Adaptive Architectures for Forward Error Correction

Sergei Sawitzki
FH Wedel (University of Applied Sciences)
Wedel, Germany
e-mail: Sergei.Sawitzki@fh-wedel.de

***Abstract*—This work discusses adaptive designs of decoders for several forward error correction codes. In the first scenario, decoders capable of decoding the codes with different parameters belonging to the same family are introduced. The results suggest, that the hardware overhead caused by the additional flexibility in most cases is as low as 5 – 10% additional silicon footprint compared to the implementation based on the fixed code parameters. In the second scenario, reconfigurable designs of multi-family decoders are discussed. Since some parts of the data-path and internal memory can be reused for different decoders, silicon area savings of more than 40 % are achievable compared to the overall chip area costs of the individual decoder implementation per code family. The overall usefulness of such reconfigurable decoder designs depends on the application case, for instance, the throughput requirements or the necessity to process data streams encoded with different codes in parallel. The figures reported herein summarize and extend some previous work known from literature, as well as the research carried out by the author.**

Keywords-forward error correction; FEC decoders; adaptive decoder architectures.

I. Introduction and Related Work

Most of the modern digital communication and data storage standards enforce the use of different error correcting codes. Since in most cases the error correction process is carried out by the receiver – without resending or rereading the data packet – these codes are usually referred to as Forward Error Correcting (FEC) codes and the respective hard- or software units are called FEC encoders and decoders. The FEC codes most widely used nowadays are belonging to one of the following code families: convolutional codes, Reed-Solomon (RS) codes, Low-Density Parity-Check (LDPC) codes, and turbo codes. Some standards specify varying codes for different data rates or combine codes from different families within one run to address specific channel properties like cross-talk and fading improving the overall bit error rate.

Over the last few years, mobile devices got significantly more flexible. Modern mobile phones and other wearable devices support numerous communication interfaces providing the user with almost unlimited connectivity. Even the low-price devices nowadays support wireless LAN, different mobile internet standards, and Bluetooth, just to name a few. At least some of them rely on the same kind of FEC algorithms, although the specific parameters like code generator polynomials, coding rates, constraint length, block sizes, and so on may vary even within the same communication standard. To address this issue, adaptive Viterbi and RS decoders will be discussed below and compared to the straight-forward implementation for the single code instance. Such comparison will provide a glimpse of the overhead introduced by the additional flexibility.

On the other side, some computations carried out during the decoding process of different code families are quite similar. In addition, some data need to be temporarily stored either throughout the complete decoding procedure or, in case of LDPC and turbo codes, between adjacent decoding iterations resulting in specific memory requirements. This poses the question, if at least some parts of the silicon dedicated to the data-path and memory can be reused among different standards reducing the silicon footprint and improving the power balance compared to the straight-forward implementation of every single decoder. This question will be addressed in the discussion of combined Viterbi/turbo and LDPC/turbo decoders.

The general applicability and advantages of the adaptive FEC decoder implementation depend on the use case, e.g., the necessity to process several data streams encoded with the same code or with different codes in parallel or the option to switch between different data rates. This should be kept in mind during the discussion of the different design options.

In principle, FEC decoding algorithms can be implemented both in software and hardware. However, given the fact, that most of them are quite computationally intensive, software-based solutions are usually lagging behind the high throughput requirements imposed by the majority of contemporary communication and data storage standards. For instance, iterative decoding involved in the decoding process of turbo and LDPC codes may require as many as 1 500 machine instructions per bit (or even more depending on the number of decoding iterations) [1]. Running such decoders at the data rates of several hundreds of megabits per second requires computational power which is beyond the capabilities of even most powerful microprocessors. For this reason, the scope of this paper is limited to the dedicated hardware architectures, i.e., direct mapping of the FEC decoders to silicon. Alternative approaches which are less flexible than the software solution but still offer at least some properties of the architectures built around the instruction set paradigm are Application-Specific Instruction-set Processors (ASIP) [2][3] and Networks-on-Chip (NoC) [4]. These are beyond the scope of this contribution. A comprehensive overview concerning the scalability issues and multi-standard capabilities of different FEC decoders is provided in [1][5]. Based upon these studies, some of the results are re-evaluated, supported by new findings and extended herein.

The rest of this paper is organized as follows. Section II introduces an adaptive implementation of the Viterbi decoder and compares several designs known from literature. In Sec-

 ISBN: 978-1-68558-308-8

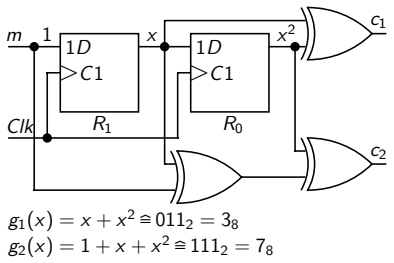

Figure 1. Simple convolutional encoder.

tion III, the additional costs of a reconfigurable architecture for a Reed-Solomon decoder are discussed. Section IV summarizes the general options of combining decoders for different code families within one design and analyzes the architectures of reconfigurable Viterbi-turbo and LDPC-turbo decoders in more detail. Finally, Section V provides a summary of this work and draws some conclusions concerning adaptive and multi-family FEC decoder implementation showing some directions for future research.

