Chapter 2
Challenges and Limitations for Very High Throughput Decoder Architectures for Soft-Decoding

Norbert Wehn, Stefan Scholl, Philipp Schläfer, Timo Lehnigk-Emden, and Matthias Alles

2.1 Motivation

In modern communications systems the required data rates are continuously increasing. Especially consumer electronic applications like video on demand, IP-TV, or video chat require large amounts of bandwidth. Already today's applications require throughputs in the order of Gigabits per second and very short latency. Current mobile communications systems achieve 1 Gbit/s (LTE [1]) and wired transmission enables even higher data rates of 10 Gbit/s (e.g., Thunderbolt [2], Infiniband [3]) up to 100 Gbit/s. For the future it is clearly expected that even higher data rates become necessary. Early results show throughputs in the order of 100 Tbit/s [4] for optical fiber transmissions.

Satisfying these high date rates poses big challenges for channel coding systems. Software solutions usually achieve only very small data rates, far away from the required speed of most applications. Therefore dedicated hardware implementations on ASIC and FPGA are mandatory to meet the requirements for high speed signal processing. To achieve speeds of Gigabits per second, these architectures need large degrees of parallelism.

Parallelism and speed can easily be increased by running several single decoders in parallel. This is however mostly an inefficient solution, because area and power increase linearly with parallelism. Moreover it implies a large latency.

N. Wehn (✉) • S. Scholl • P. Schläfer
Microelectronic System Design Research Group, University of Kaiserslautern, Kaiserslautern, Germany
e-mail: wehn@eit.uni-kl.de; scholl@eit.uni-kl.de; schlaefer@eit.uni-kl.de

T. Lehnigk-Emden • M. Alles
Creonic GmbH, Kaiserslautern, Germany
e-mail: info@creonic.com

C. Chavet, P. Coussy (eds.), *Advanced Hardware Design for Error Correcting Codes*, DOI 10.1007/978-3-319-10569-7_2

Thus it is more advantageous to investigate efficient architectures specialized to high throughput. This may also include modifications to the decoding algorithm itself.

An important metric for analyzing high throughput architectures is area efficiency. Area efficiency is defined as throughput per chip area ($[(\text{Gbit/s})/\text{mm}^2]$). The area efficiency can be increased significantly by new architectural approaches.

We present high throughput decoders for different application relevant coding schemes, such as Reed–Solomon, LDPC and Turbo codes and point out their benefits compared to state-of-the-art architectures.

2.2 Architectures for Soft Decision Reed–Solomon Decoders

2.2.1 Introduction

Reed–Solomon (RS) codes are utilized in many applications and communication standards, either as a stand-alone code or in concatenation with convolutional codes, e.g., DVB. They are traditionally decoded using hard decision decoding (HDD), using, e.g., the well-known Berlekamp–Massey algorithm. However, using also the probabilistic information—so-called soft information—on the received bits can lead to large improvement of frame error rate (FER) in comparison to HDD.

Numerous algorithms have been proposed for soft decision decoding of RS codes. They are using different approaches to achieve a gain in FER over HDD with different complexities. A selection of interesting algorithms can be found in [5–11]. This chapter will focus on the RS(255,239) and RS(63,55) codes, because they are widely used in many applications.

Up to now, only few hardware implementations for ASIC and FPGA have been proposed for soft decoding of RS codes, especially the RS(255,239). One trend becoming apparent are implementations based on Chase decoding [7] and the closely related low-complexity chase (LCC) algorithm [8]. Hardware implementations based on LCC exhibit low hardware complexity [12–14], but this low complexity comes at the expense of a poor FER gain. Implementations based on LCC provide only little FER gain over HDD of about 0.3–0.4 dB.

The design of hardware architectures for a larger gain in FER is more challenging. Architectures and implementations based on adaptive belief propagation and stochastic Chase decoding exhibit a larger FER gain (0.75 dB), but having a low throughput [15, 16].

In this chapter a third approach for soft decoding is described that enables a large gain in FER *and* high throughput. It is based on a variant of the information set decoding algorithm, for which an efficient architecture is presented. This architecture shows a uncompromised gain in FER of 0.75 dB and a high throughput that exceeds 1 Gbit/s on a Xilinx Virtex 7 FPGA [17].

2.2.2 *Information Set Decoding*

This section introduces the algorithm, which is the basis of the high throughput hardware architecture. First, the used variant of information set decoding called ordered statistics decoding (OSD) algorithm [10] is reviewed. Then, a reduced complexity version of OSD using the syndrome weight [18] is presented.

2.2.2.1 Original OSD

OSD has been proposed in [10] and belongs to the class of information set decoders [19].

Basically information set decoding works as follows: First, divide the N received bits $\bar{\mathbf{y}}$ into two groups according to their reliability. The bit reliability is determined by the absolute value of the corresponding log likelihood ratio (LLR). The first group contains the set of K reliable bits, called the *information set*. The second group contains the $M = N - K$ unreliable bits, also referred to as low reliable bit positions (LRPs).

Before actual decoding starts the M LRPs are erased. Then the information set is used to reconstruct the M erased bits using the M parity checks in the parity check matrix $\mathbf{H}$. To do so, $\mathbf{H}$ has to be put in a diagonalized form $\hat{\mathbf{H}}$ via Gaussian elimination. If the K bits of the information set are correct, all errors in the M LRPs can be corrected. This is referred to as order-0 reprocessing in OSD or OSD(0).

To perform successful correction in the case of one error in the information set, the reconstruction process is repeated several times, each time with exactly one bit of the information set flipped. This results in a list of $K + 1$ possible codewords from which the best codeword is selected by evaluating the Euclidean distance to the received LLRs. This improved decoding is called order-1 reprocessing or OSD(1).

A key part for understanding information set decoding is the reconstruction process. To successfully reconstruct the M erased bits, the rows of the parity check matrix are used, as mentioned before. It is required that the parity check equation in each row covers mostly one of the erased M bits. To fulfill this requirement, $\mathbf{H}$ is put into a diagonalized form by Gaussian elimination, such that each row covers not more than one erased bit. Note that the Gaussian elimination does not change the channel code itself, because an RS code is a linear code.

2.2.2.2 Reduced Complexity Algorithm for Hardware

The computational bottleneck of the original algorithm are the reconstructions of the M erased bits. For example, in case of decoding an RS(255,239) with OSD(1) this operation is required 2,041 times. To overcome this problem a reduced complexity algorithm is utilized which makes use of the syndrome $\hat{\mathbf{s}}$ and its weight [18] that enables fast and efficient reconstruction.

The reduced complexity algorithm starts (as the original OSD) with determining the information set according to the bit reliabilities and diagonalization of $\mathbf{H}$ by Gaussian elimination.

Original OSD then evaluates the parity check equations given by the rows of $\hat{\mathbf{H}}$ whereas the considered low-complexity algorithm merely uses the syndrome to correct the corrupted bits.

Moreover, 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 is assumed that only errors in the M bits are present, i.e., OSD(0) processing is sufficient.
- The syndrome weight is large: In this case, it is assumed that also errors in the information set are also 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 M LRPs that have led to the 1s in the syndrome vector. Conducting OSD(1) (large syndrome weight) to correct one error inside the information set is done by first flipping the bit position

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

where $\hat{\mathbf{h}}_{\mathbf{i}}$ denotes the ith column of $\hat{\mathbf{H}}$. After flipping the error inside the information set at position j, the syndrome is calculated again and the remaining errors outside the information set (i.e., among the LRPs) are corrected by performing OSD(0).

Note that this algorithm inherently determines the best codewords among the possible candidates only by looking at the syndrome weight. It is sufficient to select the candidate with smallest syndrome weight. In case of original OSD the Euclidean distance between candidate and received LLRs had to be evaluated many times.

