A Soft Input Decoding Algorithm for Generalized Concatenated Codes

Jens Spinner, Jürgen Freudenberger, *Member, IEEE*, and Sergo Shavgulidze

***Abstract*— This paper proposes a soft input decoding algorithm and a decoder architecture for generalized concatenated (GC) codes. The GC codes are constructed from inner nested binary Bose–Chaudhuri–Hocquenghem (BCH) codes and outer Reed–Solomon codes. In order to enable soft input decoding for the inner BCH block codes, a sequential stack decoding algorithm is used. Ordinary stack decoding of binary block codes requires the complete trellis of the code. In this paper, a representation of the block codes based on the trellises of supercodes is proposed in order to reduce the memory requirements for the representation of the BCH codes. This enables an efficient hardware implementation. The results for the decoding performance of the overall GC code are presented. Furthermore, a hardware architecture of the GC decoder is proposed. The proposed decoder is well suited for applications that require very low residual error rates.**

***Index Terms*— Generalized concatenated codes, Bose-Chaudhuri-Hocquenghem codes, Reed-Solomon code codes, decoder architecture, soft input decoding, flash memory.**

I. Introduction

ERROR correction coding (ECC) based on GC codes has a high potential for various applications in data communication and data storage systems, e.g., for digital magnetic storage systems [1], for non-volatile flash memories [2], and for two-dimensional bar codes [3]. Generalized concatenated codes are typically constructed from inner nested binary Bose-Chaudhuri-Hocquenghem (BCH) codes and outer Reed-Solomon (RS) codes [4]–[6]. With algebraic decoding, GC codes have a low decoding complexity compared to long BCH codes. Such codes are well suited for fast hardware decoding architectures [7].

A codeword of a GC code can be considered as a matrix. For encoding the information is stored in the matrix. In the first encoding step the rows of the matrix are protected by block codes (the outer codes) over the Galois field $GF(2^m)$. Next each column is protected with binary codes, the inner codes. Typically binary BCH codes are used as inner codes and RS codes as outer codes [8]. A decoder processes the erroneous data in multiple decoding steps. In [7] algebraic decoding is used in each decoding step. This is adequate if the channel provides no soft information about the transmitted or stored bits. However, if the channel provides reliability information, e.g. an AWGN channel, this soft information should be exploited by the decoder. For GC codes, it is sufficient to decode the inner codes exploiting the soft information. In [7] a pipelined decoder architecture for GC codes was proposed which is based on algebraic hard input decoding of the component codes. In this work we extend this design to soft input decoding. We propose a new decoding algorithm and decoder architecture for GC codes.

There exist numerous soft input decoding algorithms for binary block codes (see [5] for an overview). For instance, reliability-based decoding algorithms like Chase decoding [9], [10], ordered statistic decoding [11], and the Dorsch algorithm [12]–[14], just to name a few. Such algorithms can offer a performance that is similar to maximum-likelihood (ML) decoding, but usually do not guarantee to find the ML codeword. However, many of these methods would not be suitable for a fast hardware implementation. Furthermore, many channels with quantized output provide only a small number of decision thresholds and hence only 2 or 3 bits of soft information per code bit which is not sufficient for many reliability-based decoding algorithms.

In this work we consider sequential stack decoding as proposed in [15]. Sequential decoding has a low computational complexity, if the noise level is small. This is the case for many applications of GC codes, e.g., for error correction coding in storage systems. Sequential decoding was originally introduced for tree codes. In order to decode binary block codes, the syndrome trellis is used as a representation of the code [16]. For block codes the number of trellis states grows exponentially with the number of redundancy bits. Hence, the trellis based sequential decoding as proposed in [15] is only feasible for codes with low error correcting capabilities.

The GC construction is based on nested-BCH codes where higher levels have higher error correcting capabilities. In this work, we propose some improvements compared to the algorithm from [15]. We use the code trellis only in the first level of the GC code for decoding the inner BCH code, i.e., for the code with the lowest error correcting capability. For the next levels, we exploit the partitioning of the nested-BCH codes in order to reduce the space complexity required for representing the code. We propose a representation based on super-

Manuscript received December 21, 2015; revised May 14, 2016; accepted June 30, 2016. Date of publication July 12, 2016; date of current version September 14, 2016. This work was supported by the German Federal Ministry of Research and Education (BMBF) under Grant 03FH025IX5. Parts of this have been published in: J. Freudenberger, T. Wegmann, and J. Spinner, "An efficient hardware implementation of sequential stack decoding of binary block codes," IEEE 5th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), Berlin, 2015, pp. 135-138. doi:10.1109/ICCE-Berlin.2015.7391215. The associate editor coordinating the review of this paper and approving it for publication was L. Dolecek.

J. Spinner and J. Freudenberger are with the Institute for System Dynamics, HTWG Konstanz, University of Applied Sciences, Konstanz 78462, Germany (e-mail: jens.spinner@htwg-konstanz.de; juergen.freudenberger@htwg-konstanz.de).

S. Shavgulidze is with the Faculty of Power Engineering and Telecommunications, Georgian Technical University, Tbilisi 0175, Georgia (e-mail: sshavgulidze@gncc.ge).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCOMM.2016.2590428

codes [17]. A similar method was introduced in [18] and [19] to reduce the decoding complexity of maximum-likelihood decoding. The concept of supercode decoding is well suited for decoding of GC codes, because the higher levels of the nested-BCH codes are supercodes of the lower levels. Furthermore, we propose a sequential list-of-two decoding algorithm that improves the residual word error rate.

The residual error rates for GC codes can be determined analytically [4], which is important for applications where a low probability of failure has to be guaranteed. A particular example is coding for flash memories that provide soft information about the state of the memory cells. Traditionally, a binary symmetric channel (BSC) is used as channel model for flash memories and BCH codes are used for error correction [20]–[22]. Recently, for NAND flash memories concatenated codes were proposed that are constructed from long BCH codes [23], [24]. These codes can achieve low residual error rates, but require very long codes and hence a long decoding latency, which might not be acceptable for all applications of flash memories.

The performance of error correction coding can be improved if reliability information about the state of the cell is available [25]. In this case, the channel can be considered as binary input additive white Gaussian noise (AWGN) channel, where the channel output is quantized using a small number of bits [23]. In order to exploit the reliability information soft input decoding algorithms are required. For instance, low-density parity-check (LDPC) codes can provide stronger error correcting performance in NAND flash memories [26]–[29]. However, LDPC codes have high residual error rates (the error floor) and are not suitable for applications that require very low decoder failure probabilities [2]. For instance, the JEDEC standard for Solid-State Drive (SSD) recommends an uncorrectable bit error rate of less than 10^{-15} for client applications and of less than 10^{-16} for enterprise solutions [30]. For some applications, block error rates less than 10^{-16} are required [31]. We demonstrate that GC codes can achieve such low residual error rates.

Note that the threshold voltage sensing operation that is required to obtain the soft information incurs a larger energy consumption and a latency penalty [32]. Hence, the soft information might only be used for blocks where the decoding without reliability information fails. We therefore propose a decoder architecture that supports hard and soft input decoding. The hard input mode provides fast decoding, whereas the number of decoding cycles in the soft input mode increases with the actual number of erroneous bits. This enables a trade-off between decoding speed and hardware complexity.

In Section II, we give a brief introduction into the GC construction, explaining its structure and the decoding algorithm. Later-on we describe the stack algorithm. This algorithm is used to decode the nested-BCH codes. In Section III, we describe the proposed supercode decoding algorithm and sequential list-of-two decoding. Next, we consider the GC decoding procedure and the decoding error probability in Section IV. Finally, an implementation of the soft decoding of the GC codes and results for the decoding complexity are presented.

