Hardware Implementation of a Reed-Solomon Soft Decoder based on Information Set Decoding

Stefan Scholl, Norbert Wehn
Microelectronic Systems Design Research Group
University of Kaiserslautern
67663 Kaiserslautern, Germany
Email: {scholl, wehn}@eit.uni-kl.de

***Abstract*—Soft decision decoding of Reed-Solomon codes can largely improve frame errors rates over currently used hard decision decoding. In this paper, we present a new hardware implementation for soft decoding of Reed-Solomon codes based on information set decoding. To our best knowledge this is the first hardware implementation of information set decoding for long Reed-Solomon codes. We propose a reduced complexity version of the decoding algorithm, that is optimized for efficient hardware implementation and enables high throughput. The decoder was implemented on a Virtex 7 FPGA, achieving a gain of 0.75 dB compared to conventional hard decision decoding and a throughput of up to 1.19 GBit/s for the widely used RS(255,239). This gain in FER is achieved with less complexity and more than 15x larger throughput than other state-of-the-art architectures.**

I. Introduction

Forward error correction is an essential component of today's communication systems. In many applications and communication standards Reed-Solomon (RS) codes are utilized, either as a standalone channel code or in concatenation with convolutional codes, e.g. [2]. In this paper, we will put our main focus on the RS(255,239) code, because of its capital importance for real world communication systems.

Traditionally Reed-Solomon codes are decoded with a hard decision decoder (HDD), using e.g. the well known Berlekamp-Massey algorithm [17]. In contrast to hard decision decoding, *soft decision* decoding uses probabilistic information on the received bits, which can lead to large improvements in frame error rate (FER) over HDD [8].

Soft decoding for block codes gained a lot of attraction in the last 20 years driven by Turbo and LDPC codes [11]. These codes have previously demonstrated the advantage of soft decoding. RS codes are mainly hard decoded with HDD, because the decoding heuristics for Turbo and LDPC codes can not directly be applied to RS codes (due to the dense structure of RS codes). However recently soft decoding heuristics have been proposed using different approaches, that exhibit very different FER and complexity. A selection of interesting algorithms can be found in [5], [6], [8], [9], [14], [16], [19].

In contrast to Turbo and LDPC codes, hardware implementations of soft decoding of RS codes are yet not well investigated. Therefore, it is so far not obvious which algorithm leads to efficient hardware implementations, that feature good FER gain over HDD and high throughput. This is important, because real world applications often require high throughputs, that can not be provided by software implementations. Hence, dedicated hardware implementations of decoders on FPGAs or ASICs are mandatory to meet throughput requirements.

Up to now, few hardware implementations on ASICs and FPGAs have been proposed for soft decoding of the RS(255,239) code. One trend becoming apparent are implementations based on Chase decoding [6], especially the closely related low-complexity chase (LCC) algorithm [5]. These hardware implementations feature low hardware complexity [4] [10] [13]. However, this low complexity comes at the expense of a rather poor FER. Implementations based on LCC provide only little FER gain over HDD in the order of $0.3-0.4$ dB.

Another trend are implementations, that provide larger gains in FER. In [15] a FPGA implementation of a soft decoder for a (shortened) RS(255,239) was presented based on the adaptive belief propagation (ABP) algorithm [14]. It features 0.75 dB gain ($@FER = 10^{-4}$) and a throughput of 32 MBit/s on a Stratix II FPGA. Another implementation based on the stochastic Chase algorithm in [12] features 0.7 dB gain ($@FER = 10^{-4}$) at a throughput of 50 MBit/s on a Virtex 5 FPGA. Although the implementations based on ABP [15] and the stochastic Chase algorithm [12] feature a comparatively large gain in FER, they achieve only small throughputs. So it is desirable to look for new algorithms, that provide a more efficient approach and lead to implementations with a large FER gain and high throughput. Hardware implementations that provide larger gains in FER *and* high throughput are yet not known.

In this paper, we present a new hardware implementation based on a different algorithmic approach, that was so far not considered for RS codes. We implement a decoder based on information set decoding [7] called ordered-statistics decoding (OSD) [9] in a reduced complexity version (a modificated version of [3]). To our best knowledge this is the first implementation of information set decoding for long Reed-Solomon codes. Instead of implementing the OSD algorithm straightforward, we consider a simplified version, which considerably reduces the hardware complexity. We present a FPGA implementation for the RS(255,239) and RS(63,55) code. For the RS(255,239) we achieve a gain of 0.75 dB at a throughput of 1.19 GBit/s. For the RS(63,55) code FER gain is 1.4 dB at a throughput of 690 MBit/s.