For more detailed information on the syndrome weight OSD please refer to [18] or [17].

2.2.2.3 HDD Aided Decoding

One disadvantage of OSD over other soft decision decoding algorithms is the tendency for a weak FER performance if SNR increases. To improve FER OSD is extended with a conventional HDD, whose result is output if OSD fails.

A failure in OSD can be easily detected again by looking at the syndrome weight. If after OSD(1) reprocessing the updated syndrome still has a large weight, OSD can be considered as unsuccessful.

1. ***Sorting:***
   determine the information set of K reliable bits
2. ***Gaussian Elimination:***
   diagonalize $\mathbf{H}$ *at the* $N-K$ *low reliable positions to obtain* $\hat{\mathbf{H}}$
3. ***calculate the syndrome*** $\hat{\mathbf{s}} = \hat{\mathbf{H}}\bar{\mathbf{y}}^{\mathrm{T}}$
   and its binary weight $wgt\,(\hat{\mathbf{s}})$
4. **If** $wgt\,(\hat{\mathbf{s}}) > \Theta$: /* errors in the information set */
   flip the received bit at position
   $j = \underset{i=0,\ldots,N-1}{argmin}\ wgt\left(\hat{\mathbf{s}} \oplus \hat{\mathbf{h}}_{\mathbf{i}}\right)$
   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 errors outside the information set remaining (LRP errors) */
   For all $\hat{s}_i = 1$, *flip bit position l, for which* $\hat{h}_{il} = 1$
   output OSD result, terminate
   else
   perform HDD on $\bar{\mathbf{y}}$ *and output HDD result,* terminate

Fig. 2.1 Reduced complexity OSD(1) based on the syndrome weight that is implemented

2.2.2.4 Implemented OSD Version

The reduced complexity OSD algorithm for hardware implementation is summarized in Fig. 2.1. It features sorting to determine the information set, followed by Gaussian elimination to diagonalize the matrix. Then the syndrome weight for the diagonalized matrix is determined and a decoding strategy (OSD(1)+OSD(0) or OSD(0) only) is selected. HDD is performed only if OSD fails.

2.2.3 Hardware Architecture

In this section a hardware architecture based on the previously introduced algorithm (Fig. 2.1) is presented.

2.2.3.1 Architecture Overview

Figure 2.2 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 received LLRs are sorted using a parallelized sorter. The M bit positions outside the information set are stored in the "LRP memory." Simultaneously, the syndrome is calculated based on the original (non-eliminated) parity check matrix $\mathbf{H}$ for reasons that will be explained below. Also HDD is carried out, whose result is stored in the "HDD memory."

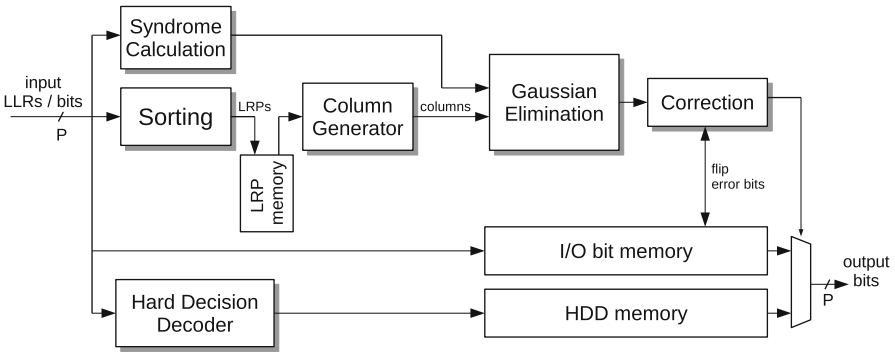

Fig. 2.2 Decoder architecture overview

After that, the column generator generated the columns of $\mathbf{H}$ corresponding to the M LRPs for Gaussian elimination. These LRP columns are fed into the Gaussian Elimination Unit to dynamically set up the unit (see below) for further processing. After having set up the Gaussian Elimination Unit, the syndrome $\mathbf{s}$ is put into the elimination unit to obtain $\hat{\mathbf{s}}$ (its version based on the diagonalized matrix).

After determining the initial syndrome $\hat{\mathbf{s}}$, 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 successively every column of $\mathbf{H}$, which are transformed into the columns of $\hat{\mathbf{H}}$ by the Gaussian Elimination Unit. The Correction Unit determines the erroneous bit positions based on these columns and the syndrome and flips these bits in the "I/O bits memory." Finally, the Correction Unit decides if the best OSD codeword or the HDD codeword is output.

2.2.3.2 Sorting Unit

The first step of decoding is finding the LRPs by taking the absolute value of the received LLRs and partially sorting them. This is accomplished by the Sorting Unit depicted in Fig. 2.3. Sorting is performed by using a shift register based insertion sort.

In order to reduce the latency for sorting, the shift register is partitioned in P parts. Each part is calculated in parallel. However, the results provided by the parallelized Sorting Unit are not exactly the LRPs, but rather an approximation of the LRPs. This introduces a small loss in FER, but simulations show that this loss is less than 0.1 dB for the RS(255,239) using an input parallelism of $P = 8$.

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

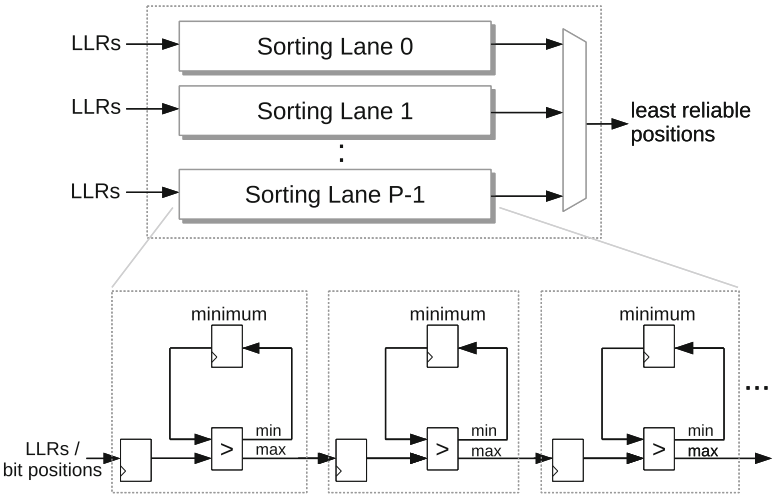

Fig. 2.3 Parallelized sorting unit

2.2.3.3 Syndrome Calculation Unit

The subsequent stages require the calculation of the syndrome using the diagonalized matrix $\hat{\mathbf{s}} = \hat{\mathbf{H}}\bar{\mathbf{y}}^{\mathbf{T}}$. However, it is advantageous to first calculate the syndrome using the original parity check matrix $\mathbf{H}$ and afterwards pass it through the Gaussian Elimination Unit to obtain $\hat{\mathbf{H}}$.

This allows to use efficient syndrome calculation using Galois field arithmetic, as it is well known in literature [20]. The syndrome unit is a parallelized implementation that can handle one received symbol (P bits) per clock cycle.

2.2.3.4 Column Generator Unit

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

2.2.3.5 Gaussian Elimination Unit

Gaussian Elimination is required to diagonalize $\mathbf{H}$ and the syndrome $\mathbf{s}$. This is the most complex operation in the algorithm. Therefore sophisticated architectures are required to achieve high throughput.

An elegant architecture for Gaussian elimination has been proposed in [21]. 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 LRPs.

