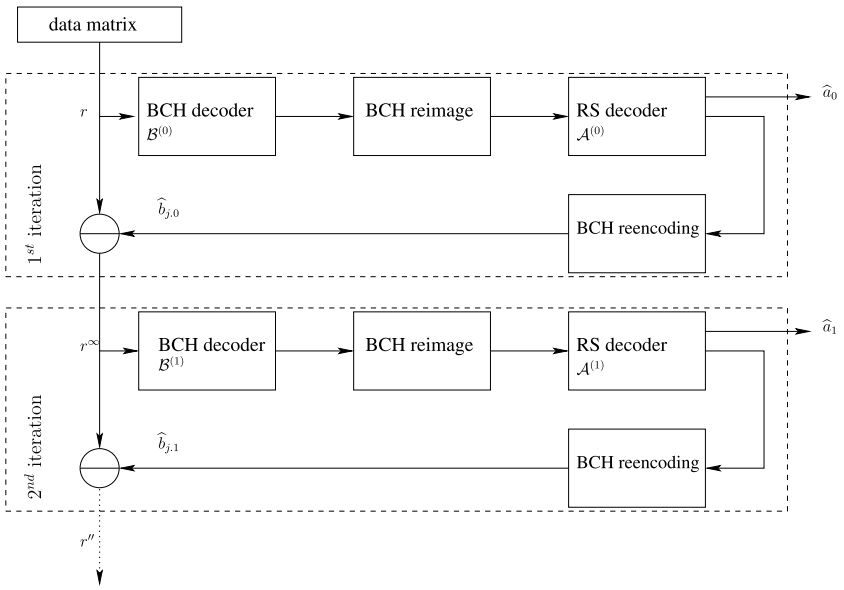

FIGURE 3 Generalized concatenated decoding schemes

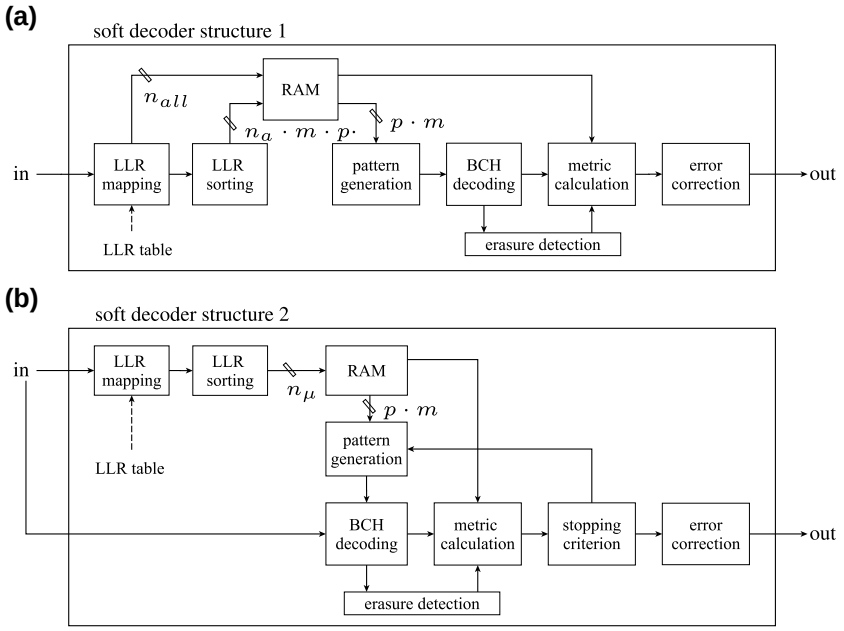

FIGURE 4 Two soft decoder structures

4.2 | GC decoding

Figure 3 illustrates the GC decoding steps, where the decoder processes level by level starting with $l = 0$. Let l be the index of the current level. First the columns are decoded with respect to the BCH code $\mathcal{B}^{(l)}$. After inner decoding, the information bits of the inner codes have to be inferred (re-image) in order to retrieve the code symbols $a_{j,l}$ of the outer RS code $\mathcal{A}^{(l)}$, where j is the column index. If all symbols of the code $\mathcal{A}^{(l)}$ are inferred the RS code can be decoded. Error and erasure decoding is used in all levels of the RS code. After RS decoding, a partial decoding result $\widehat{a}_l$ is available. This result has to be re-encoded using $\mathcal{B}^{(l)}$. The estimated codewords of the inner code $\mathcal{B}^{(l)}$ are subtracted from the codeword matrix before the next level can be decoded. The detailed encoding and decoding process is described in [13].