Due to the limited space, the detailed description of the coding schemes and decoder architectures is omitted. Instead, the key ideas are summarized and some references for the readers unfamiliar with the matter are provided where appropriate.

II. Viterbi Decoders

Convolutional codes were introduced in 1955 [6] and became one the most widely applied FEC code family. The incoming data stream is stored in a shift register, from which several bits are combined using an XOR operator. The number of delay stages in the shift register including the input of the first delay stage is referred to as *constraint length* K. Larger K values provide better error correction capabilities (at the cost of the increasing decoder complexity). Constraint lengths between 5 and 11 are common. Particular XOR operators are selected to produce the output data streams. The result is the discrete convolution of the input data with encoder's impulse responses in the time domain, hence the name of the code [7, pp. 455–456]. The number n of the output data streams defines the code rate $1/n$. For the codes used in most communication standards 2 to 4 output bits are produced for every input bit, so the code rate R lies between 1/2 and 1/4.

The particular codes differ depending on which bits from the sequence stored in the shift register are involved in the XOR calculation. This information is provided in the form of the so-called generator polynomials g, one for every output. Figure 1 shows an example of a simple convolutional encoder with $K = 3$, $R = 1/2$, $g_1 = 011$ and $g_2 = 111$. As can be seen, the generator polynomials are often specified by the binary or octal representation of their coefficients. The encoder is a Finite-State Machine (FSM) whose output depends on g.

The decoding process is based on the maximum-likelihood principle. Decoder's task is to find the state transition sequence, which encoder FSM most likely underwent based on its output sequence, which in most cases is corrupted by noise during the transmission. Afterwards, the state transitions can be mapped to the corresponding input data sequence. The most widely used decoding algorithm for convolutional codes is based on dynamic programming and was introduces in 1967 by Andrew Viterbi [8]. In the first stage of the decoding process, every received symbol is compared to all legal output symbols of the encoder. The result is called Branch Metric (BM). In the simplest case of the so-called hard-input decoding, BM is the Hamming distance. Due to modulation and analog-digital conversion the input symbols are usually integer numbers represented by several bits. This case is referred to as soft-input decoding, where BM is represented by other measures like squared euclidean distance. The number of BM values to be calculated depends on the code rate since every additional output bit increases the number of the legal output symbols by the factor of two and every received symbol needs to be compared with all of them.

In the second stage, branch metrics are used to calculate the Path Metrics (PM). One PM needs to be calculated for every state of the encoder. All path metrics are set to zero in the beginning and updated with every processed symbol. The corresponding operation is called Add-Compare-Select (ACS) and is the computational core of the decoding process. Every ACS unit consists of two adders, one comparator and one multiplexer. In addition, one register is required per unit to store the actual PM value. The number of ACS units is equal to 2^{K-1}, i.e., it increases exponentially with the constraint length of the code. The result of the ACS calculation is the new path metric and the binary decision, which state transition the encoder underwent while the corresponding output symbol was generated. This decision is stored for every processed symbol and every ACS unit in form of a single bit called *decision bit*. The decision bits represent several paths through the possible state transition sequences of the encoder with every path corresponding to the certain input sequence. It can be shown, that under normal conditions all paths merge to a single one after a certain number of steps, which ist called Trace-Back Depth (TBD). This path represents the most probable sequence of the encoder's state transitions. As a consequence, decoder needs to store the decision bits for at least the number of symbols equal to TBD. Given the fact, that one decision bit is produced by every ACS unit, the memory capacity for a single trace-back run equals to TBD$\times 2^{K-1}$. As the decision bits cannot be overwritten during the trace-back, this amount needs to be at least doubled. The exact value of the TBD depends on the channel conditions. For AGWN channels TBD values of $5 \times K$ are sufficient, but they may be higher than $20 \times K$ for channels with fading and multi-path propagation.

Trace-back is the final stage of the decoding process, in which the decision bits corresponding to the most probable path are traversed and the corresponding input bits are stored in a Last-In First-Out (LIFO) buffer. Finally, the LIFO memory is read to produce the decoded input sequence (which in optimal case will be the same as the original input to the encoder).

A Viterbi decoder for a specific convolutional code has fixed design parameters like the number of the BM and the ACS units, TBD, and so on. It is quite different in the case of the adaptive decoder. Figure 2 shows the ACS calculation