The array works in two phases:

1. The Setup Phase: The M columns, which are supposed to become unit vectors after elimination, (here: 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. The columns of the eliminated matrix are output after M clock cycles.

Note that linear independency of LRPs is required for full elimination. Possible dependencies are inherently checked during the setup phase. If an LRP column turns out to be dependent on some other LRP column, it is simply discarded, so that the matrix is not fully diagonalized. Since the number of dependent columns is usually very low the resulting loss in correction performance is negligible.

This two-phase architecture has proven to be an efficient solution for this application and outperforms standard Gaussian elimination architectures (e.g., systolic arrays) as can be seen in Table 2.1. For more information on the functionality and the architecture of the utilized Gaussian elimination please refer to [21] (Fig. 2.4).

2.2.3.6 Correction Unit

The unit first determines if OSD(0) or OSD(1) has to be performed. To determine the decoding strategy and the erroneous bit positions the syndrome weight has to

Table 2.1 Comparison of state-of-the-art implementations for Gaussian elimination of the binary 128×2040 matrix for the RS(255,239) code on a Xilinx Virtex 7 FPGA using Vivado 2012

Architecture	LUTs	FFs	f_{max} (MHz)	Throughput (matrices/s)
SMITH (estimated) [22]	780k	260k	–	–
Systolic array [23]	81.7k	98.6k	350	145k
Proposed [21]	**16.6k**	**32.9k**	**370**	**171k**

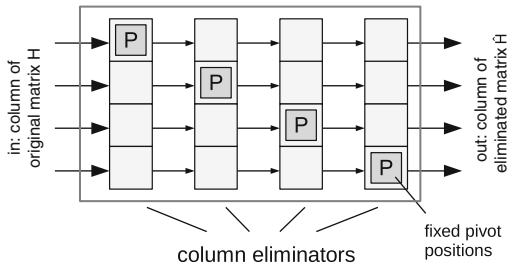

Fig. 2.4 Array for Gaussian elimination (here with $M = 4$)

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

However, the main task of this unit is to perform the actual error correction. To determine erroneous bit positions, the syndrome and its correlation to the columns of $\hat{\mathbf{H}}$ are evaluated according to Steps 4 and 5 of the decoding algorithm (Fig. 2.1). In case of an error, the corrupted bits are flipped in the "I/O bits memory."

2.2.3.7 Hard Decision Decoder

To improve the FER, an additional HDD is employed. For the FPGA implementation a HDD IP core for decoding RS codes from Xilinx [24] is used. It supports the considered codes and provides the necessary throughput for the decoder architecture.

2.2.3.8 Fixed Point Quantization Issues

Since soft information (LLRs) is only processed in the Sorting Unit, quantization of the LLR values affects only this small part of the decoder. By simulations it is determined that an LLR quantization of 7 bits for the RS(255,239) and 5 bits for the RS(63,55) code does not noticeably impact the FER performance.

2.2.3.9 Pipelining and Latency Issues

In the proposed decoding architecture, performing OSD(0) and OSD(1) has a latency of 795 and 2,838 clock cycles, respectively (RS(255,239)). This shows that OSD(1) is much more costly than OSD(0). In conjunction with the thresholding of the syndrome weight (see Sect. 2.2.2.4), decoding throughput can be increased largely, if OSD(1) is only performed, if it is actually needed. This leads to a large improvement of throughput, especially for SNR values of practical interest (typical throughput).

Moreover, a two-stage pipelining is used:

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

2.2.4 Implementation Results

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

Table 2.2 Implementation results for the new architecture for the RS(255,239) and the RS(63,55) on a Virtex 7 FPGA

	RS(255,239)	RS(63,55)
LUTs	15.9k	3100
FFs	41.7k	7480
BRAMs (36K/18K)	7/8	1/5
f_{max}	280 MHz	300 MHz
Worst case throughput	200 Mbit/s	170 Mbit/s
Typical throughput	1,190 Mbit/s	690 Mbit/s
Gain	0.75 dB	1.4 dB

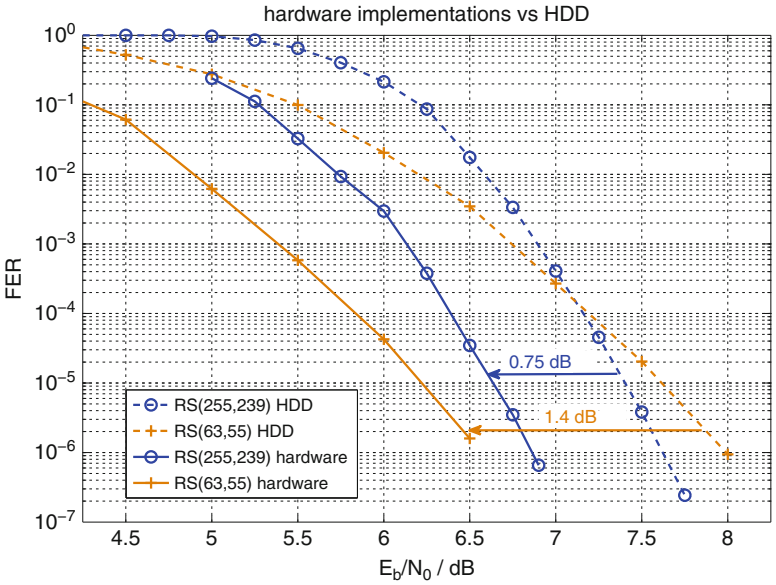

Fig. 2.5 FER for the hardware implementations

Implementation results for the RS(255,239) and for the RS(63,55) can be found in Table 2.2. For typical throughput calculations FER$= 10^{-4}$ is considered. The communication performance of the proposed decoder is shown in Fig. 2.5. For the RS(255,239) a gain of 0.75 dB and for the RS(63,55) a gain of 1.4 dB are achieved.

A comparison with other state-of-the-art FPGA soft decoders for RS codes is presented in Table 2.3. Since the other decoders rely on older FPGAs, results are given for Virtex 5, which provides a more fair comparison between the different implementations.

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

Table 2.3 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	4	0.75 dB
[16] (Chase)	Virtex 5	117k	143k	50	0.7 dB
New proposed (information set)	**Virtex 5**	**13.7k**	**41.8k**	**805**	**0.75 dB**

The implementation shows that information set decoding is a viable way to implement soft decoders for RS codes efficiently, also for large throughput requirements.

2.3 Architectures for Turbo Code Decoders

Turbo codes are widely used for error correction in mobile communications, e.g., in UMTS and LTE systems. Similar as in other areas, their throughput demand is increasing, currently reaching beyond 1 Gbit/s for LTE.

Turbo code decoding is inherently serial on component and on the decoder level. A turbo code decoder consists of two component decoders which iteratively exchange data on block level. A complete data block of length B is fully processed by one component decoder before the processed block can be sent to the other component decoder. This process continues for a certain amount of iterations, typically between 5 and 8. However, the data is not directly sent to the other component decoders. Instead the data is interleaved before exchanging. This interleaving is pseudo-random with limited locality and can result in access conflicts if several messages are produced in one clock cycle by a component decoder. Resolving these access conflicts imposes constraints on an interleaver to permit a parallelized data exchange.