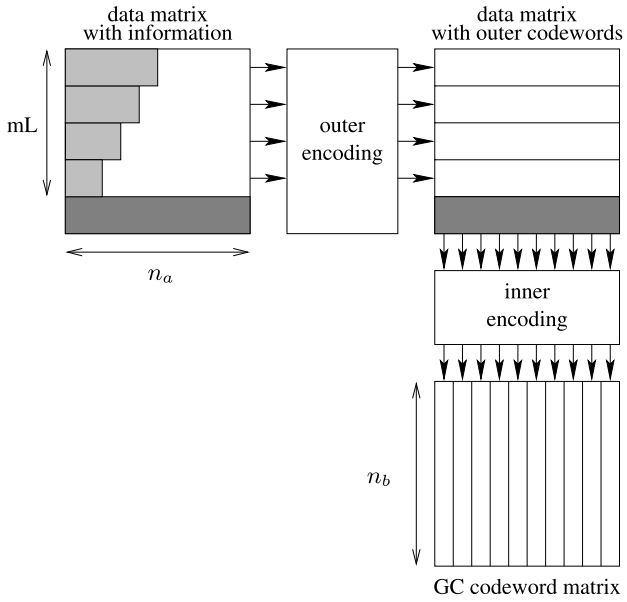


Fig. 1. GC encoding scheme for L levels. The GC codeword matrix has size $n_a \times n_b$, where n_a is the length of the outer codes over the Galois field $GF(2^m)$. The binary inner codes have length n_b. The grey areas represent the redundancy that is calculated by outer and inner encoding.

II. GC Construction

In this section we explain the GC construction and its parameters. A detailed discussion can be found in [4]. The encoding process of a GC code is illustrated in Fig. 1. The GC codeword is arranged in an $n_b \times n_a$ matrix, where n_a and n_b are the lengths of the outer and inner codes, respectively. The encoding starts with the outer codes. The rows are protected by L Reed-Solomon codes of length n_a, i.e., L denotes the number of levels. m elements of each column represent one symbol from the Galois field $GF(2^m)$. Hence, m rows form a codeword of an outer code $\mathcal{A}^{(i)}, i = 0 \ldots, L-1$. Note that the code rate of the outer codes increases from level to level. The outer codes protect Lm rows of the matrix. The remaining $n_b - Lm$ rows are used for the redundancy of the inner codes. After the outer encoding the columns of the codeword matrix are encoded with binary inner codes of length n_b. In Fig. 1, the grey areas indicate the redundancy parts of the codeword matrix that are filled by outer and inner encoding. Each column of the codeword matrix is the sum of L codewords of nested linear BCH codes.

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

Hence, a higher level code is a sub-code of its predecessor, where the higher levels have higher error correcting capabilities, i.e., $t_{b,L-1} \geq t_{b,L-2} \geq \ldots \geq t_{b,0}$, where $t_{b,i}$ is the error correcting capability of level i. The code dimensions are $k_b^{(0)} = Lm, k_b^{(1)} = (L-1)m, \ldots, k_b^{(L-1)} = m$.

The codeword of the j-th column is the sum of L codewords.

$$\mathbf{b}_j = \sum_{i=0}^{L-1} \mathbf{b}_j^{(i)}. \tag{2}$$

TABLE I

PARAMETERS OF THE CODE FROM EXAMPLE 1. $k_{b,i}$ AND $d_{b,i}$ ARE THE DIMENSION AND MINIMUM HAMMING DISTANCE OF THE BINARY INNER CODE OF LEVEL i. $k_{a,i}$ AND $d_{a,i}$ ARE THE DIMENSION AND MINIMUM HAMMING DISTANCE OF THE OUTER RS CODES

level i	$k_{b,i}$	$d_{b,i}$	$k_{a,i}$	$d_{a,i}$
0	54	3	187	157
1	45	5	321	23
2	36	9	333	11
3	27	11	331	13
4	18	15	335	9
5	9	17	337	7

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

Example 1: We consider a GC code that is designed for 2kbyte information blocks. For this GC code we use $L = 6$ levels with inner nested-BCH codes over $GF(2^6)$ and outer RS codes over $GF(2^9)$. In the first level the inner code can correct a single error and therefore six redundancy bits are needed. Thus the number of rows is $n_b = 6 \cdot 9 + 6 = 60$.

All inner codes are binary BCH codes of length $n_b = 60$, where the code $\mathcal{B}^{(0)}$ has $k_{b,0} = 54$ and minimum distance $d_{b,0} = 3$. The outer RS codes are constructed over the Galois field $GF(2^9)$. Hence, the dimension of the inner codes is reduced by $m = 9$ bits with each level. The GC code is constructed from $L = 6$ outer RS codes of length $n_a = 343$. The parameters of the codes are summarized in Table I. The code has overall dimension $k = m \sum_{i=0}^{L-1} k_{a,i} = 16596$ and length $n = n_a \cdot n_b = 20580$. The code has a code rate $R = 0.806$. The design of this code will be discussed later on.

III. SEQUENTIAL STACK DECODING

The GC decoder processes level by level, where first the inner codes and then the outer codes are decoded. In order to enable soft input decoding of the overall GC code, a soft input decoding algorithm for the inner codes is required. In this section we describe sequential decoding procedures using the stack algorithm for block codes. These decoding methods are used to decode the binary inner codes.

A. Sequential Stack Decoding Using a Single Trellis

Firstly, we consider the sequential decoding procedure as presented in [15]. All algorithms considered in this paper are based on this decoding method which uses a trellis to represent the code. Later-on, we will present improvements to this decoding algorithm.

The stack decoding procedure uses the trellis representation. A trellis $\mathcal{T} = (\mathcal{S}, \mathcal{W})$ is a labeled, directed graph, where $\mathcal{W} = \{w\}$ denotes the set of all branches in the graph and $\mathcal{S} = \{\sigma\}$ is the set of all nodes. The set $\mathcal{S}$ is decomposed into $n+1$ disjoint subsets $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_n$ that are called levels of the trellis. Similarly, there exists a partition of the set $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2 \cup \ldots \cup \mathcal{W}_n$. A node $\sigma \in \mathcal{S}_t$ of the level t may be connected with a node $\widetilde{\sigma} \in \mathcal{S}_{t+1}$ of the level $t+1$ by one or several branches. Each branch w_t is directed from a node σ of level $t-1$ to a node $\widetilde{\sigma}$ of the next level t. We assume that the end levels have only one node, namely $\mathcal{S}_0 = \{\sigma_0\}$ and $\mathcal{S}_n = \{\sigma_n\}$. A trellis is a compact method of presenting all codewords of a code. Each branch of the trellis w_t is labeled by a code symbol $v_t(w_t)$. Each distinct codeword corresponds to a distinct path in the trellis, i.e., there is a one-to-one correspondence between each codeword $\mathbf{v}$ in the code and a path $\mathbf{w}$ in the trellis: $\mathbf{v}(\mathbf{w}) = v_1(w_1), \ldots, v_n(w_n)$. We denote code sequence segments and path segments by $\mathbf{v}_{[i,j]} = v_i, \ldots, v_j$ and $\mathbf{w}_{[i,j]} = w_i, \ldots, w_j$, respectively. The *syndrome trellis*, can be obtained using its parity-check matrix [16]. The syndrome trellis is minimal inasmuch as this trellis has the minimal possible number of nodes $|\mathcal{S}|$ among all possible trellis representations of the same code.

The sequential decoding procedure as presented in [15] is a stack algorithm, i.e., a stack is required to store interim results. The stack contains code sequences of different lengths. Let $\mathbf{v}_t$ denote a code sequence of length t, i.e., $\mathbf{v}_t = v_1, \ldots, v_t$. Each code sequence is associated with a metric and a node σ_t. The node σ_t is the node in the trellis that is reached if we follow the path corresponding to the code sequence through the trellis. The metric rates each code sequence and the stack is ordered according to these metric values where the code sequence at the top of the stack is the one with the largest metric value. There exist different metrics in the literature to compare code sequences of different length. In the following, we consider the Fano metric which is defined as follows. Let v_i be the i-th code bit and r_i the i-th received symbol for transmission over a discrete memoryless channel. The Fano metric for a code bit v_i is defined by

$$M(r_i|v_i) = \log_2 \frac{p(r_i|v_i)}{p(r_i)} - B \tag{3}$$

where $p(r_i|v_i)$ is the channel transition probability and $p(r_i)$ is the probability to observe r_i at the channel output. The term B is a bias term that is typically chosen to be the code rate R [33]. The Fano metric of a code sequence $\mathbf{v}_t$ is

$$M(\mathbf{r}_t|\mathbf{v}_t) = \sum_{i=1}^{t} M(r_i|v_i) \tag{4}$$

where $\mathbf{r}_t$ is the sequence of the first t received symbols. Note that the Fano metric according to Equation (3) is only defined for discrete memoryless channels (DMC). We consider the quantized AWGN channel which is a DMC. Binary block codes typically have no tree structure. Consequently, the Fano metric is not necessarily the best metric for all binary block codes. For instance, in [34] a metric with variable bias term was proposed for linear block codes. However, in our simulations for binary BCH codes we found that $B = R$ provides good results for all considered channel conditions.