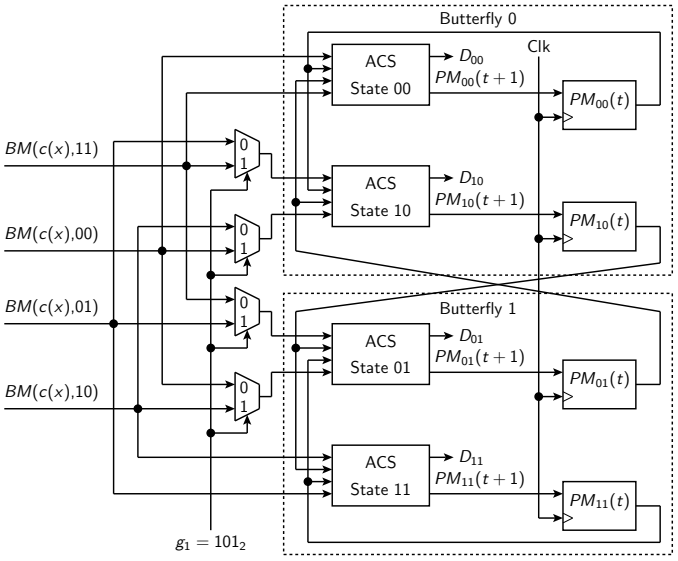

Figure 2. Path metric calculation for a reconfigurable Viterbi decoder.

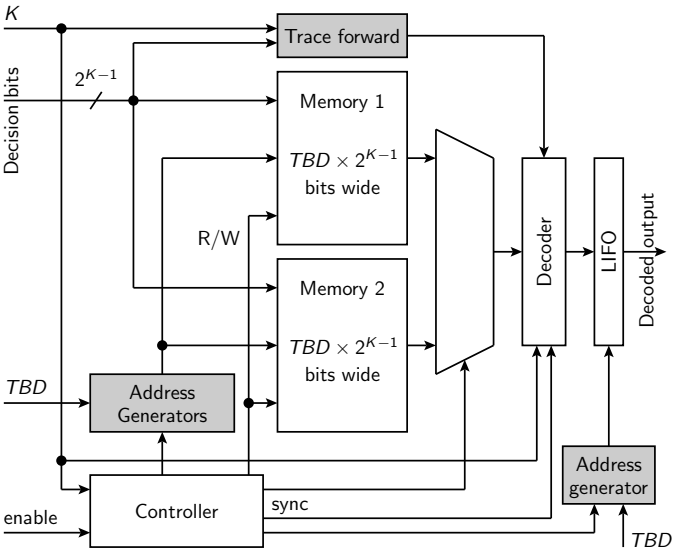

Figure 3. Trace back unit for a reconfigurable Viterbi decoder with clock and reset signals not shown for the sake of clarity.

unit for a decoder which can decode the code produced by the encoder in Figure 1, as well as the code generated by $g_1 = 101$ and $g_2 = 111$. While for each particular code the interconnect between the BM calculation outputs and PM calculation inputs is fixed, the adaptive decoder needs to switch between two different patterns depending on the code to be decoded. This functionality is realized by the multiplexers placed at the inputs of the particular ACS units. If the decoder has to process codes with different constraint lengths, additional flexibility is required in the interconnect pattern between the registers storing the actual PM values and the inputs of the ACS units as well. This means more and larger multiplexers at the inputs reducing the clock frequency and the throughput. A detailed design of the interconnect network for an adaptable Viterbi Decoder including the particular interconnect patterns for different settings is described in [9].

In general, varying the constraint length and/or the generator polynomials has the strongest impact on the additional chip area and the critical path of the decoder. For the trace-back stage, the memory architecture has to be calculated for the worst case, i.e., the largest K and TBD values. In addition, the addressing scheme has to be adapted to every single case, which imposes the need for programmable address counters. Compared to other modifications this overhead is almost neglectable. Figure 3 shows the trace-back unit of the reconfigurable Viterbi decoder for variable values of the constraint length. The building blocks affected by the reconfigurable nature of the design are shaded. In principle, the address generators could be dimensioned for the worst case, just like the memories. However, larger TBD increases the latency of the decoder, which would be an unnecessary overhead for smaller K values. At the same time, making the address generators pre-loadable and scalable adds almost nothing to the overall silicon footprint of the decoder. The trace forward block depicted in Figure 3 is used to calculate the initial state of the new trace-back cycle and is described in detail in [10]. One memory block is used to store the actual decision bits (write access) while the other is used to trace back the decision bits from the previous cycle (read access). The memories switch their roles after every cycle, which explains the need for the multiplexer at their read ports.

TABLE I. COMPARISON OF ADAPTABLE AND SINGLE-CODE VITERBI DECODERS (b REFERS TO THE INPUT BIT-WIDTH).

Architecture	Metric	Reference	Result
[11] $K = 3 \dots 7$, g variable, b fixed	silicon area	$K = 7$, g fixed	$+2,9\,\%$
	throughput	$K = 7$, g fixed	$-1,5\,\%$
VITURBO [12] $K = 3 \dots 9$, g variable, b fixed	silicon area	$K = 9$, g variable	$+9\,\%$
	max. clock frequency	$K = 9$, g variable	$-30\,\%$
[13] $K \in \{7, 9\}$, g according to EDGE, CMDA2000 and WCDMA, b fixed	silicon area	$K = 9$, g fixed (CDMA2000)	overhead is neglectable
	critical path	$K = 9$, g fixed (CDMA2000)	$+4\,\%$
[14] $K = 3 \dots 11$, g variable, $n = 2 \dots 4$, $b = 1 \dots 5$	silicon area	$K = 11$, $n = 4$, $b = 5$, g fixed	$+50,1\,\%$
	throughput	$K = 11$, $n = 4$, $b = 5$, g fixed	$-26,1\,\%$

Table I summarizes the figures of different adaptable Viterbi decoder designs. The major findings can be generalized as follows:

- The overhead increases with the degree of flexibility. The Viterbi decoders designed to support a (small) fixed amount of codes introduce the least – in some cases even neglectable – overhead. In contrast, the full flexibility regarding the constraint length, the code rate as well as the generator polynomials requires up to 50 % more silicon area and reduces the throughput by about 26 %. Although looking diminishing at the first glance, it may still be quite a good price to pay considering the alternative to implement a dedicated decoder for every single code.