The component decoders are soft-in, soft-out decoders. State-of-the-art turbo code decoders are using the Bahl, Cocke, Jelinek, and Raviv algorithm, often also called Maximum-a-posteriori algorithm (MAP) [25]. This algorithm sequentially processes a block from the beginning to the end, named forward recursion, and vice versa, called backward recursion [26]. Due to the recursive nature of the forward and backward calculations, the standard MAP algorithm cannot straightforward be parallelized. Thus, we are facing two challenges for high throughput turbo code decoders:

- parallelizing the MAP algorithm and
- parallelizing the data exchange

An overview of different levels of parallelisms for turbo code decoders can be found in [27]. Parallelizing the MAP algorithm can be achieved by splitting the

complete data block into P smaller sub-blocks. So, the sub-block size is B/P. Since this splitting breaks up the recursion for forward and backward calculation, it will result in a large degradation of the communications performance. To counterbalance this effect, a so-called acquisition can be performed at the sub-block borders for the forward and/or backwards recursion. This acquisition consists of some additional recursions steps and approximates the state probability at the borders of the sub-blocks. The accuracy of this approximation strongly depends on the number of additional recursion steps. The number of additional steps is named acquisition length L_{ACQ}. In this way each sub-block can be processed independently of the other sub-block on a dedicated MAP engine. That is, in this case a component decoder consists of P parallel working MAP engines where each MAP engine serially processes a sub-block. So, instead of B clock cycles needed to process one data block of size B (here we assume that one recursion step is performed in one clock cycle), we need only B/P clock cycles. State-of-the-art MAP decoders use the same splitting technique to reduce the storage amount when processing this sub-block inside a MAP engine. This technique is called sliding windowing [28]. We use the term window instead of sub-block to avoid a confusion with the splitting on block level. The window length is denoted L_{WL}.

Since one component decoder is always idle while waiting for the result of the other decoder, we can map both component decoders on the same hardware unit without degrading the throughput. If we assume that there is no throughput penalty due to the data exchange between the two component decoders, we can calculate the throughput TP of a state-of-the-art turbo code decoder with Eq. (2.1).

$$TP = \frac{B}{(B/P + L_{MAP}) \cdot n_{half\_iter}} \cdot log_2(r) \cdot f[Mbit/s], \tag{2.1}$$

with f being the frequency and n_{half_iter} the number of invocations of a component decoder. Please note that a component decoder is invoked twotimes per decoding iteration. r is the used radix. Radix-2 means that the MAP engine processes one recursion step per clock cycle. In radix-4, two recursion steps are merged into a single one which can be processed in one clock cycle. This merging increases the area and slightly decreases the frequency, but reduces the number of clock cycles for processing a sub-block by a factor 2. L_{MAP} is the overhead due to the parallel processing of a data block and is composed of three components:

- $L_{pipeline}$: a MAP engine is pipelined to increase the frequency. This implies a certain latency which is typically 10–20 clock cycles.
- L_{ACQ} is the aforementioned acquisition length.
- L_{WL} is the window length of the sliding windowing.

From this equation we see a linear increase in the throughput with increasing parallel processing as long as L_{MAP} can be neglected. We see also that high throughput decoders require a large P. However a large P increases the impact of L_{MAP} on TP. Moreover current communication standards like LTE specify high throughputs only for high code rates ($R = 0.95$). It can be shown that large code

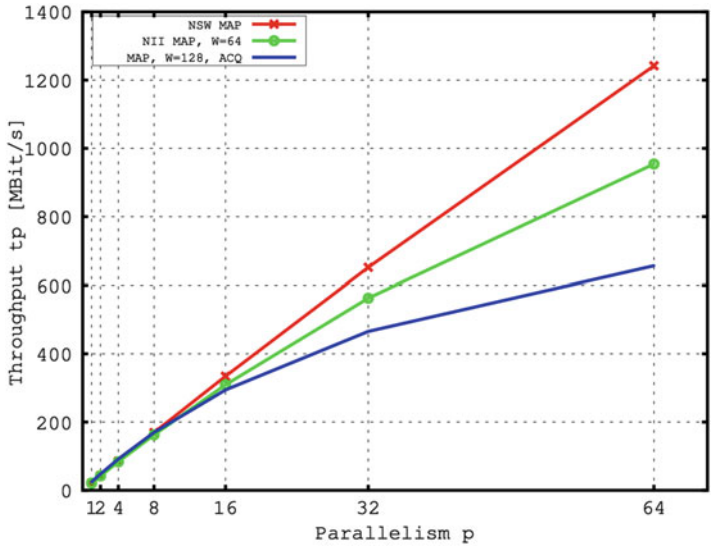

Fig. 2.6 Throughput for different MAP architectures: Non-sliding window MAP with NII and 6 iterations, sliding window MAP with NII and $l_{WS} = 64$ and 6 iterations, sliding window MAP with $l_{ACQ} = l_{WS} = 128$ and 6 iterations

rates demand large L_{ACQ} for a good communications performance which further exacerbates the dominance of L_{MAP}, e.g. in LTE $L_{ACQ} > 64$ is mandatory. In other words, the throughput starts to saturate with large P and high code rate demands, as shown in Fig. 2.6. This fact poses a further challenge for turbo decoder architectures when high throughput is required. However we can use two techniques to relax this problem:

- We can reduce the large acquisition length by exploiting the state probabilities at the borders of the sub-block from the previous decoding iteration. This technique is called next iteration initialization and largely helps to reduce L_{ACQ} [29]. Sometimes it is even possible to perform no acquisition at all by using only the information of the previous iteration.
- We quit the sliding window technique inside a MAP engine. This normally largely increases the memory and energy consumption inside the engine. However this can be avoided by so-called re-computation [30]. Here, instead of storing every metric of a forward recursion step, we store only every nth metric and re-calculate the other $n-1$ metrics. That is, we trade-off storage versus additional computations. It can be proven that the optimum n is $\sqrt{B/2P}$. For example, this technique reduces the number of metrics to be stored for LTE from 6,144 for the largest block size to 768.

So far we assumed that interleaving implies no additional latency. Obviously a component decoder produces P data per clock cycle. These data have to be

Table 2.4 Previously published high throughput 3GPP TC decoders

	This work	[33]	[34]	[35]
Radix and Parallelism	4/32	2/64	2/8	4/8
Throughput (MBit/s)	2,300	1,200	150	390
@Clock f (MHz)	500	400	300	302
Parameters affecting communications performance				
Iterations	6	6	6.5	5.5
Acquisition	NII	NII	96 + NII	30
Window length	192	64	32	30
Input quantization	6	6	6	5
Technology (nm)	65	65	65	130
Voltage	1.1 V	0.9 V	1.1 V	1.2 V
Area (mm^2)	7.7	8.3	2.1	3.57

concurrently interleaved. Since each MAP engine has its own memory, random interleavers can result in access conflicts if we have single- or dual-port memories. That is, several data have to be written simultaneously into the same memory. Such conflicts have to be resolved by serialization. Another possibility is to design the interleaver a-priori in a way such that these conflicts are avoided. Current communication standards like LTE are based on such interleavers and show no access conflicts for up to 64 simultaneous produced data. On the other side HSDPA has no conflict free interleavers due to its downward compatibility with UMTS. Parallel MAP processing was not yet an issue at the time when UMTS was defined. Sophisticated techniques exist for run-time conflicts resolution, but this discussion is not in the scope of this chapter [31]. The influence of the explained techniques on the achievable throughput is shown in Fig. 2.6.

In [32] an LTE compatible turbo code decoder was presented which used all the aforementioned techniques. It achieves a throughput of 2.15 Gbit/s on a 65 nm CMOS bulk technology under worst case PVT parameters. It uses 32 MAP engines with radix-4, next iteration initialization and no sliding window but re-computation. The detailed results and comparison with state-of-the-art decoders are shown in the Table 2.4.

2.4 High Throughput Architectures for Low Density Parity Check Decoders

As discussed in Sect. 2.3, turbo code based systems cannot provide data rates in the order of several hundred Gigabits per second. For these applications LDPC codes are the best choice. The decoding algorithms for LDPC codes have an inherent parallelism which can be exploited by highly parallel architectures.