We demonstrate Algorithm 1 in the following example, where for simplicity we assume transmission over a binary symmetrical channel.

Algorithm 1 Sequential Stack Decoding Using a Single Trellis

Data: received word $\mathbf{r}$
Result: estimated codeword $\hat{\mathbf{v}}$
sequential decoding starts in the first node σ_0 of the trellis;
calculate the metric values for $v_1 = 0$ and $v_1 = 1$;
insert both paths into the stack according to their metric values;
while *the top path has not approached the end node* σ_n **do**
  remove the code sequence $\mathbf{v}_t$ at the top from the stack;
  if *the branch* $v_{t+1} = 0$ *exists in the trellis for the node* σ_t *corresponding to the top path* $\mathbf{v}_t$ **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 0)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 0)$ into the stack;
  end
  if *the branch* $v_{t+1} = 1$ *exists in the trellis for the node* σ_t *corresponding to the top path* $\mathbf{v}_t$ **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 1)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 1)$ into the stack;
  end
end
return the codeword $\hat{\mathbf{v}}$ corresponding to the top path in the final iteration;

Example 2: Consider for instance the code $\mathcal{B} = \{(0000), (1110), (1011), (0101)\}$ *with parity-check matrix*

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

The corresponding trellis is depicted in Fig 2a). We assume transmission over a binary symmetrical channel with error probability 0.1. *Hence, we have*

$$M(r_i|v_i) \approx \begin{cases} 0.3 & \text{for } r_i = v_i \\ -2.8 & \text{for } r_i \neq v_i \end{cases}$$

The following tables represent the stack for the received sequence $\mathbf{r} = (0010)$.

1*st iteration*

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0	0.3
1	−2.8

2*nd iteration*

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
00	0.6
01	−2.5
1	−2.8

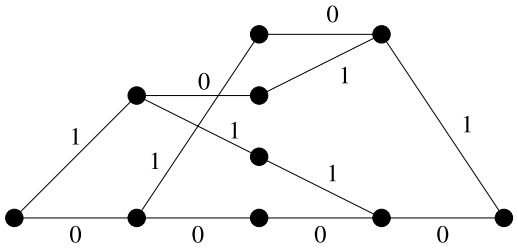
3*rd iteration*

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
000	−2.2
01	−2.5
1	−2.8

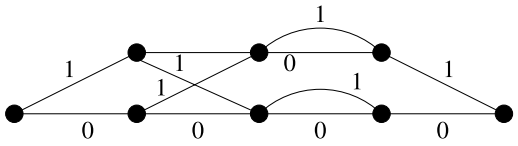
4*th iteration*

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0000	−1.9
01	−2.5
1	−2.8

a) Trellis of code $\mathcal{B}$



b) Trellis of supercode $\mathcal{B}_1$



c) Trellis of supercode $\mathcal{B}_2$

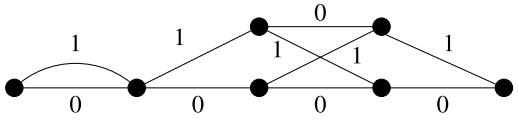


Fig. 2. Trellises of the example code and of its two supercodes. Trellis a) is a representation of the complete code, whereas the trellises b) and c) are the representations of the supercodes.

B. Supercode Decoding for Nested-BCH Codes

In this section we first describe the supercode decoding. Then we discuss the proposed application of supercode decoding for the nested-BCH codes that are used in the GC code. A supercode is a superset $\mathcal{B}_1$ of the original code $\mathcal{B} \subset \mathcal{B}_1$. In order to decode the original code $\mathcal{B}$, two supercodes $\mathcal{B}_1$ and $\mathcal{B}_2$ have to be constructed such that $\mathcal{B}_1 \cap \mathcal{B}_2 = \mathcal{B}$. The supercodes have fewer redundancy bits and thus fewer trellis states. The supercodes can be constructed such that each code has half of the original redundancy bits. This reduces the number of states from $O(2^r)$ to $O(2^{\frac{r}{2}})$ in standard order notation, where r is the number of parity bits. The concept of supercode decoding is well suited for decoding of GC codes, because the higher levels of the nested-BCH codes are supercodes of the lower levels (cf. Equation (1)).

A *supercode* $\mathcal{B}_i$ of the block code $\mathcal{B}$ is a code containing all codewords of $\mathcal{B}$. For a linear code $\mathcal{B}$ with parity-check matrix $\mathbf{H}$, we can construct two supercodes $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B} = \mathcal{B}_1 \cap \mathcal{B}_2$. Let $\mathbf{H} = \begin{pmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{pmatrix}$ be the parity-check matrix of the code $\mathcal{B}$, this means that $\mathbf{H}_1$ and $\mathbf{H}_2$ are two sub-matrices of $\mathbf{H}$. Then the sub-matrices $\mathbf{H}_1$ and $\mathbf{H}_2$ define the supercodes $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively.

Example 3: Consider the code $\mathcal{B}$ *from Example 5. We obtain*

$$\mathbf{H}_1 = \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix}$$
$$\Downarrow$$
$$\mathcal{B}_1 = \{\underline{(0000)}, (1100), \underline{(1110)}, (0010), \underline{(1011)}, (1001), \underline{(1011)}, \underline{(0101)}\}$$

and

$$\mathbf{H}_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix}$$
$$\Downarrow$$
$$\mathcal{B}_2 = \{\underline{(0000)}, (1000), (0110), \underline{(1110)}, \underline{(1011)}, (1101), (0011), \underline{(0101)}\},$$

where the underlined vectors are the codewords of the code $\mathcal{B}$. The corresponding supercode trellises are depicted in Fig. 2b) and 2c).

Next we state the proposed sequential decoding algorithm. Any path stored in the stack is associated with a metric value as well as two states $\sigma_{t,1}$ and $\sigma_{t,2}$ which are the states in the trellis for supercode $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively.

We demonstrate decoding Algorithm 2 in the following example, where we consider the same setup as in Example 5.

Example 4: The following tables represent the stack for the received sequence $\mathbf{r} = (0010)$ for the proposed algorithm.

1st iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0	0.3
1	−2.8

2nd iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
00	0.6
01	−2.5
1	−2.8

3rd iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
001	0.9
000	−2.2
01	−2.5
1	−2.8

4th iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
000	−2.2
01	−2.5
1	−2.8

5th iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0000	−1.9
01	−2.5
1	−2.8

Note that the stack in the third iteration differs from Example 5, because the code sequence 001 *exists in both supercode trellises but not in the actual code. This code sequence is deleted in the next iteration, because it cannot be extended in both supercode trellises.*

As the previous example demonstrates, the time complexity of the proposed algorithm may be larger than with Algorithm 1. This results from code sequences that exist in the super codes, but are not valid in the actual code. Nevertheless, both algorithms result in the same codeword.

Theorem 1: Algorithm 1 and Algorithm 2 result in the same estimated codeword.