In this work, we consider two modifications of the decoder proposed in [13] to reduce the decoding complexity. We apply the acceptance criterion according to Section 3 as a stopping criterion after each decoding iteration. Furthermore, we reduce the memory required for storing LLR values.

The modifications are illustrated in the decoder architectures in Figure 4. The first structure corresponds to the decoder presented in [13]. The second structure depicts the new decoder architecture. The first stage in both decoders is a mapping function that maps the quantized input value to LLR values depending on the estimated channel error probability.

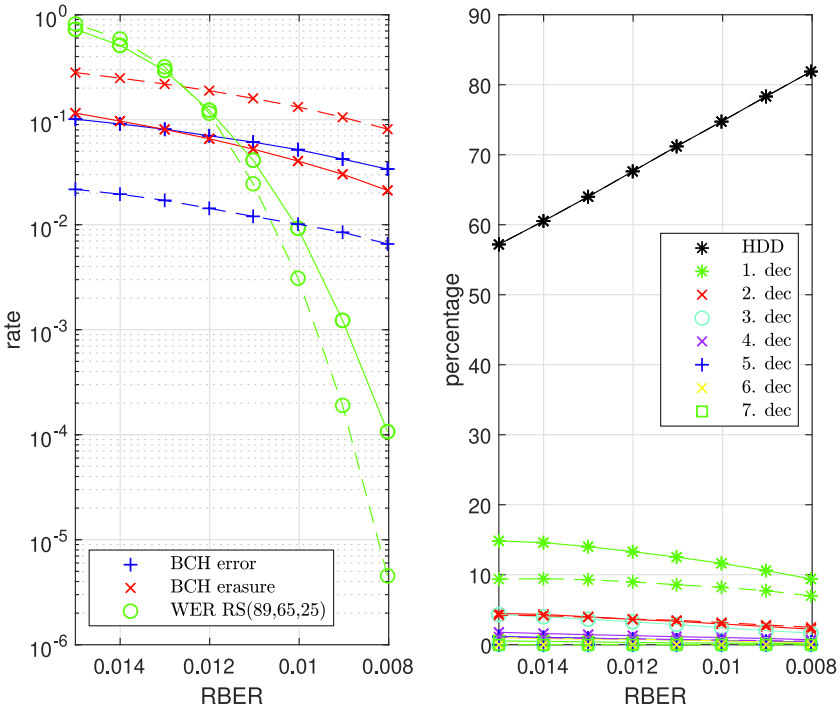

FIGURE 5 Comparison of BCH(92,84,4) soft decoding using (31) (solid line) and the proposed threshold (37) (dashed line) as the stopping criterion

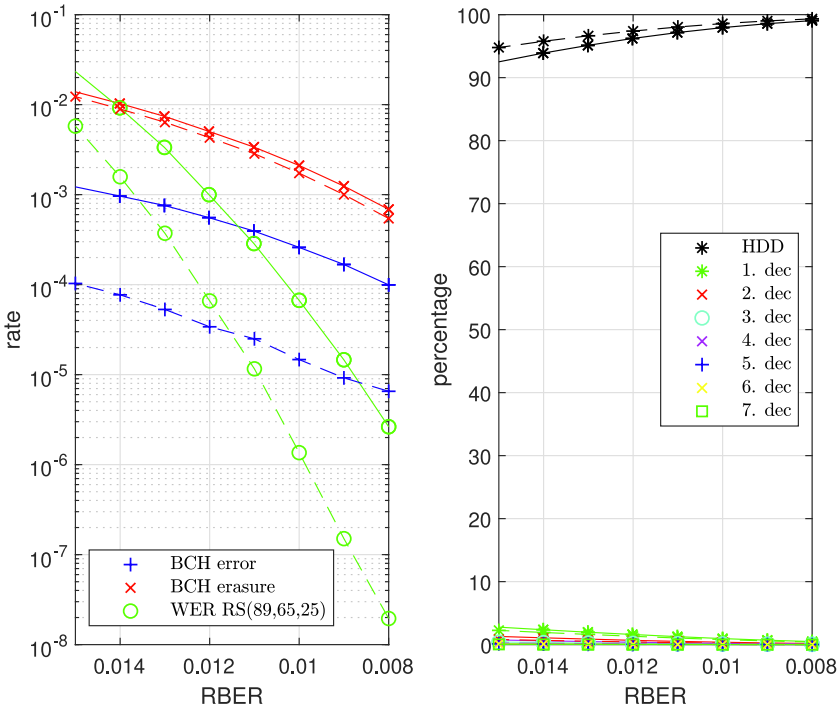

FIGURE 6 Comparison of BCH(92,70,8) soft decoding using Equation (31) (solid line) and the proposed threshold (37) (dashed line) as the stopping criterion

For the soft-decoding, these LLR values have to be stored to determine the least reliable positions. We utilise this sorting to reduce the number of soft values that have to be stored.