LDPC codes have been introduced by Gallager in 1962 [36] but the high decoding complexity made the application at that time impossible. When LDPC codes have been rediscovered in the late 1990s, the throughput demands have been moderate. Serial decoder architectures have been sufficient to fulfill the requirements. As demands on the throughput rose, partially parallel architectures became necessary. Today LDPC codes are used in a wide range of applications like 10 Gbit Ethernet (10 GBASE-T, IEEE802.3an) [37], broadband wireless communication (UWB, WiGig) [38, 39], and storage in hard disc drives [40]. State-of-the-art LDPC decoders can already process data rates in the range of 10–50 Gbit/s. This is sufficient to satisfy the requirements of all mentioned standards. However, as future standards emerge, current architectures will not be able to facilitate the demanded throughputs of 100 Gbit/s and more. For higher throughputs even LDPC decoders reach their limit. This results in a gap in decoder performance which has to be closed by new approaches. Therefore a new architecture is presented which can overcome these limitations and the key aspects for next generation LDPC decoders are discussed. It is shown that new architectures significantly reduce routing congestion which poses a big problem for high speed LDPC decoders. The presented 65 nm ASIC implementation results underline the achievable gain in throughput and area efficiency in comparison to state-of-the-art architectures.

A LDPC decoder system with state-of-the-art communications performance and a throughput far beyond 100 Gbit/s is presented which is a candidate for future communications systems.

2.4.1 LDPC Decoding

LDPC codes [36] are linear block codes defined by a sparse parity check matrix $\mathbf{H}$ of dimension $M \times N$, see Fig. 2.7a. A valid codeword $\mathbf{x}$ has to satisfy $\mathbf{Hx^T} = \mathbf{0}$ in modulo-2 arithmetic. A descriptive graphical representation of the whole code is given by a Tanner graph. Each row of the parity check matrix is represented by a check node (CN) and corresponds to one of the M parity checks. Respectively each column corresponds to a variable node (VN) representing one of the N code bits. The Tanner graph shown in Fig. 2.7b is the alternative representation for the parity check matrix of Fig. 2.7a. Edges in the Tanner graph reflect the 1's in the $\mathbf{H}$ matrix. There is an edge between VN n and CN m if and only if $\mathbf{H}_{mn} = 1$. LDPC codes can be decoded by the use of different algorithms. Belief Propagation (BP) is a group

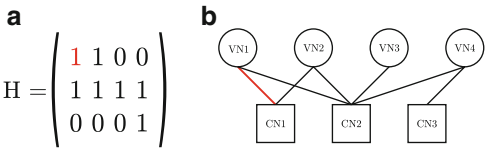

Fig. 2.7 **H** Matrix and Tanner graph hardware mapping

of algorithms which is used in most state-of-the-art decoders. Which type of BP fits best has to be chosen dependent on the required communications performance. For example, the λ-min algorithm [41] performs better than the min-sum algorithm [42] but has a significantly higher implementation complexity. All algorithms have in common that probabilistic messages are iteratively exchanged between variable and check nodes until either a valid codeword is found or a maximum number of iterations is exceeded.

2.4.2 LDPC Decoder Design Space

The LDPC decoder design space comprises a multitude of parameters which have to be tuned to the specific requirements. Each standard has different needs in means of error correction performance, number of supported code rates, codeword lengths, and throughput. There are numerous design decisions which have to be made for the hardware to satisfy these requirements. Due to their inherent parallelism, LDPC decoders are of special interest for high throughput applications. Therefore the focus is on the design decisions concerning the decoder parallelism. They have the strongest impact on the system's throughput. An in-depth investigation of the design space for slower state-of-the-art partially parallel decoders is presented in [43]. This part of the design space is not highlighted as it is orthogonal to the presented schemes. For example, different check node algorithms can be combined with all levels or parallelism presented here.

There are multiple dimensions in which the degree of parallelism can be chosen, see Fig. 2.8. The lowest level of parallelism is on the message level. Each message

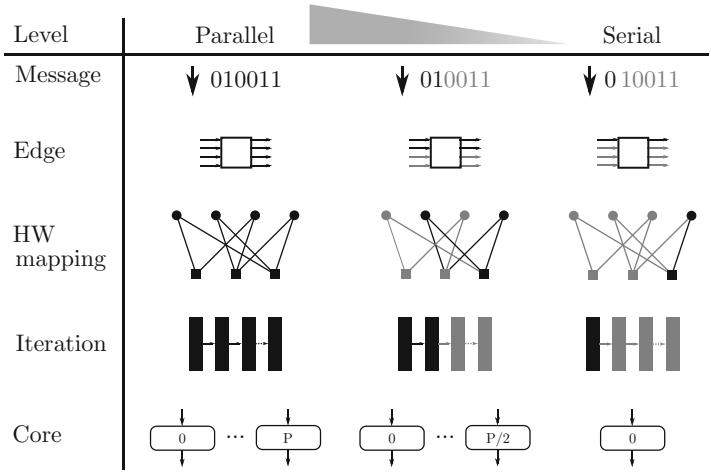

Fig. 2.8 Levels of parallelism in the high throughput LDPC decoder design space

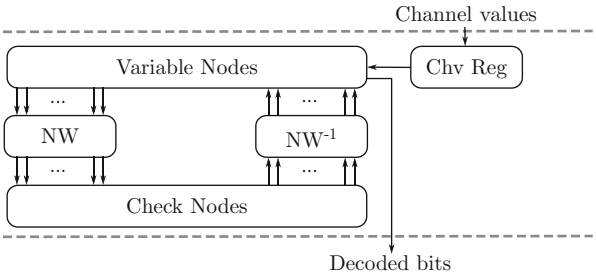

Fig. 2.9 Fully parallel hardware mapping of an LDPC decoder. All variable and check nodes are instantiated and two networks are required to exchange the messages between them. Massive routing congestion is observed for this architecture

can be transmitted in tuples of bits or fully parallel. Today fully parallel message transfer can be found in the vast majority of architectures as it has been shown to be more efficient. The second degree of parallelism is represented by the number of parallel edges. The node's in- and outgoing edges can be processed one after another, partially parallel or fully parallel. However the choice of the node's edge parallelism is directly linked to the so-called hardware mapping. The hardware mapping describes how many check and variable nodes are instantiated. When talking of a fully parallel decoder, an architecture instantiating all processing nodes is meant. In contrast partially parallel decoders have a lower number of physical nodes than the Tanner graph. They process the parity check matrix in a time multiplex fashion. This allows for easy adaption of the architecture to new standards but limits the achievable throughput.

For applications like 10 GBASE-T Ethernet only fully parallel architectures can achieve the required throughput. Figure 2.9 depicts the high-level structure of such a decoder. However in general it is not advisable to build this architecture as it has a serious drawback which is directly related with the two networks between VNs and CNs. Dependent on the code length and quantization, each of them comprises between several thousands and hundred thousands of wires which have to be routed according to the parity check matrix. To achieve a good communications performance, parity check matrices have long cycles and thus no locality, resulting in massive routing congestion. It has been shown in earlier publications [44,45] that the area utilization is heavily impaired by this fact and only 50 % of the chip is used by logic.

Fully parallel decoders can still satisfy today's requirements in means of throughput. They represent the highest level of parallelism used in state-of-the-art decoder designs. However, for future standards the throughput demands will further increase and cannot be achieved using the presented dimensions of parallelism.