Proof: Both algorithms differ only with respect to the representation of the code. To prove the proposition it is sufficient to verify that both representations are equivalent. We first prove by induction that the estimated codeword corresponds to a valid path in both supercode trellises, i.e., it is a codeword in both supercodes. The base case is the initial step where the code bits 0 and 1 are inserted in the stack. Note that a linear code has no code bit positions with constant values. Hence, the transitions $v_1 = 0$ and $v_1 = 1$ exist in both supercode trellises. For the inductive step, we assume that a path for the code sequence $\mathbf{v}_t$ exists in both supercode trellises. It follows from Algorithm 2 that this path is only extended if the extended path exists in both supercode trellises. This proves the claim that the estimated codeword corresponds to a valid path in both supercode trellises. Now note that $\mathcal{B} = \mathcal{B}_1 \cap \mathcal{B}_2$, i.e., a path is only valid in both supercode trellises if and only if it is a valid codeword of the code $\mathcal{B}$. ■

Algorithm 2 Sequential Stack Decoding Using Supercode Trellises

Data: received word $\mathbf{r}$
Result: estimated codeword $\hat{\mathbf{v}}$
sequential decoding starts in the nodes $\sigma_{0,1}$ and $\sigma_{0,2}$ of the supercode trellises;
calculate the metric values for $v_1 = 0$ and $v_1 = 1$;
insert both paths into the stack according to their metric values;
while *the top path has not approached the end nodes $\sigma_{n,1}$ and $\sigma_{n,2}$* **do**
  remove the code sequence $\mathbf{v}_t$ at the top from the stack;
  if *the branch $v_{t+1} = 0$ exists in the trellis for both nodes $\sigma_{t,1}$ and $\sigma_{t,2}$ corresponding to the top path $\mathbf{v}_t$* **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 0)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 0)$ into the stack;
  end
  if *the branch $v_{t+1} = 1$ exists in the trellis for both nodes $\sigma_{t,1}$ and $\sigma_{t,2}$ corresponding to the top path $\mathbf{v}_t$* **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 1)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 1)$ into the stack;
  end
end
return the codeword $\hat{\mathbf{v}}$ corresponding to the top path in the final iteration;

Algorithm 2 reduces the space complexity required for representing the code. We demonstrate this in the following example.

Example 5: We consider three BCH codes from Table I. All codes have length $n = 60$. In the first level, we use a single-error correcting code. This code has 3,262 nodes in the trellis. This code is a supercode of the BCH code of the second level. The trellis of the second level has 159,742 nodes. However, utilizing the trellis of the first level code, we require only a single additional supercode trellis with 2,884 nodes to represent the code at the second level. Finally, the code at the third level has a trellis with 7,079,886 nodes. Using supercode decoding, we utilize the trellises of the first and second level and require one additional supercode trellis with 2,410 nodes to represent the third code.

With sequential decoding the number of visited nodes in the trellis (the number of iterations) depends on the number

of transmission errors. Note that with the presented codes the time complexity with Algorithm 2 is at most 1.75 times larger than with Algorithm 1.

C. List-of-Two Decoding

Next, we present two techniques to improve the performance and the complexity of Algorithm 1. Firstly, we demonstrate that the soft input decoding can be omitted in cases where the hard decision of the received vector corresponds to a valid codeword. We propose a sequential list-of-two decoding algorithm. List-of-two decoding is motivated by the fact that Algorithm 1 is not a maximum-likelihood decoding procedure. Hence, we may search for further codewords in order to find better candidates than the result of Algorithm 1.

Consider an additive white Gaussian noise channel with binary phase shift keying. A binary code symbol $v_t \in \mathbb{F}_2$ is mapped to the transmission symbol $x_t \in \{+1, -1\}$ by $x_t = 1-2v_t$. The transmitted symbol vector $\mathbf{x}$ is distorted by a noise vector $\mathbf{n}$ such that the received sequence is $\mathbf{r} = \mathbf{x}+\mathbf{n}$. The noise vector $\mathbf{n}$ is a vector of independent identically distributed Gaussian random variables with mean zero. Hence,

$$p(r_t \mid x_t = \pm 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(r_t \mp 1)^2}{2\sigma^2}}, \tag{5}$$

where σ^2 denotes the variance of the Gaussian distribution. For this channel, it is common practice to use the quadratic Euclidean distance $d_E^2(\mathbf{x}, \mathbf{r}) = \sum_{t=1}^{n} |x_t - r_t|^2$ as metric, because

$$\arg\left(\max_{\mathbf{v}\in\mathcal{C}} P(\mathbf{r}|\mathbf{v})\right) = \arg\left(\min_{\mathbf{v}\in\mathcal{C}} d_E^2(\mathbf{x}, \mathbf{r})\right). \tag{6}$$

However, we have

$$d_E^2(\mathbf{x}, \mathbf{r}) = \sum_{t=1}^{n} x_t^2 - 2\sum_{t=1}^{n} x_t r_t + \sum_{t=1}^{n} r_t^2 \tag{7}$$

Let $\tilde{r}_t = \mathrm{sgn}(r_t)$ denote the sign, i.e., the hard decision, of r_t. Using

$$\sum_{t=1}^{n} x_t r_t = \sum_{t=1}^{n} |r_t| - 2 \sum_{t:\ x_t \neq \tilde{r}_t} |r_t| \tag{8}$$

we obtain

$$d_E^2(\mathbf{x}, \mathbf{r}) = n + 4 \sum_{t:\ x_t \neq \tilde{r}_t} |r_t| - 2\sum_{t=1}^{n} |r_t| + \sum_{t=1}^{n} r_t^2 \tag{9}$$

Note that $\sum_{t:\ x_t \neq \tilde{r}_t} |r_t|$ is the only term in (9) that depends on $\mathbf{x}$. Consequently, instead of minimizing the quadratic Euclidean distance we may also minimize $\sum_{t:\ x_t \neq \tilde{r}_t} |r_t|$. Note that $\sum_{t:\ x_t \neq \tilde{r}_t} |r_t| = 0$ if the vector $\tilde{\mathbf{r}} = (\tilde{r}_1, \ldots, \tilde{r}_n)$ corresponds to a valid codeword. Hence, in this case, $\tilde{\mathbf{r}}$ is the maximum-likelihood estimate.

Now we consider list-of-two decoding. In order to enable a trade-off between performance and complexity, we introduce a threshold ρ for the metric of the estimated codeword.

In Algorithm 3 we apply Algorithm 1 to decode the inner codes at the first level, i.e., the codewords of the code $\mathcal{B}^{(0)}$, whereas we apply Algorithm 2 for the lower levels. Figure 3 presents the performance of Algorithm 1 and Algorithm 3 with respect to the residual word error rate (WER) for transmission over the AWGN channel. The code is a one error correcting binary BCH code of length $n = 60$. This code is later-on used as inner code in the first level of the GC code. The decoding performance and the number of decoding iterations depend on the threshold ρ. Figure 4 presents a comparison with respect to the number of iterations, where we have used two different threshold values denoted by ρ_1 and ρ_2, respectively. The values of ρ_2 were obtained by computer search in order to minimize the word error rate for a given signal to noise ratio. The values of ρ_1 were chosen to demonstrate that Algorithm 3 can reduce the word error rate compared with Algorithm 1 with a similar complexity.

Algorithm 3 Sequential List-of-Two Decoding

Data: received word $\mathbf{r}$, threshold ρ
Result: estimated codeword $\hat{\mathbf{v}}$
if $\tilde{\mathbf{r}}$ *corresponds to a valid codeword* **then**
  return the codeword $\hat{\mathbf{v}}$ corresponding to $\tilde{\mathbf{r}}$;
else
  calculate a first estimate $\mathbf{v}_1$ using either Algorithm 1 or Algorithm 2;
  if $M(\mathbf{r}, \mathbf{v}_1) \geq \rho$ **then**
    return the codeword $\hat{\mathbf{v}} = \mathbf{v}_1$;
  else
    remove $\mathbf{v}_1$ from the stack;
    calculate a second estimate $\mathbf{v}_2$ using either Algorithm 1 or Algorithm 2;
    if $M(\mathbf{r}, \mathbf{v}_1) \geq M(\mathbf{r}, \mathbf{v}_2)$ **then**
      return $\hat{\mathbf{v}} = \mathbf{v}_1$;
    else
      return $\hat{\mathbf{v}} = \mathbf{v}_2$;
    end
  end
end

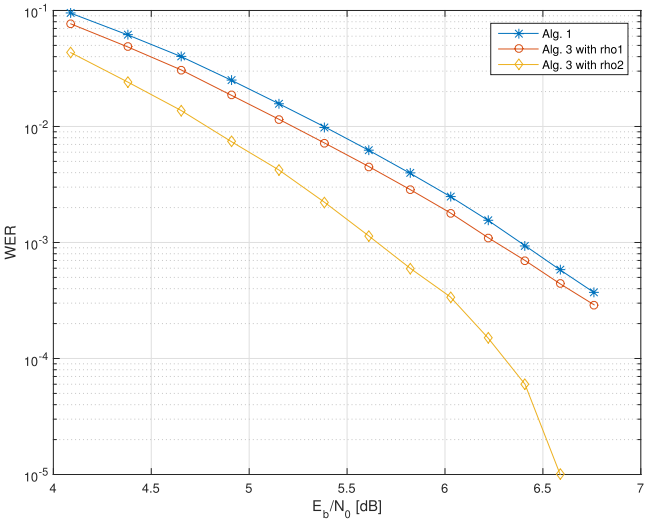


Fig. 3. Comparison of Algorithm 1 and Algorithm 3 with respect to the residual word error rate (WER).