In [13], the LLR values for all code symbols and index values that indicate the positions of the least reliable symbols are stored. These positions are needed to determine the bit-flipping patterns for the soft-input decoder as indicated by the block pattern generation in Figure 4. The corresponding LLR values are required for the metric calculation, which determines the sum over the magnitudes of LLR values corresponding to all altered positions according to (20). The positions altered by the bit-flipping patterns can be determined by sorting the soft-values, but the positions altered by the algebraic decoder are only known after each algebraic decoding step. Hence, all LLR values are stored in [13].

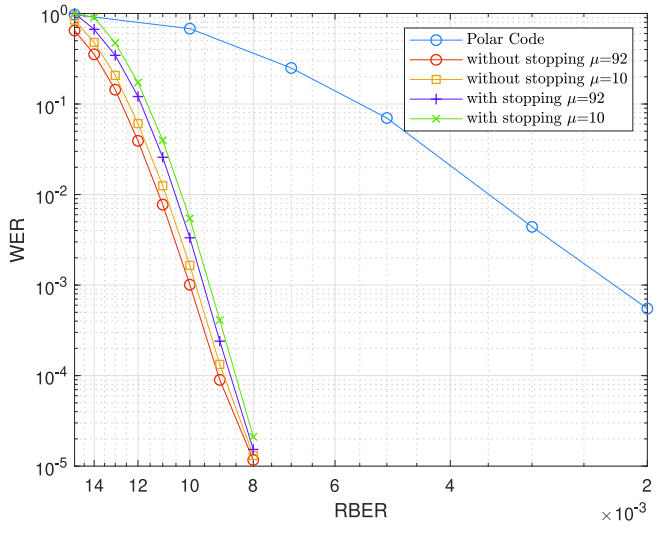

FIGURE 7 Word error rate for different decoding methods for the GC code ($n = 8188$ and $k = 7168$) in comparison with a polar code ($n = 8192$ and $k = 7168$) from [38]

In order to reduce the RAM requirements, we store only the positions and the LLR values corresponding to the least reliable values, where we increase the search depth for the LLR values. This increases the size of the logic for the sorting, but enables a compression of the soft-values. This concept utilises the fact that symbols with low reliability are more likely to be altered by the algebraic decoder. Moreover, quantized input and LLR values reduce the possible variations in the LLR values. For the considered high-rate BCH codes, most input values are mapped to LLR values with maximal magnitude. Only a small fraction of input symbols will have a small magnitude. In [13], the search depths are determined by the maximum number of flipped bit positions p for chase decoding. We store only the LLR values and their positions for the $\mu > p$ least reliable symbols. If the algebraic decoder alters symbols within the stored positions, we can calculate the correct metric value. For positions that are not stored, we set the LLR value to the maximum possible magnitude. With a sufficiently value of μ, this modification typically results in the correct metric according to (20). However, occasionally the metric calculation and the codeword selection will be affected. This leads to a small performance loss as we will see in Section 5.

Example 2 To demonstrate the size reduction for the memory, we consider the BCH codes of length $n_b = 92$ as presented in Example 1. For these codes, only the $\mu = 10$ smallest LLR values are needed to achieve good simulation results, where we consider quantized soft information with six sensing levels as in [38]. The quantized input values are mapped to LLR values with $n_{LLR} = 5$ bits each. One bit is the hard decision bit and four bits indicate the magnitude. To store all soft values,

TABLE 4 Parameters for acceptance criterion

Level	Without stopping	With stopping
0	$M = 4$; $T = 2$	$M = 3$; $T = 1$
1	$M = 6$; $T = 2$	$M = 5$; $T = 0$
2	$M = 8$; $T = 2$	$M = 7$; $T = 1$