The paper is structured as follows: In Section II we will introduce the necessary notation. Section III reviews original

OSD and its reduced complexity version for hardware implementation. The proposed hardware architecture can be found in detail in Section IV. Implementation results for a Xilinx Virtex 7 FPGA and a comparison with other state-of-the-art decoders are presented in Section V.

II. Notation

A binary linear (N, K) code C has N code bits and K information bits. The number of redundant bits is denoted as $M = N - K$, the code rate is $R = \frac{K}{N}$. C is defined by a parity check matrix $\mathbf{H}$ of dimension $M \times N$. h_{ij} denotes the bit in the ith row and the jth column, $\mathbf{h_i}$ denotes the ith column vector of $\mathbf{H}$.

In the assumed transmission system, a binary vector of information bits $\mathbf{u} = (u_0, u_1, ..., u_{K-1})$, $u_i \in GF(2)$ is encoded to obtain a binary code word $\mathbf{x} = (x_0, x_1, ..., x_{N-1})$, $x_i \in GF(2)$, which is modulated using BPSK and transmitted over an additive white Gaussian noise (AWGN) channel. After demodulation the received bits are described as a vector of log likelihood ratios (LLRs) $\mathbf{y} = (y_0, y_1, ..., y_{N-1})$, $y_i \in \mathbb{R}$, with $y_i = ln\left(\frac{p(x_i=0)}{p(x_i=1)}\right)$ Let $\bar{\mathbf{y}}$ be the hard decision vector of $\mathbf{y}$. The reliabilities of the received LLRs are given by $|\mathbf{y}|$. The transmission can be interpreted as an addition of a binary error vector $\mathbf{e}$, such that $\bar{\mathbf{y}} = \mathbf{x} + \mathbf{e}$. The syndrome vector is denoted by $\mathbf{s} = \mathbf{H}\bar{\mathbf{y}}^{\mathbf{T}} = \mathbf{H}\mathbf{e}^{\mathbf{T}}$.

Soft decision maximum likelihood decoding is performed by finding the code word with minimum value of the following metric:

$$W(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{N-1} x_i y_i \qquad (1)$$

$$\mathbf{x}_{ML} = \underset{\mathbf{x} \in C}{argmin}\, W(\mathbf{x}, \mathbf{y}) \qquad (2)$$

Note, that RS codes are non-binary codes, which are defined over a Galois Field (GF) $GF\left(2^P\right)$ of size $q = 2^P$. However, the non-binary code words and parity check matrix can be transformed into the binary domain. To transform the non-binary parity check matrix to its binary counterpart, binary image expansion is used. In binary image expansion the non-binary matrix entries of the original RS parity check matrix are replaced by $P \times P$ binary matrices [8]. E.g. the non-binary RS(255,239) code ($P = 8$) can be described as a binary $(2040, 1912)$ code, having a 128×2040 parity check matrix $\mathbf{H}$. In the remainder of this paper we treat the RS code as a binary code, so the decoding algorithms for binary codes can be applied.

III. Decoding Algorithm

This section investigates the algorithm, which is the basis of our hardware implementation. First, we review the original OSD algorithm, as proposed by Fossorier et al. [9]. Then, we review a reduced complexity version of OSD using syndrome weights [3]. We demonstrate the advantages of this modified OSD for hardware implementations and propose further improvements.

$$\hat{H} = \left[\underbrace{\begin{matrix} 1\,0 \dots 0 \\ 0\,1 \dots 0 \\ \vdots\ \vdots \quad \vdots \\ 0\,0 \dots 1 \end{matrix}}_{\text{LRIPs}} \underbrace{\begin{matrix} u_{0,0} & \dots & u_{0,N-M-1} \\ u_{1,0} & \dots & u_{1,N-M-1} \\ \vdots & & \vdots \\ u_{M-1,0} & \dots & u_{M-1,N-M-1} \end{matrix}}_{\text{MRPs}}\right]$$

Fig. 1. $\mathbf{H}$ after Gaussian elimination with sorted bit positions, resp. columns

A. Original Ordered Statistics Decoding

OSD has been proposed in [9] and belongs to the class of information set decoders, that originate from [7].