- Varying the constraint length K has the highest impact on the area overhead of the adaptive Viterbi decoder implementation. Two factors can be identified as being responsible for this matter. On one hand, the number of ACS units increases exponentially with K, which means, the number and the size of the multiplexers required to implement different interconnect patterns increases accordingly. On the other hand, the TBD value grows linearly with K, so the memory depth of the trace-back grows accordingly. At the same time, the memory width grows exponentially with K. These consequences become crucial for higher values of K: the area overhead for $K = 9$ is about 9 %, while for $K = 11$ it is more than 50 %. For Viterbi decoders with smaller constraint lengths the area overhead is almost neglectable.

III. Reed-Solomon Decoders

The basic idea of the error correcting linear cyclic codes nowadays knows as Reed-Solomon codes can be traced back to 1952 [15][16]. In its current form the codes were introduced in the seminal paper by Irving S. Reed and Gustave Solomon, hence the name [17]. RS codes are defined over a finite field $\mathrm{GF}(q^m)$, which is an extension field of $\mathrm{GF}(q)$, where q is a prime number and m is a natural number different from zero. For all practically applied codes $q = 2$ is chosen, which means, that the number of the elements in the field is a power of two. Binary representation of the elements of $\mathrm{GF}(2^m)$ requires m bits per element. This is the reason, why almost all RS codes used in communication and storage systems are defined over $\mathrm{GF}(2^8)$: each symbol can be encoded by exactly one byte. In contrast to the most other code families, the error correcting properties of the RS codes are defined symbol-wise, i.e., it does not matter, if a single bit or any other number of bits within one symbol are corrupted – the decoder treats it as a single symbol error. This explains, why RS codes are very powerful at correcting burst errors. A (255,239) RS code has the block size of 255 bytes, with 239 bytes of original message data and $2t = 255 - 239 = 16$ bytes of the checksum added during encoding. $t = 16/2 = 8$ is the number of corrupted symbols which can be corrected by the code. If 8 adjacent symbols (64 adjacent bits in the worst case) would be corrupted during the transmission, the decoder would still be able to correct all of them. The general notation used to specify RS codes is (n, k), where n is the block size and k is the original message size before encoding (both specified in symbols of m bits width). As stated in the example above $n - k = 2t$, where t is the number of correctable symbol errors and $R = k/n$ is the code rate.

The t parameter can be chosen depending on the required error correction capacity of the code. Once fixed, it defines the generator polynomial for the code, which has the degree $2t$:

$$\mathbf{g}(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{2t-1} x^{2t-1} + x^{2t}. \quad (1)$$

The coefficients $g_0, \ldots, g_{2t-1}$ are all elements of $\mathrm{GF}(2^m)$ and the polynomial itself is defined in such a way, that its $2t$ roots correspond to the $2t$ consecutive powers of the single element of $\mathrm{GF}(2^m)$. For practical reasons, usually the primitive element of the extension field $\mathrm{GF}(q^m)$ is chosen as the first root (being the generator of the multiplicative group of this field).

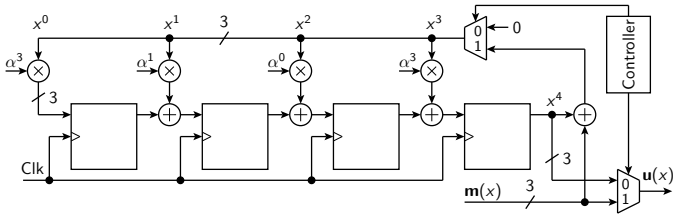

Figure 4. Reed-Solomon encoder for a (7,3) code over $\mathrm{GF}(2^3)$.

The encoding procedure appends the rest $\mathbf{p}(x)$ of the division of the polynomial representing the original message $\mathbf{m}(x)$ shifted to the left by $n-k$ symbols by the generator polynomial:

$$x^{n-k}\mathbf{m}(x) = \mathbf{q}(x)\mathbf{g}(x) + \mathbf{p}(x), \quad (2)$$

which is equivalent to

$$\mathbf{p}(x) = x^{n-k}\mathbf{m}(x) \bmod \mathbf{g}(x). \quad (3)$$

This operation can be easily carried out using feedback shift-register as illustrated in Figure 4 for a very simple (7,3) RS code. Note, that addition and multiplication operations are defined on the elements of the corresponding finite field (i.e., not in terms of common arithmetic).