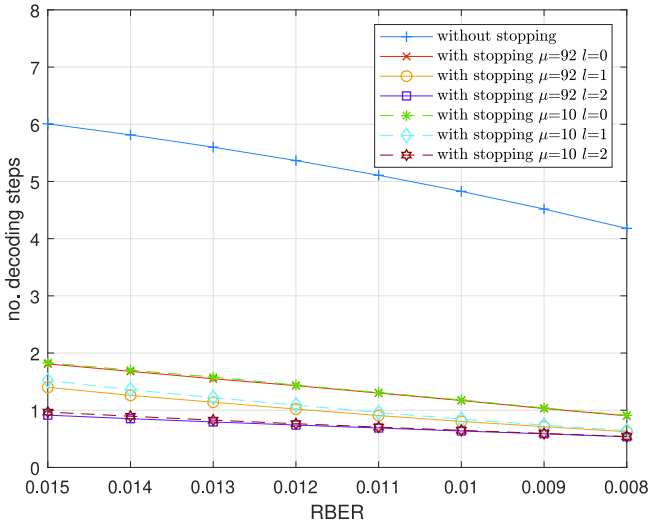

FIGURE 8 Average number of decoding steps for the inner BCH code with and without stopping

$$n_{all} = n_a \cdot n_b \cdot (n_{LLR} - 1) = 8\,188 \cdot 4 = 32\,752$$

bits have to be stored. Additionally, $n_a \cdot m \cdot p = 1869$ bits are required to store symbol positions.

With the proposed approach, we store m bits for each position and $(n_{\mathrm{LLR}} - 1)$ bits for the corresponding magnitude. This results in a total of

$$n_\mu = \mu \cdot n_a \cdot (m + (n_{LLR} - 1))$$
$$n_\mu = 10 \cdot 89 \cdot (7 + 4) = 9\,790$$

bits. In this example, the memory for soft values can be reduced by 72%.

5 | SIMULATION RESULTS

In this section, we present simulation results for the GC code over an AWGN channel with six sensing levels as in [38]. First, we consider the performance of the BCH codes according to Example 1.

The left-hand side of Figure 5 shows different error rates for the codes of the first level $l = 0$. For the inner BCH code (92,84,4), the word error rate and failure (erasure) probability are depicted with Kaneko's stopping rule (solid lines) and the

proposed acceptance criterion (dashed lines). Moreover, word error rates for the outer RS code with error and erasure decoding are presented. In this case, Kaneko's rule results in similar error and erasure probabilities. However, the RS code can correct more erasures than errors. Hence, reducing the error rate at the price of a higher failure rate can improve the performance of RS decoding. This can be observed from the word error rate of the RS code. The new stopping rule reduces the RS word error rate by one order of magnitude at high channel error rates. Note that the new stopping rule does hardly affect the complexity of the decoding. This is demonstrated in the right-hand side of Figure 5, which depicts the percentage of decoding rounds that are stopped after the indicated iteration.

Similar results are presented for the codes of the third level ($l = 2$) in Figure 6. In this case, the performance of the RS code is improved by two orders of magnitude and the decoding complexity is slightly reduced compared with Kaneko's stopping rule. Due to the good error correction capability of this level, 95% to 99% of all received words in the third level can be decoded with hard-input decoding. As the error correction capability increases from level to level, the probability of soft decoding decreases from level to level. Hence, we omit soft-input decoding for the higher levels $l > 2$.

Figure 7 shows the performance of the overall GC code with the different decoding methods. In particular, we use two sets of trade-off parameters for the proposed criterion. These parameters are summarised in Table 4. One set is used for the decoding without premature stopping. In this case, all 2^p iterations of the bit-flipping decoder are used and the criterion according to (37) is used as an acceptance criterion for the final candidate codeword as proposed in [8]. The second set is used for the decoding with premature stopping. All parameters were obtained by simulation thus optimising the performance of the outer RS code with error and erasure decoding.

All simulations were performed for two storage schemes for the LLR values. For the curves denoted by $\mu = 92$, all LLR values were stored, whereas $\mu = 10$ indicates results with reduced memory size. As anticipated the setup without premature stopping and without memory restriction has the best performance. However, differences occur only at higher channel bit error rates. For the target error rate of the design at 0.008, there is only a very small performance loss with the proposed decoding methods. For comparison, we present results for a polar code with similar parameters and with soft-input decoding as reported in [38]. The GC code has a significantly better performance.

Finally, we consider the complexity reduction with the stopping criterion compared with the decoding, as proposed in [13]. The corresponding simulation results are depicted in Figure 8. The figure depicts the average number of decoding steps for the inner BCH code with and without stopping (as well as with and without memory restriction) for the first 3 GC levels. Note that all levels have the same complexity with the decoding according to [13]. With the proposed stopping rule, the number of decoding iterations is reduced by a factor $3 - 4$ for the first level; for the higher levels the complexity is reduced by a factor 4−6 depending on the channel error rate. The memory reduction does hardly affect the number of iterations. The proposed soft-input decoder achieves nearly the performance of the hard-input decoder reported in [13], that is, the overall soft-input decoding process is twice as fas as the decoding in [13].