In OSD the received bits and their bit positions in $\bar{\mathbf{y}}$ are first sorted according to their reliabilities $|\mathbf{y}|$. The received bit positions are divided into two groups: the M least reliable independent positions (LRIP) and the remaining K more reliable positions (MRPs).

In a first step OSD erases the M LRIPs. Then the remaining K MRPs are used to reconstruct the M erased LRIPs using the M parity check equations. To perform the reconstruction, the parity check matrix has to be put into a diagonalized form $\hat{\mathbf{H}}$ by Gaussian elimination, see Fig.1. This process always outputs a valid code word and is called order-0 reprocessing or OSD(0). OSD(0) corrects all errors in the low reliable part, if the K MRPs are correct.

To perform successful correction in the case of one error in the K MRPs, the reconstruction process is repeated several times, each time with exactly one of the MRPs flipped. This results in a list of $K + 1$ possible code words from which the best code word is chosen by evaluating the metric of Eq. (1). This improved decoding is called order-1 reprocessing or OSD(1).

High order reprocessings, like OSD(2), further improve FER, but are not considered in this paper, because of their computational complexity.

1) ***Sorting:*** *determine M independent bit positions with lowest reliability $|\mathbf{y}|$ (LRIPs)*
2) ***Erasure:*** *erase the M LRIPs*
3) ***Gaussian Elimination:*** *diagonalize $\mathbf{H}$ at the LRIP columns to obtain $\hat{\mathbf{H}}$*
4) ***OSD(0):*** *reconstruct the LRIPs using $\hat{\mathbf{H}}$ to obtain a code word $\mathbf{x}$*
5) ***OSD(1):*** *flip every bit in the MRP once and reconstruct again to obtain K code words $\mathbf{x}$*
6) ***List decoding:*** *among all code words found: select the one with the smallest metric $W(\mathbf{x}, \mathbf{y})$ (Eq. 1)*

Fig. 2. Original OSD algorithm, here with order-1 reprocessing

B. Reduced Complexity OSD for Hardware

The computational bottleneck of the original algorithm are the dynamic reconstructions of the M LRIPs. E.g. in case of decoding a RS(255,239) with OSD(1) this operation is required 2041 times. To overcome this problem we use a reduced complexity algorithm which makes use of the syndrome weight [3].

The reduced complexity algorithm starts (as the original OSD) with determining the LRIPs and Gaussian elimination. However, the reconstruction process of the original OSD is replaced by determining the corrupted bits using the syndrome vector and then flipping these erroneous bits.

If the syndrome vector is calculated using the diagonalized parity check matrix, i.e. $\hat{\mathbf{s}} = \hat{\mathbf{H}}\bar{\mathbf{y}}^{\mathbf{T}}$, two distinct cases for the binary weight of the syndrome vector can be observed:

- The syndrome weight is small: In this case, it can be assumed, that only errors in the LRIPs are present, i.e. OSD(0) processing is sufficient.
- The syndrome weight is large: In this case, it can be assumed, that also errors in the MRP are present. Then OSD(1) processing is performed.

A (fixed) weight threshold to decide between the two cases is denoted by $\Theta \in \mathbb{N}$ and determined by simulation.

OSD(0) (small syndrome weight) is performed by simply flipping the LRIPs that have lead to the 1s in the syndrome vectors. Conducting OSD(1) (large syndrome weight) to correct one MRP error is done by first flipping the bit position

$$j = \underset{i=0,\ldots,N-1}{argmin} \; wgt\left(\hat{\mathbf{s}} \oplus \hat{\mathbf{h}}_{\mathbf{i}}\right)$$

After flipping the MRP error bit at j, the syndrome is calculated again and the remaining LRIPs errors are corrected by performing OSD(0).

We have to point out, that the syndrome weight based OSD does not exactly perform OSD, but merely approximates OSD behaviour. The crucial part is the decision for OSD(0) (only LRIP errors) or OSD(1) (MRP and LRIP errors), which can not always be accomplished correctly by the syndrome weight. However, as can be seen in Fig. 3, this approach works well for the RS(255,239) code and considerably reduces the complexity of OSD. For more detailed information on the syndrome weight OSD please refer to [3].