Given the fact, that the encoding process can be represented by means of polynomial multiplication over $\mathrm{GF}(q^m)$, it becomes clear, how the transmission errors can be recognized (and corrected). Since multiplication operation preserves the roots of the generator polynomial, the resulting encoded message can be inspected by the decoder by checking if $2t$ consecutive powers of the primitive element of $\mathrm{GF}(q^m)$ are the roots of the polynomial corresponding to the received message. This is the first step of the decoding process which is called syndrome calculation. If at least one of the syndromes is not equal to zero, then the message was corrupted.

The syndrome values are used in the next step of the decoding process to calculate the number of the corrupted symbols. The most intuitive way to accomplish this is – in the first step – to assume the number of errors to be one and try to solve the corresponding system of equations based on the co-called error location polynomial. The solution indicates the position of the corrupted symbol. In case the system of equations for a single error is not solvable, the assumption of two corrupted symbols has to be made resulting in a different system of equations. The process is repeated until a solvable system of equations is found. This procedure is called Peterson-Zierler-Gorenstein algorithm [18][19]. For the hardware implementation the Berlekamp-Massey algorithm [20] is better suited, since it can be better parallelized and modified to reuse the same hardware to calculate both error location and error value polynomials [21]. Its detailed description, however, would go far beyond the scope of this paper. Independent of the choice of the algorithm, finite field arithmetic is heavily involved in the search for the error positions and values.

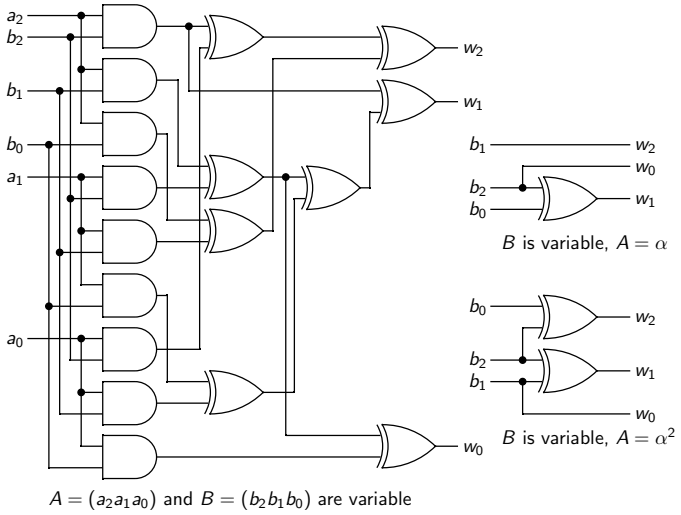

Figure 5. Variable-variable versus constant-variable multiplier for $\mathrm{GF}(2^3)$.

As just stated, since RS codes work on symbols of m bits width, knowing the symbol error position is not enough. An additional step is required to determine the error values. This can be done by using the syndromes computed in the first step, since every set of syndromes can be mapped to certain errors. Alternatively, the so-called error value polynomial can be calculated [22]. In the final decoding step, the error values are added to the received symbols at the calculated error positions restoring the original message. To accomplish this, the received (corrupted) data is stored in the First-In First-Out (FIFO) memory making sure the right values appear at the output of the decoder at the right time. The finite filed addition corresponds to a bitwise XOR operation, which scales well and is trivial from the implementation point of view.

As mentioned above, decoding the RS codes requires numerous calculations based on the finite field arithmetic. In an RS decoder designed for a specific code, many of the GF-multipliers have one constant input (variable-constant multipliers). For adaptive decoding of different codes, a variable-variable multiplier is required. The corresponding overhead is quite high as Figure 5 illustrates. However, since only multipliers involved in the syndrome and error location computation are affected, the impact on the area of the whole RS decoder is limited.

Table II summarizes the area and throughput figures of different RS decoder designs. For the adaptive decoding, n, t, and m parameters can be changed, although changing m does not make a lot of sense, since almost all real-world RS codes are based on $m = 8$ (ITU-T J.83, IEEE802.3bp, and IEEE802.3bj standards being an exception by specifying codes based on $\mathrm{GF}(2^7)$, $\mathrm{GF}(2^9)$, and $\mathrm{GF}(2^{10})$ correspondingly). In addition, different polynomials can be chosen as the finite field generators. Varying the GF generator polynomial $\mathbf{f}(x)$ results in a different encoding of the particular elements of the finite field. However, it has been shown, that all finite fields for the fixed values of q and m are isomorphic.

TABLE II. COMPARISON OF ADAPTABLE AND SINGLE-CODE REED-SOLOMON DECODERS.