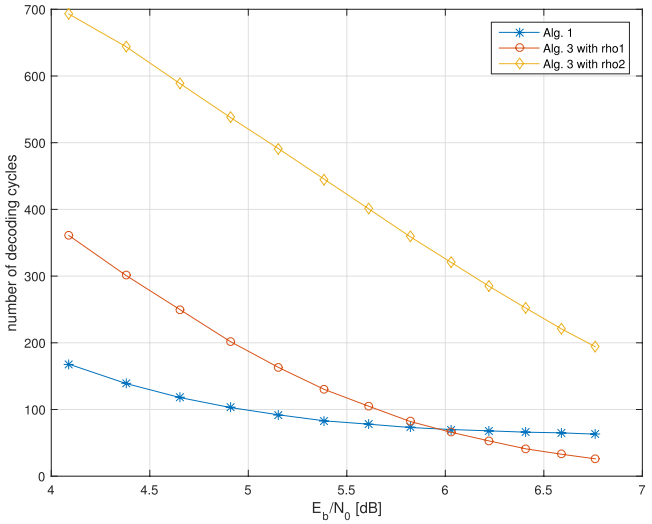


Fig. 4. Comparison of Algorithm 1 and Algorithm 3 with respect to the number of iterations.

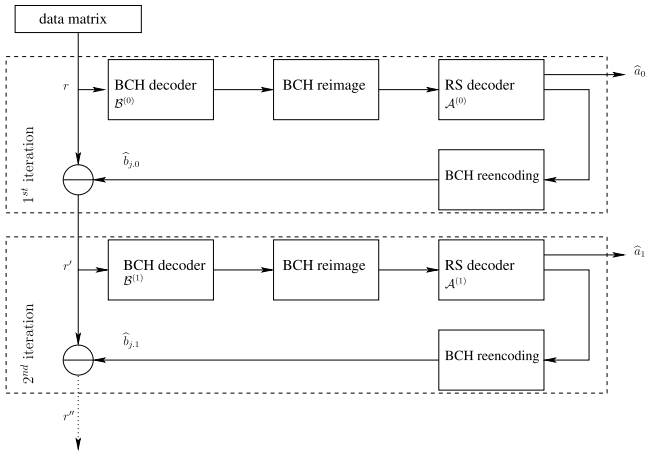


Fig. 5. GC decoding schemes.

IV. GC Decoding and Decoding Error Probability

The decoder processes level by level starting with $i = 0$. Fig. 5 depicts the decoding steps. Let i be the index of the current level. First the columns are decoded with respect to $\mathcal{B}^{(i)}$ and the information bits have to be inferred (re-image) in order to retrieve the code symbols $a_{j,i}$ of $\mathcal{A}^{(i)}$ where j the column index. If all symbols of the code $\mathcal{A}^{(i)}$ are inferred the RS code can be decoded. At this point a partial decoding result $\hat{a}_i$ is available. Finally this result has to be re-encoded using $\mathcal{B}^{(i)}$. The estimated codewords of the inner code $\mathcal{B}^{(i)}$ are subtracted from the codeword matrix before the next level can be decoded. The detailed encoding and hard input decoding process is described in [35].

In the first level $i = 0$ we use soft input decoding according to Algorithm 1. Starting with the second level, we exploit the structure of the nested-BCH codes and use Algorithm 2, where the code at level $i - 1$ can be used as supercode of the code of level i. For the implementation, we limit the number of decoding iterations for each inner code. If the number of iterations exceeds a threshold a decoding failure is declared. For the outer RS codes we employ error and erasure decoding [36], where the decoding failures of the inner codes are regarded as erased symbols of the RS code.

In the following, we present an analysis of the probability of a decoding error for the GC decoder. Afterwards, we present an example that illustrates the performance of the proposed decoding procedure.

A. Probability of a Decoding Error

The performance of the soft input decoding of the inner codes can be determined using Monte Carlo simulation. Let $P_{b,i}$ be the error probability for the decoding of the inner code $\mathcal{B}^{(i)}$. Furthermore, let $P_{e,i}$ be the corresponding probability of a decoder failure. We bound the probability of a decoding error with the multi-stage decoding algorithm.

Let $T_i = n_a - k_{a,i}$ be the number of redundancy symbols for the outer RS code $\mathcal{A}^{(i)}$ at the i-th level. The probability $P_{a,i}$ of a decoding error with error and erasure decoding at the i-th level can be computed as follows [36]:

$$P_{a,i} = \sum_{q=0}^{T_i} \sum_{t=\lfloor \frac{T_i - q}{2} \rfloor + 1}^{n_a - q} P_q \binom{n_a - q}{t} P_{b,i}^t (1 - P_{b,i})^{n_a - q - t} + \sum_{q=T_i+1}^{n_a} P_q \tag{10}$$

where P_q is the probability of q erasures

$$P_q = \binom{n_a}{q} P_{e,i}^q (1 - P_{e,i})^{n_a - q}. \tag{11}$$

Using the union bound, we can estimate the block error rate P_{GC} for the GC code, i.e., the likelihood of the event that at least one level is in error

$$P_e \leq \sum_{i=0}^{L-1} P_{a,i}. \tag{12}$$

Example 6: Consider the code from Example 1. This code has a code rate $R = 0.806$ and was designed to guarantee $P_e \leq 10^{-16}$ according to (12) for $\frac{E_b}{N_0} \geq 4.7dB$, where soft input decoding is used in the first three levels and hard input decoding in the remaining levels.

B. Comparison Error Correction Performance

We compare the error correction performance of the GC code in different decoding modes with the performance of long BCH codes with hard input decoding. As performance measure, we use the code rate that is required to guarantee for a given signal to noise ratio an overall word error rate less than 10^{-10} or 10^{-16}, respectively. All codes are constructed similar to the code presented in Example 1. In particular, the inner codes are chosen according to Table I. Whereas the error correcting capability of the outer codes are adapted to obtain the highest possible code rate for a given signal to noise ratio. Note that in this example, the overall code rate of the GC code is at most $R = 0.9$, because of the choice of the inner code.

Fig. 6 depicts the code rate versus the signal to noise ratio for $P_e = 10^{-10}$, whereas the results for $P_e = 10^{-16}$ are

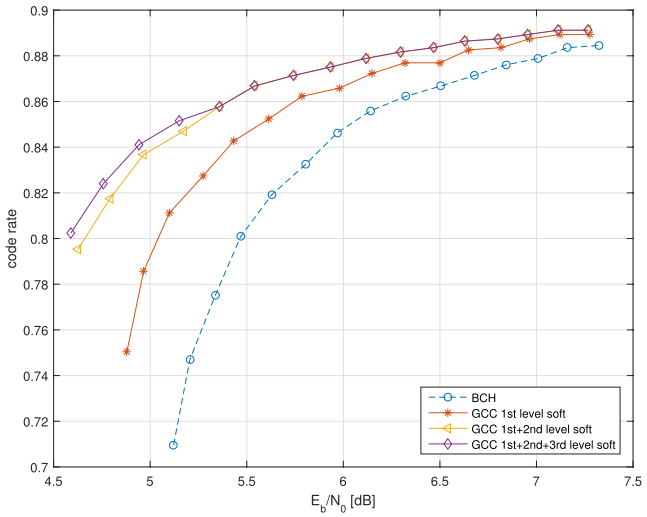


Fig. 6. Code rate versus signal to noise ratio for $P_e = 10^{-10}$.