C. Remarks and Modifications of the Syndrome Weight OSD

In this subsection some characteristics of the reduced complexity approach are discussed and modifications to the algorithm are proposed to reduce hardware complexity and to improve FER.

1) HDD aided decoding: One disadvantage of OSD is the tendency for a weak FER performance if SNR increases. To improve FER we propose to extend OSD with a conventional HDD, which may provide an additional code word for the list decoding. The FER gain for HDD aided decoding is shown in Fig. 3.

2) stopping criterion: The syndrome weight can also be used to decide early in the decoding process, if OSD(0) processing is sufficient or also OSD(1), which is much more time consuming, is necessary. Since in practical cases OSD(1) needs to be executed only very rarely, this feature speeds up decoding considerably.

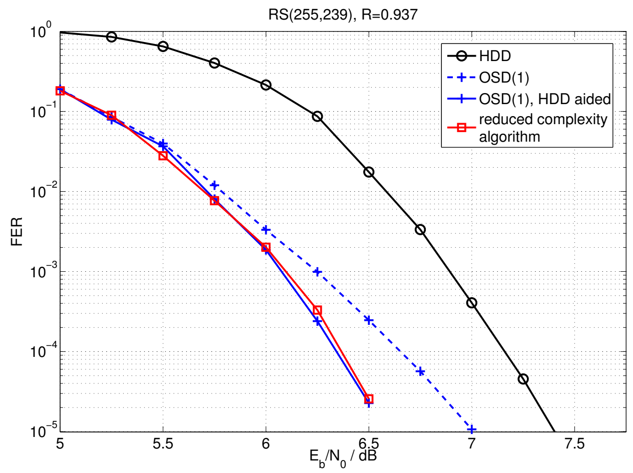

Fig. 3. Algorithmic considerations for the RS(255,239)

3) list decoding: Since OSD is a list decoding algorithm, it is usually required to calculate the sum in Eq. 1 for every candidate code word. The candidate code words include the OSD(0) and the HDD code word as well as K code words from OSD(1). Calculating Eq. 1 is very costly in costly both in time and hardware resources. However here the calculation is avoided by evaluating syndrome weights. In OSD(1) the syndrome weights after the bit flips are immediately evaluated. Among all K OSD(1) candidates the one with the smallest weight is selected (Step 4 in Fig. 4). If finally the syndrome weight falls below the weight threshold Θ additional OSD(0) is assumed to be successful, otherwise OSD is declared as decoding error and the HDD solution is output (Step 5 in Fig. 4). Since the syndrome weight is a sum of M 1 bit values, it is much less complex than evaluating the sum of N (fixed point) LLRs values in Eq. 1.

The reduced complexity OSD algorithm for our hardware implementation is summarized in Fig. 4. An evaluation of the FER performance is shown in Fig. 3.

```
1)  Sorting:
    determine the $M$ least reliable bit positions (LRP)
2)  Gaussian Elimination:
    diagonalize $\mathbf{H}$ at the LRIPs to obtain $\hat{\mathbf{H}}$
3)  calculate the syndrome $\hat{\mathbf{s}} = \hat{\mathbf{H}}\bar{\mathbf{y}}^{\mathbf{T}}$
    and its binary weight $wgt(\hat{\mathbf{s}})$
4)  If $wgt(\hat{\mathbf{s}}) > \Theta$: /* MRP errors */
    flip the received bit at position
    $j = \underset{i=0,\ldots,N-1}{argmin} \; wgt(\hat{\mathbf{s}} \oplus \hat{\mathbf{h}}_{\mathbf{i}})$
    update the syndrome $\hat{\mathbf{s}} = \hat{\mathbf{s}} \oplus \hat{\mathbf{h}}_{\mathbf{j}}$ and $wgt(\hat{\mathbf{s}})$, goto 5
5)  If $wgt(\hat{\mathbf{s}}) \leq \Theta$: /* only LRIP errors remaining */
    For all $\hat{s}_i = 1$, flip the LRIP $l$, for which $\hat{h}_{il} = 1$
    output OSD result, terminate
    else
    perform HDD on $\bar{\mathbf{y}}$ and output HDD result, terminate
```

Fig. 4. Reduced complexity OSD(1) based on the syndrome weight, that we have implemented

IV. Hardware Architecture

In this section we present a hardware architecture based on the previously introduced algorithm in Fig. 4.