Comparison metric	Decoder architecture				
	[24]	[25]	[26]	[23] 1st variant	[23] 2nd variant
Code parameter	fixed	n and t variable	Universal decoder: n, t, m and $\mathbf{f}(x)$ variable		
m	8	8	$1 \ldots 8$	$1 \ldots 10$	$1 \ldots 8$
t	8	$1 \ldots 8$	$1 \ldots 8$	$1 \ldots 8$	$1 \ldots 16$
Erasure correction	no	no	no	≤ 16	≤ 16
Throughput in Gb/s	1,6	0,8	0,048	2,2	2,4
Gate equivalents	21 000	34 000	44 000	75 000 + 35 Kbit RAM	39 000 + 15 Kbit RAM

The conclusions from the study of the different RS decoder designs are as follows:

- The area overhead of the adaptive RS decoder implementation can be quite significant. Compared to the fixed-code decoder with $m = t = 8$, a fully reconfigurable design can consume about 85 % more silicon area in terms of gate equivalents in addition to 15 KBit more SRAM. Just like in the case of adaptive Viterbi decoder this is still quite a low price to pay, since the error correction capabilities are much better and the additional erasure correcting feature is useful in many application scenarios (erasures are errors whose position is known in advance).
- Throughput is usually not an issue with RS decoding, since the symbol-per-second decoding rate is multiplied by m to obtain the bit-per-second values. For adaptive decoding it means, that the reduced clock rates caused by the additional overhead of the adaptive implementation are not significant in most cases.

IV. MULTI-FAMILY DECODERS

As can be seen from previous sections, the arithmetic involved in decoding of convolutional and RS codes is quite different which limits the possibilities of senseful hardware reuse. However, it may be an option to combine decoders for the other code families within one design with Viterbi decoder or with each other.

Turbo codes were introduced in 1993 [27]. The basic idea is to use several (usually two) recursive convolutional encoders to process the same input data. First encoder receives the data directly, the second one gets an interleaved data stream. The size of the interleaver is usually in the order of several kilobits. This approach makes the same data appearing as two statistically independent messages. The decoding is done in an iterative manner with two decoders which process the received data and pass the output to each other (with interleaving and deinterleaving steps inbetween). The decoders are based on the Soft-Input Soft-Output (SISO) principle, i.e., their output is a metric providing the likelihood for a bit to be 0 or 1 instead of the final bit decision (soft-input was already discussed in the Section II). The idea is, that a SISO decoder uses the output of its predecessor as a-priori information to improve its own

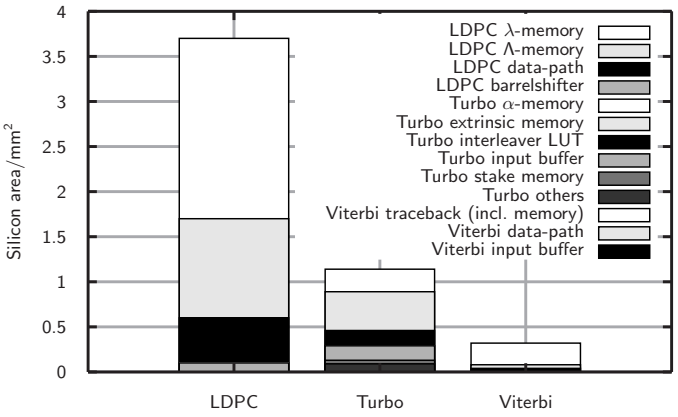

Figure 6. Chip area distribution for different FEC decoder designs (adapted from [1]).

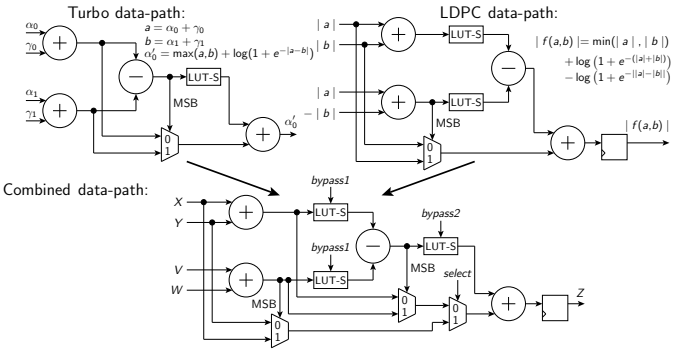

Figure 7. Combining Turbo and LDPC data-paths within a single FFU (adapted from [30], LUT-S are Look-Up-Tables storing the approximated logarithmic values).

decisions, which are then passed back to the first decoder and so on. The certainty of the calculated metrics improves over time, so that the decoding process usually can be stopped after several iterations.

LDPC codes were described in detail in the dissertation of Robert Gallager in 1963 [28]. Due to the high computational requirements of the decoding process, first practical applications came more that 30 years later as LDPC were proposed as channel codes for several communication standards. In a nutshell, LDPC codes are linear block codes with very large, very sparsely filled generator matrix. As in the case of turbo codes, the decoding process is iterative. In one iteration, parities are checked to determine, which bits of the information block are responsible for most unsatisfied parity equations. These bits are inverted and the next iteration starts. The decoding ends, as soon as all equations are satisfied or a certain number of iterations is reached. As in the case of turbo codes, most applications use soft metrics during the decoding process, which are converted to the hard bit decisions at the end.