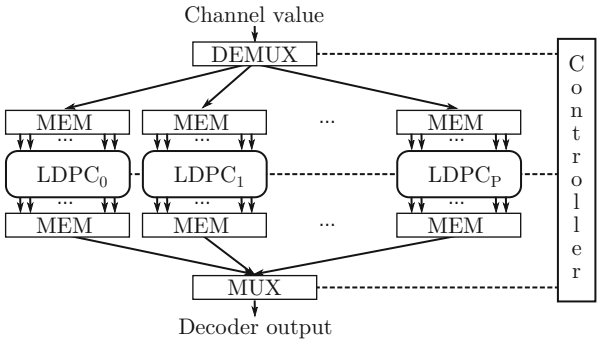

Fig. 2.10 Core duplication architecture

2.4.3 Exploring a New Dimension in the High Throughput LDPC Decoder Design Space

Considering the degrees of parallelism which are used for state-of-the-art decoders no further increase in throughput can be acquired. In the following section two more degrees in parallelism are discussed which can be explored to overcome the limitations in LDPC decoder throughput, see Fig. 2.8. Moreover it is shown that the area efficiency of decoders can even be increased by the proposed techniques.

2.4.3.1 Core Duplication

One solution to achieve the throughputs required by future standards is to instantiate several LDPC decoder cores in parallel, see Fig. 2.10. There are two possible starting points for the core duplications. Partially parallel architectures which allow for flexibility but suffer from high latencies and low throughput. The second option is to instantiate several fully parallel decoder cores allowing for reduced latency and high throughput. However due to routing congestion they cannot achieve a satisfying area efficiency and flexibility. To connect multiple instances of a decoder, a distribution network and memories are required. Moreover a control unit to keep the blocks in order must be instantiated in the system.

Summarized straightforward decoder duplication can increase the system throughput. However the latency issues caused by partially parallel architectures cannot be solved. Potential enhancements due to the increased parallelism are not explored and the system's efficiency is slightly decreased due to the introduced overhead.

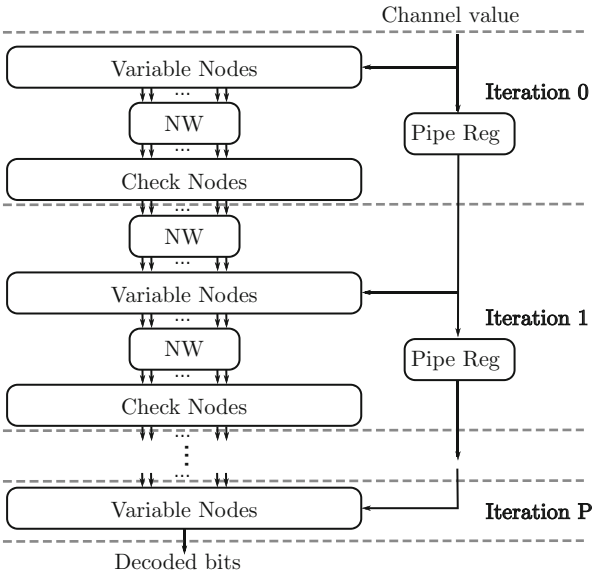

Fig. 2.11 In an unrolled LDPC decoder architecture each decoding iteration is instantiated as a dedicated hardware. A feedback from the end of the iteration back to the beginning is no more required with this approach. One of the two networks between variable and check nodes is removed and makes the routing feasible. Due to the unidirectional data flow pipelining can be applied without penalty in throughput. Synthesis results show an increased area efficiency compared to a fully parallel decoder

2.4.3.2 Unrolling Iterations

A new architecture is proposed in [46], shown in Fig. 2.11 to overcome the highlighted drawbacks. The iterative decoding loop is unrolled and an unrolled, fully parallel, pipelined LDPC decoder is instantiated. It has several advantages over core duplication and is a good choice for very high throughput architectures.

A drawback of this architecture is the need to specify the maximum number of iterations at design time. Once the decoder is instantiated there is no possibility to increase the performance by additional decoding iterations. The number of pipeline stages determines the latency, but the throughput is fixed by the cycle duration. The decoder can be considered as one big pipeline where received codewords are fed into and decoded words are returned at the end. Hence this architecture has a throughput of one codeword per clock cycle and can be pipelined as deep as required to achieve the target frequency. This allows for ultra-high throughput LDPC decoder cores.

Compared to the core duplication approach, no overhead in means of distribution networks and memory is introduced by the unrolling. Moreover there is an essential change in the resulting data flow. Where before data have iteratively been exchanged between VNs and CNs, now all data flow in one direction. Each iteration has a dedicated hardware unit and thus the decoder's overall area scales linear with the

number of decoding iterations. The result is an unidirectional wiring avoiding the overlap of opposed networks. This is a big benefit for the routing and makes the architecture more area efficient than state-of-the-art decoders which is shown in Sect. 2.4.4.

The control flow of the proposed architecture is reduced to a minimum. A valid flag is fed into the first decoding stage whenever a block is available. This flag is propagated along the decoding pipeline and enables the corresponding stages as soon as new data is available. At the same time this implies that all hardware blocks which are not used currently get clock gated.

Even though the number of decoding iterations is defined at design time, schemes like early termination can be applied to further reduce the energy consumption. Once a valid codeword is found all following decoding stages are clock gated for this block and the decoded data is bypassed in the channel value registers. By this approach besides some multiplexors no hardware overhead is introduced and the energy per decoded bit can be reduced significantly.

Even different code rates can be implemented in the unrolled architecture. Special codes like the one used in the IEEE 802.11ad standard allow for the operation of one CN instance as two CNs by cutting the node degree by two. Using these codes only minor modifications are required for the check nodes and the routing network to support all codes of the IEEE 802.11ad family. For a more detailed explanation of the CN splitting scheme see [47]. A similar scheme as proposed there can also be applied on the unrolled architecture. The control flow for the different code rates can easily be implemented by an additional flag propagated with the according input block. This allows to change the code rate for every block, e.g., in each clock cycle. Table 2.5 summarizes the benefits and drawbacks of the different approaches.

2.4.4 *Comparison of Unrolled LDPC Decoders to State-of-the-Art Architectures*

Two decoder architectures are presented which are compared to a state-of-the-art LDPC decoder from literature. The first decoder presented is a fully parallel architecture with iterative decoding. The second has also a fully parallel hardware

Table 2.5 Parallel LDPC decoder architectures

Architecture	Flexibility	Low latency	Area efficiency
Parallel inst., partially parallel architecture	+	−	0
Parallel inst., fully parallel architecture	−	0	−
Unrolled, fully parallel architecture	−	+	+

Table 2.6 State-of-the-art high throughput LDPC decoder comparison

Decoder	[48]	Iterative	Proposed unrolled
CMOS technology	65 nm	65 nm SVT	65 nm SVT
Frequency (MHz)	400	189	194
Standard	IEEE 802.3an	IEEE 802.11ad	IEEE 802.11ad
Block size	2,048	672	672
Iterations	8	6	6
Quantization (bit)	4	5	5
Post P&R area (mm^2)	5.1	1.4	7.0
Throughput (Gbit/s)	8.5	5.3	**130.6**
Area Eff. (Gbit/s/mm^2)	1.7	3.8	**18.7**

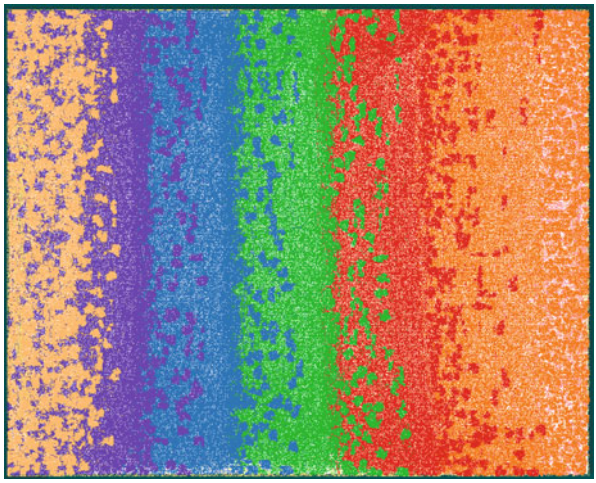

Fig. 2.12 Unrolled fully parallel LDPC decoder chip layout. Each iteration is represented by one of the vertical areas. Channel messages ripple from left to right through the decoder pipeline. All routing is very structured and pointing from one iteration to the next. A very high utilization of more than 70 % is achieved by the simplified routing

mapping but in addition the decoding iterations are completely unrolled. Both decoders support the same standard (IEEE 802.11ad) and use the same algorithm, quantization, etc.