Architecture Overview

Fig. 5 shows the overall hardware architecture. P LLRs are fed in parallel into the decoder and stored in the "I/O bit memory". During data input the LLRs are sorted in a parallelized sorter and the low reliable bit positions (LRPs) are stored in the "LRP memory". Simultaneously, the syndrome is calculated and HDD is carried out, whose result is stored in the "HDD memory".

Then the column generator outputs the columns of $\mathbf{H}$ corresponding to the LRPs for Gaussian elimination. These LRP columns are fed into the Gaussian Elimination Unit to dynamically set up the unit (see below). After setting up the Gaussian Elimination Unit, the syndrome $\mathbf{s}$ is put into the elimination unit to obtain $\hat{\mathbf{s}}$.

After determining the initial syndrome, the Correction Unit calculates the syndrome weight and determines the decoding strategy (OSD(0) or OSD(1)). If OSD(1) is executed, the Column Generator outputs every column of $\mathbf{H}$ one after another, which are then transformed into the columns of $\hat{\mathbf{H}}$ by the Gaussian Elimination Unit. Finally, the Correction Unit determines the erroneous bit positions and flips these bits in the "I/O bits memory". Furthermore, the Correction Unit decides if the best OSD code word or the HDD code word is output.

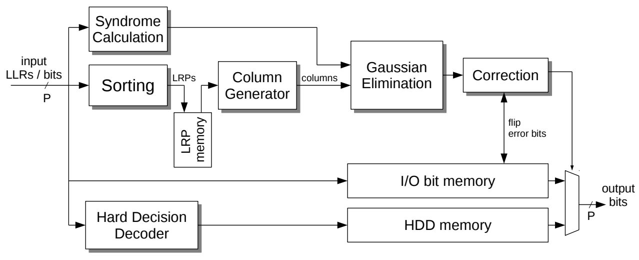

Fig. 5. Decoder architecture overview

Sorting Unit

The first step of decoding is finding the least reliable bit positions. This is accomplished by the Sorting Unit depicted in Fig. 6. Sorting is performed by using a shift register based insertion sort. At every stage the "minimum" registers contain the current minimum reliability value $|y_i|$ together with its bit position. New received LLRs (and their corresponding position value) are shifted through the shift register from left to right and compared with each current minimum. If a new LLR is less than the content of the current "minimum" register the register is updated with this new value.

To reduce the latency for sorting, this shift register is partitioned in P parts. Each part is calculated in parallel. The reduction in latency is achieved without any significant increase in hardware resources. However, the results provided by the parallelized Sorting Unit are not exactly the overall LRPs, but rather an approximation of the LRPs. This introduces a loss in FER, but simulations show, that this loss is less than 0.1 dB (RS(255,239), $P = 8$).

Finally the LRPs are read out of the shift register and stored in the "LRP memory" for further processing.

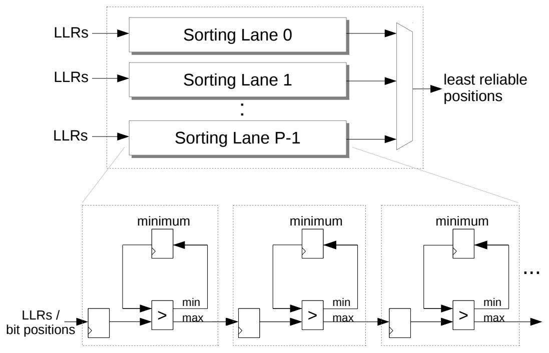

Fig. 6. Parallelized Sorting Unit

Syndrome Calculation Unit

The syndrome is determined by calculating $\mathbf{s} = \mathbf{H}\bar{\mathbf{y}}^{\mathbf{T}}$. For RS codes this calculation can be done using Galois field arithmetic as it is well known in literature [17]. Our syndrome unit is a parallelized implementation, that can handle one received symbol (P bits) per clock cycle.