According to this brief description of the iterative decoding, it should be clear that a lot of local memory is required to store the information bits and intermediate results. Figure 6 shows the silicon area distribution between different parts of the design for LDPC, turbo, and Viterbi decoders. The extreme disproportion in the required silicon area between LDPC and Viterbi decoders suggests, that a combination of both within the same design with the goal to reuse some of the resources does not make sense. Even if the complete Viterbi decoder area could be reused for LDPC (which is virtually impossible), the overall gain would sum up to less than 10 % of the silicon footprint.

At the same time, the SISO module required for turbo decoding can be implemented based on the Soft-Output Viterbi-Algorithm (SOVA), which can also be used to decode the convolutional codes. This increases the potential for hardware reuse at the cost of slightly reduced bit error ratio for turbo decoder compared to the SISO module based on the Max-log-MAP algorithm [29].

A closer look at the internal data-path of the turbo and LDPC decoders reveals some similarities as well. Figure 7 shows, how the calculations involved in both decoding algorithms can be combined within the same Flexible Functional Unit (FFU) [30]. As in the case of multi-standard Viterbi decoder, some additional multiplexers are required to switch between the codes, which slightly increases the critical path.

TABLE III. COMPARISON OF MULTI-FAMILY FEC DECODERS.

Architecture	Metric	Reference	Results
Viterbi-Turbo Decoders			
VITURBO [12]	silicon area	Viterbi decoder ($K = 3 \ldots 9$)	+5 % logic +25 % memory
[29] (fully parallel)	silicon area	separate turbo and Viterbi decoder	$-14,5\,\%$
	throughput	separate turbo and Viterbi decoder	no change
[29] (time-multiplex)	silicon area	separate turbo and Viterbi decoder	$-28,1\,\%$
	throughput	separate turbo and Viterbi decoder	same for Viterbi 1/2 for turbo
[31]	silicon area	Turbo decoder	$+20\,\%$
Turbo-LDPC Decoders			
[1]	silicon area	separate turbo and LDPC decoder	$-10\,\%$
[30]	silicon area	turbo decoder LDPC decoder reused vs. separate data-paths	$+10 \ldots +20\,\%$ $+15 \ldots +20\,\%$ $-39 \ldots -42\,\%$

Accordingly, the only options for multi-family decoders are Viterbi-Turbo or Turbo-LDPC designs. Some case studies of such designs are known from literature. Table III summarizes the results. As expected, some portions of the data-path and memory can be reused resulting in silicon area savings of 10–42 % while throughput is not affected in most cases of combined Viterbi-turbo decoder. If adaptable designs are compared with a single family decoder, the overhead of introducing an additional code family is very low. For instance, adding the LDPC functionality to the existing turbo design comes at only 10–20 % additional silicon area [30].

V. CONCLUSION AND FUTURE WORK

This paper discussed several options of the adaptive FEC decoder design. The findings based on literature study and own research suggest that overhead of extending a decoder

towards other codes of the same family comes at moderate cost. As expected, this overhead increases with the amount of flexibility and code complexity. At the same time, decoders covering several code families can take advantage of the resource sharing reducing the overall silicon footprint compared to the dedicated implementation of one particular decoder per family. One aspect not discussed herein is the impact of the adaptive decoder implementation on the power consumption. On one hand, reconfigurability comes at the cost of additional area and thus should result in increased energy-per-bit figures. On the other hand, it can be expected that reusing parts of the data-path and memory for different decoding algorithms leads to overall power savings. Supporting these assumptions by concrete numbers is a promising direction for further research.

REFERENCES

[1] J. T. M. H. Dielissen, N. Engin, S. Sawitzki, and C. H. van Berkel, "FEC Decoders for Future Wireless Devices: Scalability Issues and Multi-Standard Capabilities", in *Circuits and Systems for Future Generations of Wireless Communications*, A. Tasić, W. A. Serdijn, L. E. Larson, and G. Setti, Eds., ser. Series on Integrated Circuits and Systems, Berlin: Springer, 2009, pp. 271–297.

[2] T. Vogt and N. Wehn, "A Reconfigurable ASIP for Convolutional and Turbo Decoding in an SDR Environment", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 10, pp. 1309–1320, Oct. 2008.

[3] P. M. Velayuthan, "Towards Optimized Flexible Multi-ASIP Architectures for LDPC/Turbo Decoding", PhD thesis, Université de Bretagne-Sud, Dec. 2012.

[4] M. Scarpellino, A. Singh, E. Boutillon, and G. Masera, "Reconfigurable Architecture for LDPC and Turbo Decoding: A NoC Case Study", in *Proceedings of IEEE 10th International Symposium on Spread Spectrum Techniques and Applications*, Bologna, Italy: IEEE, Aug. 2008, pp. 671–676.

[5] J. T. M. H. Dielissen, N. Engin, S. Sawitzki, and C. H. van Berkel, "Multi-Standard FEC Decoders for Wireless Devices", *IEEE Transactions on Circuits and Systems II*, vol. 55, no. 3, pp. 284–288, May 2008.

[6] P. Elias, "Coding for Noisy Channels", *IRE Convention Record*, vol. 3, no. 4, pp. 37–46, 1955.