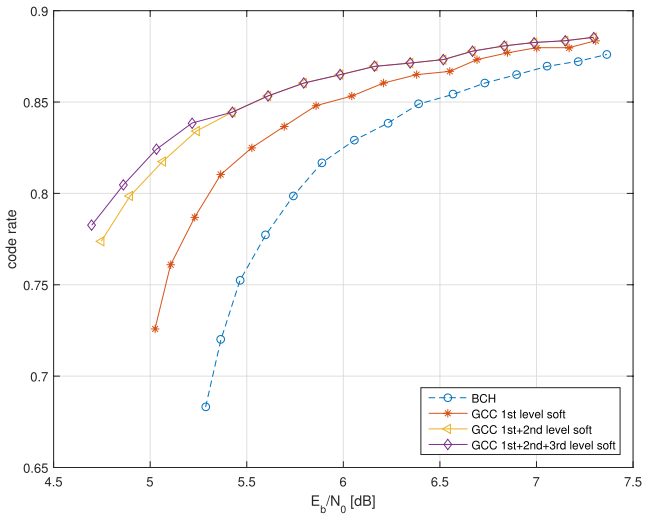


Fig. 7. Code rate versus signal to noise ratio for $P_e = 10^{-16}$.

presented in Fig. 7. The GC code with soft input decoding outperforms the GC code with hard input decoding for all error probabilities and the BCH code for code rates below 0.88. The soft input decoding was simulated with a 3-bit quantization. The three curves with soft input decoding use different decoding strategies, where the soft input decoding is applied only to the first level, first and the second level, or levels 0 to 2, respectively. The soft input decoding improves the performance by up to 1.3 dB. For instance, the GC code with code rate $R = 0.8$ achieves a block error rate less than 10^{-16} at a signal to noise ratio of $E_b/N_0 = 4.7$ dB which is only 1 dB from the channel capacity of the quantized AWGN channel. For $P_e = 10^{-10}$, the code rate $R = 0.8$ is sufficient for a signal to noise ratio $E_b/N_0 \geq 4.6$ dB. Note that soft input decoding of the first and second level is sufficient for all SNR values above $E_b/N_0 = 5.5dB$.

V. Decoder Architecture

In this section we propose a decoder architecture for a GC soft input decoder. First we discuss the integration of the stack algorithm as inner decoder into the implementation of the GC decoder presented in [35]. Then the stack algorithm implementation for supercode decoding with its subsystems is presented and discussed.

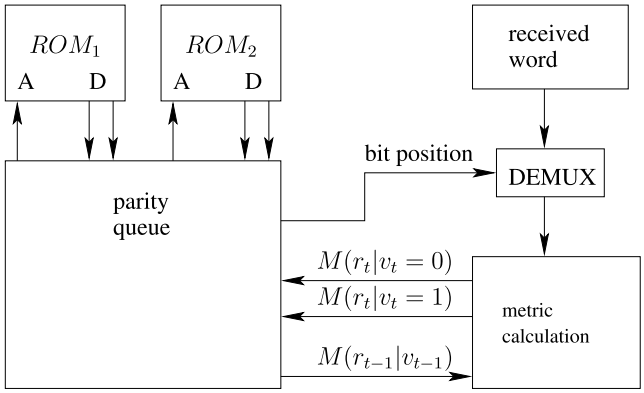


Fig. 8. Block diagram of the sequential decoder.

The original hard input GC decoder implementation in [35] uses algebraic syndrome decoding. In this implementation the first levels of $\mathcal{B}$ can decode $t_{b,0} = 1$ and $t_{b,1} = 2$ errors. Thus high error correction capabilities of the outer codes $\mathcal{A}^{(0)}$ and $\mathcal{A}^{(1)}$ are required. This leads to lower code rates and a high decoding complexity of those outer codes. On the other hand the soft decoding complexity of the column codes increases significantly with each code level. Hence soft decoding is of interest for the lower levels.

Subsequently the algebraic decoding logic for the column code remains in the implementation. Therefore it is possible to check whether the syndrome is zero. In this case the codeword can be assumed to be correct, i.e., neither algebraic decoding nor sequential decoding result in a different codeword.

A. Decoding Logic

A brief system overview is depicted in Fig. 8. The system consists of a word array of size n_b and a desired width which stores the q-ary word. Furthermore a demultiplexer selects the currently processed bit position depending on the top path of the stack and delivers this value to the metric calculator. Based on the received codeword symbol r_i and the previous metric $M(r_{t-1}|v_{t-1})$ the metric module returns $M(r_t|v_t)$ to the priority queue block. To represent the supercode trellis asynchronous ROM is used. Each word of the ROM represents a trellis node $\sigma_{t,i}$. The data consists of two pointers for the successor nodes $v_{t+1} = 0$ and $v_{t+1} = 1$.

Depending on the top entry of the priority queue the desired codeword symbol is selected and the next branches for the actual nodes $\sigma_{t,1}$ and $\sigma_{t,2}$ are loaded from the trellis ROM. The priority queue unloads the top entry and loads the new paths in a single clock cycle.

Each entry of the priority queue contains several elements. The first element is the metric value. The path in the trellis, the length of the path, and a pointer to the current node are stored. All entries have to be ordered by the metric values such that the top entry has the highest value.

The process of the priority queue starts with its initialization. The starting node, its initial metric value and the path length are set. Each update cycle begins with the load phase in which the next node pointers are loaded from the trellis ROM. Simultaneously the next codeword symbol is loaded based on the path length index. The next metric value can be determined based on the code symbol and the available branches.

With binary codes there exists at least one possible branch and at most two branches. The resulting branches are presorted using combinatorial logic. In the following we call these two entries the major and the minor entries, where the major entry has the better metric value.

All priority queue elements are successively ordered in a chain. Each element can exchange its date with its previous or next neighbour (see Fig. 9). Furthermore each element can decide whether it keeps its own data, take the data from its neighbor, load the new major data or the new minor data. In each element the metric value is compared with the new value. The result of this comparison is signaled to its predecessor and successor elements. If the signal of a predecessor is false and the major metric value comparator gives a positive signal the new major value will be stored. Likewise if an element receives a false signal from its successor and the minor metric value comparator signals a new metric value that is less than the current value, the new minor data is stored. In the case that an element receives a signal from its neighbors, space for the new data has to be created by shifting all entries to next element.

There exist two special cases that have to be taken into account. The first special case occurs if a node has only a single outgoing branch. In this case the shifting of elements has to be prevented by signalling. The second special case occurs if the new major and the new minor elements are designated to be inserted into the same entry register. This case can be detected and preventing by passing this value to the next element.

The algorithm terminates if the maximum possible path length is reached. The stored path in the top element is the decoded codeword. In the practical implementation an iteration counter will terminate after a determined maximum number of iterations. This abort can be used to mark this decoded GCC column as a erasure symbol for the outer RS code.

In order to decode supercodes, the following extensions have to be implemented. First for each supercode a distinct ROM is needed which represents its trellis. The metric calculation has to take all trellis branches of each supercode into account. Furthermore, all node pointers have to be stored in the priority queue elements.

B. Area Comparison and Decoding Speed

In this section we present an FPGA implementation of the proposed soft input decoder and compare it with the hard input decoder presented in [35]. The hard input decoder uses algebraic decoding. It consists of the syndrome calculation, the Berlekamp–Massey algorithm (BMA), and the Chien search module. The soft input decoder is implemented as proposed in Section III-B. It has two limitations. First, the length of the priority queue is limited to 64 elements. Furthermore the accuracy of the metric calculation is limited to 16 bits and we use 3-bit quantization of the input symbols.