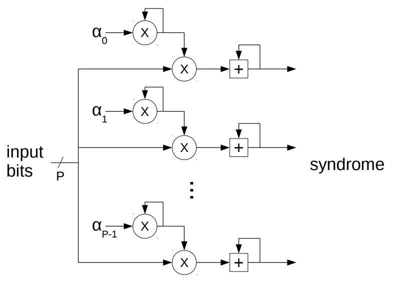

Fig. 7. Syndrome calculation using GF addition and GF multiplication

After the syndrome calculation $\mathbf{s}$ is fed into the Gaussian Elimination Unit to obtain the required $\hat{\mathbf{s}} = \hat{\mathbf{H}}\bar{\mathbf{y}}^{\mathbf{T}}$.

Column Generator Unit

The columns generator consists of a ROM, which holds the original parity check matrix $\mathbf{H}$. The Column Generator accepts a column number at its input and outputs the requested column of $\mathbf{H}$.

Gaussian Elimination Unit

If Gaussian elimination is implemented in a straightforward manner, the parity check matrix is stored in a memory. During Gaussian elimination read and write operations are performed to find the pivot elements and to eliminate the columns step by step. This iterative process is not only time consuming but also involves usually elaborated memory accesses.

A more elegant architecture for Gaussian elimination has been proposed in [18]. This architecture consists of a pipelined array, which eliminates the columns on the fly. The columns of the original matrix $\mathbf{H}$ are input from the left and the corresponding columns of the eliminated matrix $\hat{\mathbf{H}}$ are output at the right. Each of the M column eliminators is responsible for carrying out the operations needed to eliminate exactly one of the M columns corresponding to the LRIPs.

The array works in two phases:

1) The Setup Phase: The M columns of the LRPs are passed into the array to dynamically set up the structure for row adding in the column eliminators.
2) The Elimination Phase: Columns of the original matrix are passed into the array. After a latency of M clock cycles the columns of the eliminated matrix are output.

Note, that linear independency of LRPs is inherently checked in during the setup phase. If a LRP turns out to be dependent on some other LRP it is simply discarded, resulting is neglectable loss in correction performance. Hence finally $\hat{\mathbf{H}}$ is diagonalized at independent positions (at LRIPs).

This two phase architecture has proven to be an efficient solution for this application and outperforms standard Gaussian elimination architectures, like systolic arrays. For more information on the functionality and the architecture of the Gaussian elimination, please refer to [18].

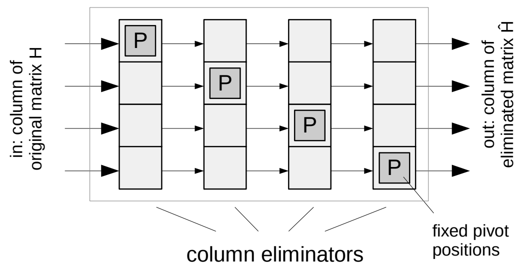
in: column of original matrix H
P
P
P
P
out: column of eliminated matrix Ĥ
column eliminators
fixed pivot positions

Fig. 8. Array for Gaussian elimination (here with M = 4)

Correction Unit

The main task is to determine erroneous data positions. In case of an error, the erroneous bits are read from the "I/O bits memory", flipped and afterwards written back. The unit further determines if OSD(0) or OSD(1) has to be performed. To determine the decoding strategy and the erroneous bit positions syndrome weights have to be calculated.

The weight calculation of binary vectors is accomplished by an adder tree of P stages. Several pipeline stages have been added between the adder stages to reduce the critical path.

Hard Decision Decoder

Here we use a HDD IP core for Reed-Solomon codes from Xilinx [1]. It supports the considered codes and provides the necessary throughput for our architecture.

Quantization Issues and Parameters

Since soft information (LLRs) is only processed in the Sorting Unit, quantization of the LLR values affects only this small part of the decoder. Yet the number of quantization bits is to be chosen as low as possible to reduce the complexity in the Sorting Unit. By simulations we have determined that a LLR quantization of 7 bits for the RS(255,239) and 5 bits for the RS(63,55) code does not noticeable impact the FER performance.

Furthermore the threshold Θ for the syndrome weight is of major importance. Simulations show best results when choosing $\Theta = 35$ for the RS(255,239) and $\Theta = 10$ for the RS(63,55). Parallelism in the Sorting Unit is chosen to be P, which equals the number of bits per GF symbol.

Pipelining and Latency Issues

We present two different implementations of our RS decoder:

- Architecture 1: variable latency with pipelining
- Architecture 2: fixed latency without pipelining.