6 | CONCLUSION

In this work, we have proposed different methods to reduce the decoding complexity for binary BCH codes. We have investigated low-complexity hard- and soft-input decoding methods for single, double, and triple error correcting BCH codes. The algebraic decoder and the stopping rule for the bit-flipping decoder can significantly speed up the decoding process. Furthermore, a concept was discussed to reduce the memory requirements for soft-values.

We have discussed the proposed decoding methods in the context of generalized concatenated codes which are constructed from inner BCH codes and outer RS codes. Such codes are used for error correction in flash memories [8, 13]. We have demonstrated that the decoding can be accelerated and the memory size reduced with hardly any impact on the overall decoding performance. The memory size could be reduced further if the soft values are compressed additionally by exploiting the different probabilities of occurrence.

The proposed concepts can be useful for other concatenated codes, for example, product codes, half-product codes, staircase codes. However, these concepts are limited to the decoding of BCH codes with small error correction capability. Peterson's algorithm is only useful for single, double, and triple error correcting BCH codes. For $t \geq 4$ we were not able to find a solution, where Peterson's algorithm has a lower complexity than the BMA. Moreover, the Chase-2 decoder requires up to 2^{d-1} test patterns, that is, the worst case complexity increases exponentially with minimum Hamming distance d of the code. Hence, the proposed bit-flipping decoding was also limited to codes with low error correction capability.

ACKNOWLEDGEMENTS

The German Federal Ministry of Research and Education (BMBF) supported the research for this article (16ES1045). We thank Hyperstone GmbH, Konstanz for supporting this project.

ORCID

Jürgen Freudenberger https://orcid.org/0000-0002-5913-4981

REFERENCES

1. Cho, S. et al.: Block-wise concatenated BCH codes for NAND flash memories. IEEE Trans. Commun. 62(4), 1164–1177 (2014)
2. Kim, D., Ha, J.: Quasi-primitive block-wise concatenated BCH codes with collaborative decoding for NAND flash memories. IEEE Trans. Commun. 63(10), 3482–3496 (2015)
3. Kim, D., Ha, J.: Serial quasi-primitive BC-BCH codes for NAND flash memories. In: IEEE International Conference on Communications (ICC), pp. 1–6 (2016)