[7] T. K. Moon, *Error Correction Coding. Mathematical Methods and Algorithms*. John Wiley & Sons, 2005.

[8] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[9] W. Tang, N. Engin, F. A. Steenhof, M. Klaassen, A. Hekstra, and S. Sawitzki, *Multi-Standard Viterbi Processor*, US Patent 8,904,266 B2, NXP B. V., Eindhoven, The Netherlands, Dec. 2014.

[10] P. J. Black and T. H. Meng, "Hybrid Survivor Path Architectures for Viterbi Decoders", in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Minneapolis, MN, USA, Apr. 1993, pp. 433–436.

[11] K. Chadha and J. R. Cavallaro, "A Reconfigurable Viterbi Decoder Architecture", in *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems & Computers*, M. B. Matthews, Ed., vol. 1, Pacific Grove, CA, USA: IEEE, Nov. 2001, pp. 66–71.

[12] J. R. Cavallaro and M. Vaya, "VITURBO: A Reconfigurable Architecture for Viterbi and Turbo Decoding", in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Hong Kong: IEEE, Apr. 2003, pp. 497–500.

[13] T. Vogt, N. Wehn, and P. Alves, "A Multi-Standard Channel-Decoder for Base-Station Applications", in *17th Symposium on Integrated Circuits and Systems Design (SBCCI 2004)*, Porto de Galinhas, Brazil: IEEE, Sep. 2004, pp. 192–197.

[14] S. Sawitzki, "Embedded Reconfigurable Computing: Architekturen, Anwendungen und Entwurfswerkzeuge (Architectures, Applications, Tools)", Habilitation Thesis, TU Dresden, 2024.

[15] K. A. Bush, "Orthogonal Arrays of Index Unity", *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 426–434, Sep. 1952.

[16] W. C. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[17] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[18] W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes", *IRE Transactions on Information Theory*, vol. 6, no. 4, pp. 459–470, Sep. 1960.

[19] D. E. Gorenstein and N. Zierler, "A Class of Error Correcting Codes in p^m Symbols", *Journal of the Society of Industrial and Applied Mathematics*, vol. 9, no. 2, pp. 207–214, Jun. 1961.

[20] E. R. Berlekamp, *Algebraic Coding Theory*, revised. Aegean Park Press, Jun. 1984.

[21] H.-J. Kang and I.-C. Park, "A High-Speed and Low-Latency Reed-Solomon Decoder based on a Dual-Line Structure", in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Orlando, FL, USA, May 2002, pp. 3180–3183.

[22] G. D. Forney, "On Decoding BCH Codes", *IEEE Transactions on Information Theory*, vol. 11, no. 4, pp. 549–557, Oct. 1965.

[23] F.-K. Chang, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, "Universal Architectures for Reed-Solomon Error-and-Erasure Decoder", in *Asian Solid-State Circuits Conference*, Hsinchu, Taiwan: IEEE, Nov. 2005, pp. 229–232.

[24] A. G. M. Strollo, N. Petra, D. De Caro, and E. Napoli, "An Area-Efficient High-Speed Reed-Solomon Decoder in $0.25\,\mu$m CMOS", in *Proceedings of the IEEE 30th European Solid State Circuits Conference*, Leuven, Belgium: IEEE, Sep. 2004, pp. 479–482.

[25] H.-Y. Hsu and A.-Y. Wu, "VLSI Design of a Reconfigurable Multi-Mode Reed-Solomon Codec for High-Speed Communication Systems", in *IEEE Asia-Pacific Conference on ASIC Proceedings*, Taipei, Taiwan: IEEE, Aug. 2002, pp. 359–362.

[26] J.-C. Huang, C.-M. Wu, M.-D. Shieh, and C.-H. Wu, "An Area-Efficient Versatile Reed-Solomon Decoder for ADSL", in *Proceedings of the 1999 IEEE International Symposium on Circuit and Systems (ISCAS'99)*, vol. 1, Orlando, FL, USA: IEEE, May 1999, pp. 517–520.

[27] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes", in *Proceedings of the IEEE International Conference on Communication*, Geneva, Switzerland: IEEE, May 1993, pp. 1064–1070.

[28] R. G. Gallager, "Low-Density Parity-Check Codes", Sc. D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1963.

[29] G. Krishnaiah, N. Engin, and S. Sawitzki, "Scalable Reconfigurable Channel Decoder Architecture for Future Wireless Handsets", in *Design, Automation and Test in Europe Conference and Exhibition, DATE 2007*, Nice, France: European Design and Automation Association, Apr. 2007, pp. 1563–1568.

[30] Y. Sun and J. R. Cavallaro, "A Flexible LDPC/Turbo Decoder Architecture", *Journal of Signal Processing Systems*, vol. 64, no. 1, pp. 1–16, Jul. 2011.

[31] I. Ahmed and T. Arslan, "A Reconfigurable Viterbi Decoder for a Communication Platform", in *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications (FPL)*, T. Rissa, S. Wilton, and P. Leong, Eds., Tampere, Finland: IEEE, Aug. 2005, pp. 435–440.