In the proposed decoding architectures, performing OSD(0) and OSD(1) has a latency of 795 and 2838 clock cycles, respectively (RS(255,239)). Therefore OSD(1) is much more costly than OSD(0). In conjunction with the thresholding of the syndrome weight (see Section III-C), decoding throughput can be increased tremendously, if OSD(1) is only performed, if it is actually needed. This leads to a huge improvement of throughput, especially for high SNR values. This approach is used for Architecture 1. Moreover, we propose to use a two stage pipelining for Architecture 1:

- Stage 1: LLR input, sorting, syndrome calc., HDD
- Stage 2: Gaussian elimination and error correction

Architecture 2 has a fixed latency, that equals the worst case, i.e. OSD(1). For this architecture no pipelining is considered.

V. Implementation Results

In this section, we present the implementation results for the RS(255,239) and the RS(63,55) codes based on our new architecture. FPGA implementations have been done on a Virtex 7 (xc7vx690t-3) device using Xilinx ISE 14.4. All results have been obtained after place & route.

Implementation results for the RS(255,239) and for the RS(63,55) can be found in Table I. The results are obtained for Architecture 1 and 2. The results for Architecture 2 are given in brackets. For throughput calculations for Architecture 1 we consider FER=10^{-4}. The communication performance of the proposed decoders is shown in Fig. 9. For the RS(255,239) a gain of 0.75 dB and for the RS(63,55) a gain of 1.4 dB is achieved.

It should be mentioned, that the major part of resource utilization is contributed by the Gaussian elimination. Gaussian

TABLE I. Implementation results for the RS(255,239) and the RS(63,55) for a Virtex 7 device (number in brackets are for Architecture 2)

	RS(255,239)	RS(63,55)
LUTs	15.9k	3100
FFs	41.7k	7480
BRAMs (36K/18K)	7/8 (7/6)	1/5 (1/3)
fmax	280 MHz	300 MHz
throughput	1190 (200) MBit/s	690 (170) MBit/s
gain	0.75 dB	1.4 dB

elimination utilizes approximately 10.5k out of 15.9k LUTs and 35.7k out of 41.7k FFs.

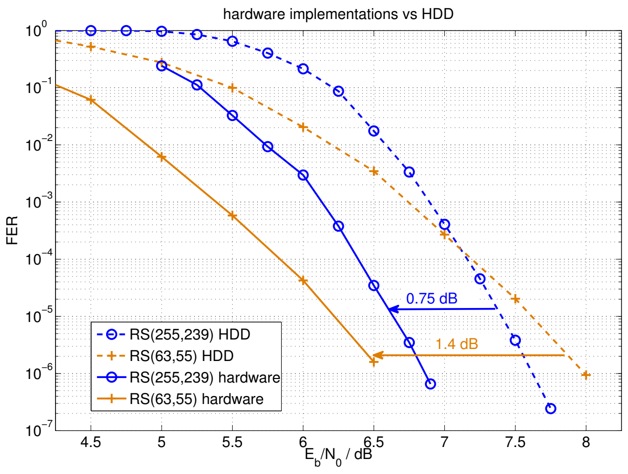

Fig. 9. FER for the hardware implementations

A comparison with other state-of-the-art FPGA soft decoders for RS codes is shown in Table II. Since the other decoders rely on older FPGAs, we implemented our design also on Virtex 5, which provides a more fair comparison between the different architectures.

TABLE II. COMPARISON WITH OTHER SOFT DECODER IMPLEMENTATIONS FOR RS(255,239) ON FPGA

implementation (algorithm)	FPGA	LUTs	FFs	throughput (MBit/s)	gain over HDD (FER=10^{-4})
[15] (ABP)	Stratix II	43.7k	n/a	32	0.75 dB
[12] (Chase)	Virtex 5	117k	143k	50	0.7 dB
new proposed (information set)	**Virtex 5**	**13.7k**	**41.8k**	**805**	**0.75 dB**

In terms of FER gain, our architecture is comparable to the state-of-the-art implementations (gain of 0.7-0.75 dB). However our decoder achieves this FER gain with significantly higher throughput and considerably less resource utilization.

It shows, that the previously disregarded information set decoding is a viable way to implement soft decoders for RS codes efficiently.