The LDPC decoders are implemented on a 65 nm low power bulk CMOS library. The post place and route (P&R) results are summarized in Table 2.6. The physical layout of the unrolled LDPC decoder can be seen in Fig. 2.12. Comparing the synthesis results of the iterative and the unrolled decoder shows that the routing congestion is significantly reduced by the loop unrolling. The number of introduced buffers for the interconnect is significantly reduced in the unrolled decoder and leads to a high utilization of more than 70 %. A five times higher area efficiency of the unrolled decoder underlines this finding.

For the comparison, in addition to the two presented decoders, a partially parallel decoder from literature is listed. The number of iterations, quantization, and algorithm is reasonably similar to allow for a fair comparison. It can be observed that no state-of-the-art decoder architecture is capable to produce a competitive area efficiency to the unrolled architecture. The presented architecture has a throughput which is more than fifteen times higher than the one of state-of-the-art decoders. Moreover, if more throughput is required the unrolled architecture can easily be pipelined deeper to increase the core frequency. These results show the great potential of unrolled decoder architectures for future applications.

2.4.5 Future Work

The unrolled LDPC decoder architecture allows for several optimizations. In this section the most important of them are presented and current research topics are pointed out.

Unrolling the decoding loop generates a dedicated hardware instance for each iteration. While for systems working iteratively, a generic hardware fulfilling the needs of all iterations needs to be built, an unrolled architecture gives the designer the freedom to optimize the hardware for each iteration independent of the others. Thus it is possible to use specialized hardware instances for each iteration. For example, one can implement different algorithmic approximations. For example, for a decoder performing P iterations, a simplified decoding algorithm can be applied for iterations $1 \ldots i$ and an exact but more complex algorithm might be necessary only for iterations $i+1 \ldots P$. Like this a targeted communications performance can tightly be met while minimizing the required hardware resources. Even more than the area efficiency, the energy efficiency can be increased by this approach. Most blocks are decoded in the simplified first iterations of the decoding process and the higher complexity part of the decoder must not be used for them. This significantly reduces the energy per decoded bit and has almost no impact on the communications performance. Other aspects like message quantization can also be applied to this scheme and generate many new possibilities in the LDPC decoder design space. These new possibilities are currently investigated and must be considered for future architectures.

Regarding energy optimizations the proposed architecture is an excellent candidate for a Near-Threshold circuit technique [49]. For example, the throughput of 10 Gbit/s can already be fulfilled by the presented decoder running at less than 20 MHz. Thus aggressive voltage scaling to 0.5–0.6 V can be applied. This increases the energy efficiency by at least a factor of three and allows for a better energy efficiency than any other state-of-the-art decoder.

Conclusion

In this chapter, we presented soft decision Reed–Solomon, turbo, and LDPC decoder implementations with high throughput.

The introduced soft decision decoder architecture for Reed–Solomon codes is based on information set decoding. It allows a considerable improvement of error rates in combination with a high throughput. The FPGA implementation shows a throughput of beyond 1 Gbit/s and a gain of 0.75 dB over HDD for the widely used RS(255,239) code. Further research includes the evaluation of the architecture using ASIC technology and the further improvement of the correction performance.

For turbo decoding the design space has been summarized. The key techniques to a high throughput implementation have been introduced. It was demonstrated how a LTE turbo code decoder can be implemented that achieves 2.15 Gbit/s on a 65 mm ASIC technology. In the future, further investigations have to be made to ultimately increase the throughput of a turbo code by unrolling iterations.

A new LDPC decoder architecture was presented that achieves an outstanding throughput and state-of-the-art communications performance. The ASIC implementation provides a throughput of 130 Gbit/s and has a very high efficiency. Further optimizations for even higher area and energy efficiency have been discussed and will be investigated in the future.

Acknowledgements This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) within the projects "Entwicklung und Implementierung effizienter Decodieralgorithmen für lineare Blockcodes" and "Optimierung von 100 Gb/s Nahbereichs Funktransceivern unter Berücksichtigung von Grenzen für die Leistungsaufnahme."

References

1. Third Generation Partnership Project (2010) 3GPP TS 36.212 V10.0.0; 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 10)
2. Intel (2014) Thunderbolt. URL http://www.thunderbolttechnology.net
3. Infiniband Association (2014). URL http://www.infinibandta.org
4. Qian D, Huang MF, Ip E, Huang YK, Shao Y, Hu J, Wang T (2011) 101.7-Tb/s (370x294-Gb/s) PDM-128QAM-OFDM transmission over 3x55-km SSMF using pilot-based phase noise mitigation. In: Optical fiber communication conference and exposition (OFC/NFOEC), 2011 and the national fiber optic engineers conference, pp 1–3
5. Wenyi J, Fossorier M (2008) Towards maximum likelihood soft decision decoding of the (255,239) Reed Solomon code. IEEE Trans Magn 44(3):423. DOI 10.1109/TMAG.2008.916381
6. Jiang J (2007) Advanced channel coding techniques using bit-level soft information. Dissertation, Texas A&M University
7. Chase D (1972) Class of algorithms for decoding block codes with channel measurement information. IEEE Trans Inf Theory 18(1):170. DOI 10.1109/TIT.1972.1054746