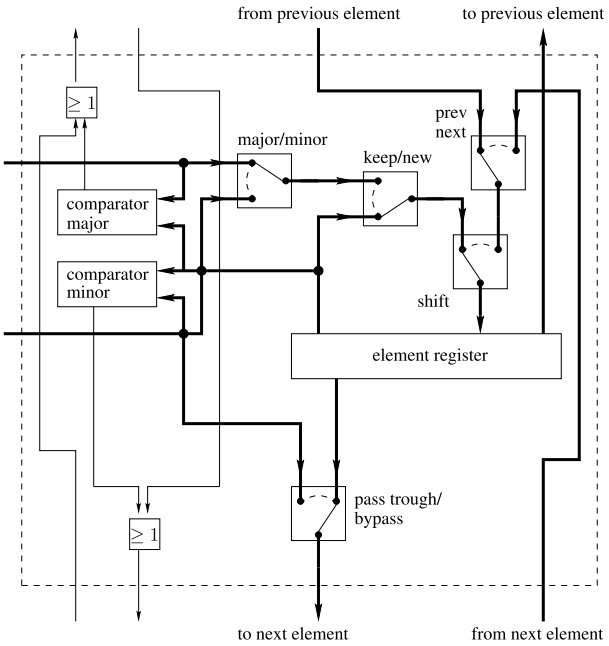


Fig. 9. Priority queue element.

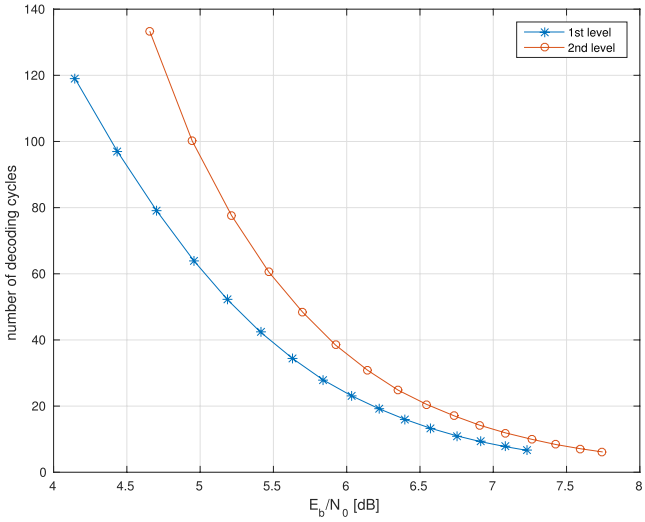


Fig. 10. Average number of iterations for the first and second level.

The stack algorithm has a variable execution time depending on the error pattern. This algorithm needs at least 61 cycles to traverse the entire trellis if no error occurred. This case can be omitted by checking whether the syndrome of a column word is zero. If no error is detected the soft decoding can be avoided and thus only a single cycle is needed.

Fig. 10 depicts the average number of cycles needed for the stack algorithm. It shows the dependency between the channel bit error rate and the computational complexity, i.e., fewer errors lead to fewer decoding cycles. Note that the algebraic hard-input decoder needs four cycles for the first and six cycles

TABLE II
RESULTS OF AN FPGA SYNTHESIS

Module	LUT	Slices
RS module (t=78)		
Syndrome	1 701	1 395
BMA	21 701	6 662
Forney alg.	1 046	729
Chien search	854	712
BCH Module (n=60,t=8)		
Syndrome	184	46
BMA	2 006	732
Chien search	1 557	240
reimage	148	336
Total	**29 197**	**10 852**
Stack alg.	**23 896**	**9 885**

for the second level. Hence, for high signal to noise ratios the stack algorithm requires an average number of cycles that is comparable with hard-input decoding.

Next we present FPGA synthesis result for the stack algorithm. The synthesis was performed with Xilinx Vivado and a Virtex-7 target device. Table II shows the number of slices and look-up tables (LUT) of the hard input and the soft input decoder with 3-bit quantization. From these results we observe that the number of logic elements required for the stack algorithm is about 82% of the number of logic gates required for the GC hard input decoder.

VI. CONCLUDING REMARKS AND FUTURE DIRECTIONS

In this work we have presented a soft input decoder for generalized concatenated codes. We have proposed a sequential decoding algorithm based on supercode trellises in order to reduce the complexity of the soft input decoding compared to ordinary trellis based decoding. This implementation improves the error correction capability significantly compared with hard input decoding. Consequently, the proposed method is a promising approach for soft input decoding of GC codes. The implementation of the stack algorithm is nine times large than the algebraic decoder. This complexity can be reduced by moving the majority of the register entries into read only memory. Nevertheless, the proposed soft input decoding increases the overall complexity of the GC decoder by 82% compared to the decoder presented in [7]. Other soft input decoding methods for binary BCH codes may reduce the overall decoding complexity.

The proposed coding scheme is well suited for applications that require very low residual error rates, e.g., it might be appropriate for flash memories that provide soft information about the state of the memory cells. For flash memories, various concatenated coding schemes were proposed in order to enable soft input decoding, e.g., product codes [37] and concatenated coding schemes based on trellis coded modulation and outer BCH oder RS codes [31], [38], [39]. For the presented simulation results we assumed a quantized additive white Gaussian noise channel. For practical flash memories, this model is not accurate. With multi-level cell and triple-level cell technologies, the reliability of the bit levels and cells varies and coding schemes were proposed that take these error characteristics into account [40]–[43]. We believe that the adaptation of the proposed GC coding scheme for flash memories is a promising direction for further research.

REFERENCES

[1] A. Fahrner, H. Griesser, R. Klarer, and V. V. Zyablov, "Low-complexity GEL codes for digital magnetic storage systems," *IEEE Trans. Magn.*, vol. 40, no. 4, pp. 3093–3095, Jul. 2004.

[2] J. Freudenberger, U. Kaiser, and J. Spinner, "Concatenated code constructions for error correction in non-volatile memories," in *Proc. Int. Symp. Signals, Syst., Electron. (ISSSE)*, Potsdam, Germany, Oct. 2012, pp. 1–6.

[3] J. Freudenberger, J. Spinner, and S. Shavgulidze, "Generalized concatenated codes for correcting two-dimensional clusters of errors and independent errors," in *Proc. Int. Conf. Commun. Signal Process. (CSP)*, Barcelona, Spain, Feb. 2014, pp. 1–5.

[4] I. Dumer, "Concatenated codes their multilevel generalizations," in *Handbook of Coding Theory*, vol. 2. Amsterdam, The Netherlands: Elsevier, 1998.

[5] M. Bossert, *Channel Coding for Telecommunications*. New York, NY, USA: Wiley, 1999.

[6] V. Zyablov, S. Shavgulidze, and M. Bossert, "An introduction to generalized concatenated codes," *Eur. Trans. Telecommun.*, vol. 10, no. 6, pp. 609–622, 1999.

[7] J. Spinner and J. Freudenberger, "Decoder architecture for generalised concatenated codes," *IET Circuits, Devices Syst.*, vol. 9, no. 5, pp. 328–335, 2015.

[8] A. Neubauer, J. Freudenberger, and V. Kühn, *Coding Theory: Algorithms, Architectures and Applications*. New York, NY, USA: Wiley, 2007.

[9] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.

[10] C. Argon, S. W. McLaughlin, and T. Souvignier, "Iterative application of the Chase algorithm on Reed–Solomon product codes," in *Proc. IEEE ICC*, Jun. 2001, pp. 320–324.

[11] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inf. Theory*, vol. IT-41, no. 5, pp. 1379–1396, Sep. 1995.

[12] B. Dorsch, "A decoding algorithm for binary block codes and J-ary output channels (corresp.)," *IEEE Trans. Inf. Theory*, vol. 20, no. 3, pp. 391–394, May 1974.

[13] M. Tomlinson, C. Tjhai, and M. Ambroze, "Extending the Dorsch decoder towards achieving maximum-likelihood decoding for linear codes," *IET Commun.*, vol. 1, no. 3, pp. 479–488, Jun. 2007.

[14] A. Gortan, R. P. Jasinski, W. Godoy, and V. A. Pedroni, "Achieving near-MLD performance with soft information-set decoders implemented in FPGAs," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Dec. 2010, pp. 312–315.

[15] L. E. Hguado and P. G. Farrell, "On hybrid stack decoding algorithms for block codes," *IEEE Trans. Inf. Theory*, vol. 44, no. 1, pp. 398–409, Jan. 1998.