VI. Conclusion

To our best knowledge we presented the first soft decoder for long RS codes based on information set decoding. We implemented a reduced complexity version of ordered-statistics decoding, that simplifies the implementation and improves FER. The decoder has been implemented on a Xilinx Virtex 7 FPGA. The decoder's FER performance is state-of-the-art. This is achieved with more than 15x higher throughput and less resource consumption than state-of-the-art solutions. It shows, that information set decoding is an efficient approach for hardware implementations of soft decision RS decoding.

Acknowledgement

We gratefully acknowledge partially financial support by the DFG (project-ID: KI 1754/1-1) as well as by the Center of Mathematical and Computational Modelling of the University of Kaiserslautern. We thank Frank Kienle for his valuable comments and suggestions.

References

[1] Xilinx LogiCORE IP Reed-Solomon Decoder. http://www.xilinx.com/products/intellectual-property/DO-DI-RSD.htm, March 2013.

[2] IEEE 802.16. *Local and metropolitan area networks; Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems; Amendment 2:Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands*.

[3] A. Ahmed, R. Koetter, and N. R. Shanbhag. Performance analysis of the adaptive parity check matrix based soft-decision decoding algorithm. In *Proc. Conf Signals, Systems and Computers Record of the Thirty-Eighth Asilomar Conf*, volume 2, pages 1995–1999, 2004.

[4] Wei An. *Complete VLSI Implementation of Improved Low Complexity Chase Reed-Solomon Decoders*. PhD thesis, Massachusetts Institute of Technology, September 2010.

[5] J. Bellorado and A. Kavcic. A Low-Complexity Method for Chase-Type Decoding of Reed-Solomon Codes. In *Proc. IEEE Int Information Theory Symp*, pages 2037–2041, 2006.

[6] D. Chase. Class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, 18(1):170–182, 1972.

[7] B. Dorsch. A decoding algorithm for binary block codes and$J$-ary output channels (Corresp.). *IEEE Transactions on Information Theory*, 20(3):391–394, 1974.

[8] M. El-Khamy and R. J. McEliece. Iterative algebraic soft-decision list decoding of Reed-Solomon codes. *IEEE Journal on Selected Areas in Communications*, 24(3):481–490, 2006.

[9] M. P. C. Fossorier and Shu Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, 41(5):1379–1396, 1995.

[10] F. GarcÃa-Herrero, J. Valls, and P.K. Meher. High-Speed RS(255, 239) Decoder Based on LCC Decoding. *Circuits, Systems, and Signal Processing*, 30:1643–1669, 2011.

[11] K. Gracie and M.-H. Hamon. Turbo and Turbo-Like Codes: Principles and Applications in Telecommunications. *Proceedings of the IEEE*, 95(6):1228–1254, June 2007.

[12] R. Heloir, C. Leroux, S. Hemati, M. Arzel, and W.J. Gross. Stochastic chase decoder for reed-solomon codes. In *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, pages 5 –8, june 2012.

[13] Chih-Hsiang Hsu, Yi-Min Lin, Hsie-Chia Chang, and Chen-Yi Lee. A 2.56 Gb/s soft RS (255,239) decoder chip for optical communication systems. In *Proc. ESSCIRC (ESSCIRC)*, pages 79–82, 2011.

[14] Jing Jiang. *Advanced Channel Coding Techniques using Bit-Level Soft Information*. Dissertation, Texas A&M University, August 2007.

[15] M. Kan, S. Okada, T. Maehara, K. Oguchi, T. Yokokawa, and T. Miyauchi. Hardware implementation of soft-decision decoding for Reed-Solomon code. In *Proc. 5th Int Turbo Codes and Related Topics Symp*, pages 73–77, 2008.

[16] R. Koetter and A. Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 49(11):2809–2825, 2003.

[17] S. Lin and D.J. Costello Jr. *Error Control Coding 2nd.* Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 2004.

[18] S. Scholl, C. Stumm, and N. Wehn. Hardware Implementations of Gaussian Elimination over GF(2) for Channel Decoding Algorithms. In *Proc. IEEE AFRICON 2013*.

[19] Jin Wenyi and M. Fossorier. Towards Maximum Likelihood Soft Decision Decoding of the (255,239) Reed Solomon Code. *IEEE Transactions on Magnetics*, 44(3):423–428, 2008.