8. Bellorado J, Kavcic A (2006) A low-complexity method for chase-type decoding of Reed-Solomon codes. In: Proceedings of the IEEE international information theory symposium, pp 2037–2041. DOI 10.1109/ISIT.2006.261907
9. Koetter R, Vardy A (2003) Algebraic soft-decision decoding of Reed-Solomon codes. IEEE Trans Inf Theory 49(11):2809. DOI 10.1109/TIT.2003.819332
10. Fossorier MPC, Lin S (1995) Soft-decision decoding of linear block codes based on ordered statistics. IEEE Trans Inf Theory 41(5):1379. DOI 10.1109/18.412683
11. El-Khamy M, McEliece RJ (2006) Iterative algebraic soft-decision list decoding of Reed-Solomon codes. IEEE J Sel Areas Commun 24(3):481. DOI 10.1109/JSAC.2005.862399
12. An W (2010) Complete VLSI implementation of improved low complexity chase Reed-Solomon decoders. Ph.D. thesis, Massachusetts Institute of Technology
13. García-Herrero F, Valls J, Meher P (2011) High-speed RS(255, 239) decoder based on LCC decoding. Circuits Syst Signal Process 30:1643. DOI 10.1007/s00034-011-9327-4. URL http://dx.doi.org/10.1007/s00034-011-9327-4
14. Hsu CH, Lin YM, Chang HC, Lee CY (2011) A 2.56 Gb/s soft RS (255,239) decoder chip for optical communication systems. In: Proceedings of the ESSCIRC (ESSCIRC), pp 79–82. DOI 10.1109/ESSCIRC.2011.6044919
15. Kan M, Okada S, Maehara T, Oguchi K, Yokokawa T, Miyauchi T (2008) Hardware implementation of soft-decision decoding for Reed-Solomon code. In: Proceedings of the 5th international symposium on turbo codes and related topics, pp 73–77. DOI 10.1109/TURBOCODING.2008.4658675
16. Heloir R, Leroux C, Hemati S, Arzel M, Gross W (2012) Stochastic chase decoder for reed-solomon codes. In: 2012 IEEE 10th international conference on new circuits and systems (NEWCAS), pp 5–8. DOI 10.1109/NEWCAS.2012.6328942
17. Scholl S, Wehn N (2014) Hardware implementation of a Reed-Solomon soft decoder based on information set decoding. In: Proceedings of the design, automation and test in Europe (DATE '14)
18. Ahmed A, Koetter R, Shanbhag NR (2004) Performance analysis of the adaptive parity check matrix based soft-decision decoding algorithm. In: Proceedings of the conference on signals, systems and computers record of the thirty-eighth Asilomar conference, vol 2, pp 1995–1999. DOI 10.1109/ACSSC.2004.1399514
19. Dorsch B (1974) A decoding algorithm for binary block codes and$J$-ary output channels (Corresp.). IEEE Trans Inf Theory 20(3):391. DOI 10.1109/TIT.1974.1055217. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1055217
20. Lin S, Costello DJ Jr (2004) Error control coding 2nd edn. Prentice Hall PTR, Upper Saddle River
21. Scholl S, Stumm C, Wehn N (2013) Hardware implementations of Gaussian elimination over GF(2) for channel decoding algorithms. In: Proceedings of the IEEE AFRICON
22. Bogdanov A, Mertens M, Paar C, Pelzl J, Rupp A (2006) A parallel hardware architecture for fast Gaussian elimination over GF(2). In: 14th annual IEEE symposium on field-programmable custom computing machines, 2006 (FCCM '06), pp 237–248. DOI 10.1109/FCCM.2006.12
23. Kung HT, Gentleman WM (1982) Matrix triangularization by systolic arrays. Technical Report Paper 1603, Computer Science Department. URL http://repository.cmu.edu/compsci/1603
24. Xilinx LogiCORE IP Reed-Solomon Decoder (2013). http://www.xilinx.com/products/intellectual-property/DO-DI-RSD.htm
25. Bahl L, Cocke J, Jelinek F, Raviv J (1974) Optimal decoding of linear codes for minimizing symbol error rate. IEEE Trans Inf Theory IT-20:284
26. Robertson P, Villebrun E, Hoeher P (1995) A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log-domain. In: Proceedings of the 1995 international conference on communications (ICC '95), Seattle, Washington, 1995, pp 1009–1013
27. Thul MJ, Gilbert F, Vogt T, Kreiselmaier G, Wehn N (2005) A scalable system architecture for high-throughput turbo-decoders. J VLSI Signal Process Syst (Special Issue on Signal Processing for Broadband Communications) 39(1/2):63
28. Mansour MM, Shanbhag NR (2003) VLSI architectures for SISO-APP decoders. IEEE Trans Very Large Scale Integr (VLSI) Syst 11(4):627

29. Dielissen J, Huiskens J (2000) State vector reduction for initialization of sliding windows MAP. In: Proceedings of the 2nd international symposium on turbo codes & related topics, Brest, France, pp 387–390
30. Schurgers C, Engels M, Catthoor F (1999) Energy efficient data transfer and storage organization for a MAP turbo decoder module. In: Proceedings of the 1999 international symposium on low power electronics and design (ISLPED '99), San Diego, California, 1999, pp 76–81
31. Sani A, Coussy P, Chavet C (2013) A first step toward on-chip memory mapping for parallel turbo and LDPC decoders: a polynomial time mapping algorithm. IEEE Trans Signal Process 61(16):4127. DOI 10.1109/TSP.2013.2264057. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6517513
32. Ilnseher T, Kienle F, Weis C, Wehn N (2012) A 2.12Gbit/s turbo code decoder for LTE advanced base station applications. In: 2012 7th international symposium on turbo codes and iterative information processing (ISTC) (ISTC 2012), Gothenburg, Sweden, 2012
33. Sun Y, Cavallaro J (2010) Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder. Integr VLSI J. DOI 10.1016/j.vlsi.2010.07.001
34. May M, Ilnseher T, Wehn N, Raab W (2010) A 150Mbit/s 3GPP LTE turbo code decoder. In: Proceedings of the design, automation and test in Europe, 2010 (DATE '10), pp 1420–1425
35. Studer C, Benkeser C, Belfanti S, Huang Q (2011) Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE. IEEE J Solid State Circuits 46(1):8
36. Gallager RG (1962) Low-density parity-check codes. IRE Trans Inf Theory 8(1):21
37. IEEE 802.3an-2006 (2006) Part 3: CSMA/CD Access Method and Physical Layer Specifications - Amendment: Physical Layer and Management Parameters for 10 Gb/s Operation, Type 10GBASE-T. IEEE 802.3an-2006
38. WiMedia Alliance (2009) Multiband OFDM Physical Layer Specification, Release Candidate 1.5
39. IEEE 802.11ad (2010) Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment: Enhancements for Very High Throughput in the 60 GHz Band. IEEE 802.11ad-draft
40. Kavcic A, Patapoutian A (2008) The read channel. Proc IEEE 96(11):1761. DOI 10.1109/JPROC.2008.2004310
41. Guilloud F, Boutillon E, Danger J (2003) λ-min decoding algorithm of regular and irregular LDPC codes. In: Proceedings of the 3nd international symposium on turbo codes & related topics, Brest, France, pp 451–454
42. Chen J, Dholakia A, Eleftheriou E, Fossorier MPC, Hu XY (2005) Reduced-complexity decoding of LDPC codes. IEEE Trans Commun 53(8):1288
43. Schläfer P, Alles M, Weis C, Wehn N (2012) Design space of flexible multi-gigabit LDPC decoders. VLSI Des J 2012. DOI 10.1155/2012/942893
44. Blanksby A, Howland CJ (2002) A 690-mW 1-Gb/s, rate-1/2 low-density parity-check code decoder. IEEE J Solid State Circuits 37(3):404
45. Onizawa N, Hanyu T, Gaudet V (2010) Design of high-throughput fully parallel LDPC decoders based on wire partitioning. IEEE Trans Very Large Scale Integr (VLSI) Syst 18(3):482. DOI 10.1109/TVLSI.2008.2011360
46. Schläfer P, Wehn N, Lehnigk-Emden T, Alles M (2013) A new dimension of parallelism in ultra high throughput LDPC decoding. In: IEEE workshop on signal processing systems (SIPS), Taipei, Taiwan
47. Weiner M, Nikolic B, Zhang Z (2011) LDPC decoder architecture for high-data rate personal-area networks. In: Proceedings of the IEEE international symposium on circuits and systems (ISCAS), pp 1784–1787. DOI 10.1109/ISCAS.2011.5937930
48. Zhang Z, Anantharam V, Wainwright M, Nikolic B (2010) An efficient 10GBASE-T ethernet LDPC decoder design with low error floors. IEEE J Solid State Circuits 45(4):843. DOI 10.1109/JSSC.2010.2042255
49. Calhoun B, Brooks D (2010) Can subthreshold and near-threshold circuits go mainstream? IEEE Micro 30(4):80. DOI 10.1109/MM.2010.60

http://www.springer.com/978-3-319-10568-0

Advanced Hardware Design for Error Correcting Codes

Chavet, C.; Coussy, P. (Eds.)

2015, IX, 192 p. 81 illus., 25 illus. in color., Hardcover
ISBN: 978-3-319-10568-0