[16] J. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inf. Theory*, vol. 24, no. 1, pp. 76–80, Jan. 1978.

[17] J. Freudenberger, T. Wegmann, and J. Spinner, "An efficient hardware implementation of sequential stack decoding of binary block codes," in *Proc. IEEE 5th Int. Conf. Consum. Electron. (ICCE-Berlin)*, Berlin, Germany, Sep. 2015, pp. 135–138.

[18] J. Freudenberger and M. Bossert, "Maximum-likelihood decoding based on supercodes," in *Proc. 4th. Int. ITG Conf. Source Channel Coding*, Erlangen, Germany, Jan. 2004, pp. 185–190.

[19] J. Freudenberger, *Bounded Distance Decoding And Decision Feedback*. Düsseldorf, Germany: VDI Verlag, 2004.

[20] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 7, pp. 872–877, Jul. 2005.

[21] R. Micheloni, A. Marelli, and R. Ravasio, *Error Correction Codes for Non-Volatile Memories*. Rotterdam, The Netherlands: Springer, 2008.

[22] J. Freudenberger and J. Spinner, "A configurable Bose–Chaudhuri–Hocquenghem codec architecture for flash controller applications," *J. Circuits, Syst., Comput.*, vol. 23, no. 2, pp. 1–15, Feb. 2014.

[23] S.-G. Cho, D. Kim, J. Choi, and J. Ha, "Block-wise concatenated BCH codes for NAND flash memories," *IEEE Trans. Commun.*, vol. 62, no. 4, pp. 1164–1177, Apr. 2014.

[24] D. Kim and J. Ha, "Quasi-primitive block-wise concatenated BCH codes for NAND flash memories," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2014, pp. 611–615.
[25] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in NAND flash memory," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 2, pp. 429–439, Feb. 2011.
[26] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives," presented at the part 11th USENIX Conf. File Storage Technol. (FAST). San Jose, CA, USA, 2013, pp. 243–256. [Online]. Available: https://www.usenix.org/conference/fast13/technical-sessions/presentation/zhao
[27] J. Wang *et al.*, "Enhanced precision through multiple reads for LDPC decoding in flash memories," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 880–891, May 2014.
[28] W. Lin *et al.*, "A low power and ultra high reliability LDPC error correction engine with digital signal processing for embedded NAND flash controller in 40 nm COMS," in *Symp. VLSI Circuits Dig. Tech. Papers*, Jun. 2014, pp. 1–2.
[29] K. Haymaker and C. A. Kelley, "Structured bit-interleaved LDPC codes for MLC flash memory," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 870–879, May 2014.
[30] *Solid-State Drive (SSD) Requirements Endurance Test Method*, document JESD218, JEDEC Solid State Technology Association, 2010.
[31] S. Li and T. Zhang, "Improving multi-level NAND flash memory storage reliability using concatenated BCH-TCM coding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 10, pp. 1412–1420, Oct. 2010.
[32] S. Qi, D. Feng, and J. Liu, "Optimal voltage signal sensing of NAND flash memmory for LDPC code," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2014, pp. 1–6.
[33] J. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 196–198, Jan. 1972.
[34] V. Sorokine and F. R. Kschischang, "A sequential decoder for linear block codes with a variable bias-term metric," *IEEE Trans. Inf. Theory*, vol. 44, no. 1, pp. 410–416, Jan. 1998.
[35] J. Spinner and J. Freudenberger, "Design and implementation of a pipelined decoder for generalized concatenated codes format," in *Proc. 27th Symp. Integr. Circuits Syst. Design (SBCCI)*, Aracaju, Brazil, Sep. 2014, pp. 1–16.
[36] L. Weiburn and J. K. Cavers, "Improved performance of Reed–Solomon decoding with the use of pilot signals for erasure generation," in *Proc. 48th IEEE Veh. Technol. Conf. (VTC)*, vol. 3. May 1998, pp. 1930–1934.
[37] C. Yang, Y. Emre, and C. Chakrabarti, "Product code schemes for error correction in MLC NAND flash memories," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 12, pp. 2302–2314, Dec. 2012.
[38] F. Sun, S. Devarajan, K. Rose, and T. Zhang, "Design of on-chip error correction systems for multilevel NOR and NAND flash memories," *IET Circuits, Devices Syst.*, vol. 1, no. 3, pp. 241–249, Jun. 2007.
[39] J. Oh, J. Ha, J. Moon, and G. Ungerboeck, "RS-enhanced TCM for multilevel flash memories," *IEEE Trans. Commun.*, vol. 61, no. 5, pp. 1674–1683, May 2013.
[40] E. Yaakobi, J. Ma, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf, "Error characterization and coding schemes for flash memories," in *Proc. IEEE GLOBECOM Workshops*, Dec. 2010, pp. 1856–1860.
[41] E. Yaakobi, L. Grupp, P. Siegel, S. Swanson, and J. K. Wolf, "Characterization and error-correcting codes for TLC flash memories," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Jan. 2012, pp. 486–491.
[42] R. Gabrys, E. Yaakobi, and L. Dolecek, "Graded bit-error-correcting codes with applications to flash memory," *IEEE Trans. Inf. Theory*, vol. 59, no. 4, pp. 2315–2327, Apr. 2013.
[43] R. Gabrys, F. Sala, and L. Dolecek, "Coding for unreliable flash memory cells," *IEEE Commun. Lett.*, vol. 18, no. 9, pp. 1491–1494, Sep. 2014.

Jens Spinner received the B.Sc. degree in computer science and the M.Sc. degree in computer science from the HTWG-Konstanz in 2009 and 2011, respectively. He is currently pursuing the Ph.D. degree with the Institute of System Dynamics, HTWG Konstanz. His main research interests are concatenated codes and coding for non-volatile memories.

Jürgen Freudenberger (S'99–M'04) received the Dipl-Ing. and Ph.D. degrees in electrical engineering from the University of Ulm, Germany, in 1999 and 2004, respectively.

In 2003, he joined the Daimler-Chrysler Research Center, Ulm, where he was involved in signal processing for automatic speech recognition. Since 2006, he has been a Professor with the Hochschule Konstanz University of Applied Sciences, Germany, where he is currently the Head of the Information and Media Center. His research interests include coding theory for communication as well as storage systems and adaptive systems for speech processing. He has authored over 90 papers on these subjects. He has co-authored the book entitled *Coding Theory: Algorithms, Architectures and Applications* (John Wiley & Sons, 2007) with A. Neubauer and V. Kühn.

Dr. Freudenberger is a member of the German Association for Electrical, Electronic/Information Technologies (ITG) and received the ITG Award for his dissertation in 2005.

Sergo Shavgulidze received the Diploma degree with excellence in communication engineering from the Georgian Technical University, Tbilisi, in 1980, the Candidate of Tech. Scien. degree from the Institute for Control Problems, Moscow, in 1984, and the Doctor of Techn. Scien. degree from the Institute for Information Transmission Problems, Moscow, in 1991. Since 1980, he has been with the Georgian Technical University, where he is currently a Full Professor with the Department of Telecommunications. Since 2004, he has also been with the Georgian National Communications Commission, where he is currently the Head of the Radio Frequency Management Department. From 2007 to 2008, he was the Associate Member on the Information and Communications Security Panel under the NATO Science for Peace and Security Program. He headed his country's delegation to RA-2007, WRC-2007 and RA-2015, WRC-2015, and was elected as the Vice-Chairman of ITU-R Study Group 5 (Terrestrial Services) at the RA-2015. His research interests include coding theory and communication systems with a special emphasis on (generalized) concatenated codes, woven codes, coded continuous phase modulation, and space-time coding. He has authored over 140 papers on these subjects. He has co-authored the book entitled *Generalized Concatenated Constructions on a Base of Convolutional Codes* (Moscow, Russia: Nauka, 1991) with V. Zyablov. On leave from Georgian Technical University, he held different research positions at Linköping University and Lund University in Sweden; Darmstadt Technical University, Konstanz University of Applied Sciences, and Ulm University in Germany; Technical University of Denmark in Lyngby, Denmark; and Lancaster University and HW Communications Ltd., U.K.