4. Fahrner, A. et al.: Low-complexity GEL codes for digital magnetic storage systems. IEEE Trans. Magn. 40(4), 3093–3095 (2004)
5. Spinner, J., Freudenberger, J., Shavgulidze, S.: A soft input decoding algorithm for generalized concatenated codes. IEEE Trans. Commun. 64(9), 3585–3595 (2016)
6. Zhilin, I., Kreschuk, A., Zyablov, V.: Generalized concatenated codes with soft decoding of inner and outer codes. In: International Symposium on information theory and its applications, pp. 290–294. ISITA (2016)
7. Zhilin, I.V., V.V.Z.: Generalized error-locating codes with component codes over the same alphabet. Probl. Inform. Transm. 53(2), 114–135 (2017)
8. Rajab, M., Freudenberger, J., Shavgulidze, S.: Soft-input bit-flipping decoding of generalized concatenated codes for application in non-volatile flash memories. IET Commun (2018)
9. Lee, K. et al. 100Gb/s two-iteration concatenated BCH decoder architecture for optical communications. In: IEEE Workshop on Signal Processing Systems, pp. 404–409 (2010)
10. Zhang, X., Wang, Z.: A low-complexity three-error-correcting BCH decoder for optical transport network, IEEE Trans. Circuits Syst. II Express Briefs. 59, (10), pp. 663–667 (2012)
11. Yang, C., Emre, Y., Chakrabarti, C.: Product code schemes for error correction in MLC NAND flash memories. IEEE Trans. Very Large Scale Integr. Syst. 20(12), 2302–2314 (2012)
12. Spinner, J., Freudenberger, J.: Decoder architecture for generalized concatenated codes. IET Circuits Dev. Syst. 9(5), 328–335 (2015)
13. Spinner, J., Rohweder, D., Freudenberger, J.: Soft input decoder for high-rate generalised concatenated codes. IET Circuits Dev. Syst. 12(4), 432–438 (2018)
14. Smith, B.P. et al.: Staircase codes: FEC for 100 Gb/s OTN. J. Lightwave Technol. 30(1), 110–117 (2012)
15. Hu, G., Sha, J., Wang, Z.: Beyond 100Gbps encoder design for staircase codes. In: IEEE International Workshop on Signal Processing Systems, pp. 154–158 SiPS (2016)
16. Strukov, D.: The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories. In: Fortieth Asilomar Conference on Signals, Systems and Computers, pp. 1183–1187 (2006)
17. Amato, P. et al.: Ultra fast, two-bit ECC for emerging memories. In: IEEE 6th International Memory Workshop, pp. 1–4. IMW (2014)
18. Yang, C. et al.: Cost-effective design solutions for enhancing PRAM reliability and performance. IEEE Trans. Multi-Scale Comput. Syst. 3(1), 1–11 (2017)
19. Peterson, W.: Encoding and error-correction procedures for the Bose-Chaudhuri codes. IRE Trans. Infor. Theory. 6(4), 459–470 (1960)
20. Lu, E.H., Wu, S.W., Cheng, Y.C.: A decoding algorithm for triple-error-correcting binary BCH codes. Inf Proc. Lett. 80(6), 299–303 (2001)
21. Lin, S., Costello, D.J.: Error control Coding. Prentice-Hall, Upper Saddle River (2004)
22. Freudenberger, J., Rajab, M., Shavgulidze, S.: A low-complexity three-error-correcting BCH decoder with applications in concatenated codes. In: 12th International ITG Conference on Systems, Communications and Coding, pp. 1–5. SCC (2019)
23. Spinner, J., Rajab, M., Freudenberger, J.: Construction of high-rate generalized concatenated codes for applications in non-volatile flash memories. In: IEEE 8th International memory Workshop, pp. 1–4. IMW) (2016)
24. Neubauer, A., Freudenberger, J., Kühn, V.: Coding Theory: Algorithms, Architectures and Applications. John Wiley & Sons (2007)
25. Jiang, Y.: A Practical Guide to Error-Control Coding Using Matlab. Artech House (2010)
26. Ahmadi, N. et al. An optimal architecture of BCH decoder. In: 4th International Conference on Application of Information and Communication Technologies, 2010, pp. 1–5
27. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2m) using normal bases. Inf Comput. 78(3), 171–177 (1988)
28. Liu, W., Rho, J., Sung, W.: Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories. In: IEEE Workshop on Signal Processing Systems Design And Implementation, pp. 303–308. SIPS) (2006)
29. Freudenberger, J., Spinner, J.: A configurable Bose-Chaudhuri-Hocquenghem codec architecture for flash controller applications. J Circ Syst Comput. 23(2), 1–15 (2014)
30. Chase, D.: Class of algorithms for decoding block codes with channel measurement information, IEEE Transactions on Information Theory, 1972, pp. 170–182
31. Tendolkar, N., Hartmann, C.: Generalisation of Chase algorithms for soft decision decoding of binary linear codes. IEEE Trans Inf Theor. 30(5), 714–721 (1984)
32. Kaneko, T. et al.: An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder, IEEE Trans Inf Theor, 1994, 40, (2), pp. 320–327
33. Kaneko, T., Nishijima, T., Hirasawa, S.: An improvement of soft-decision maximum-likelihood decoding algorithm using hard-decision bounded-distance decoding. IEEE Trans Inf Theor. 43(4), 1314–1319 (1997)
34. Yamamoto, H., Itoh, K.: Viterbi decoding algorithm for convolutional codes with repeat request, IEEE Trans Inf Theor. 26(5), pp. 540–547 (1980)
35. Forney, G.: Exponential error bounds for erasure, list, and decision feedback schemes. IEEE Trans Inf Theor. 14(2), 206–220 (1968)
36. Dumer, I.: Concatenated codes and their multilevel generalizations. Handbook of Coding Theory, vol. II. Elsevier, Amsterdam (1998)
37. Bossert, M.: Channel coding for telecommunications. Wiley (1999)
38. Song, H., et al.: Polar-coded forward error correction for MLC NAND flash memory. Sci China Inf Sci. 61(10), 102307 (2018)

How to cite this article: Freudenberger J, Nicolas Bailon D, Safieh M. Reduced complexity hard- and soft-input BCH decoding with applications in concatenated codes. *IET Circuits Devices Syst*. 2021;15:284–296. https://doi.org/10.1049/cds2.12026