

ASIP Decoder Architecture for Convolutional and LDPC Codes

Steffen Kunze, Emil Matuš and Gerhard P. Fettweis
Vodafone Chair Mobile Communications Systems
Technische Universität Dresden
D-01062 Dresden, Germany
Email: steffen.kunze,matus,fettweis@ifn.et.tu-dresden.de

Abstract—In this paper we present a multi-mode decoder architecture for convolutional codes and structured low-density parity-check (LDPC) codes based on a novel computation unit that is able to process Min-Sum as well as Add-Compare-Select (ACS) operations. Realized as application-specific instruction set processor (ASIP), this allows decoding of a vast number of different channel codes and implementation of various communication standards' channel coding schemes with just one single IPcore. Implemented in 130 nm technology, this results in a Viterbi decoding throughput of 30 Mbit/s at 200 MHz for an area of 745 kGates and power consumption of 130 mW.

I. INTRODUCTION

For modern communications devices it is desirable to cover many different communications standards for functionality and compatibility reasons. One part of the signal processing chain that can not be easily ported to flexible software defined radio (SDR) or general purpose processing platforms is the channel decoding part of the outer receiver. Besides being of high computational complexity, there are also many different coding schemes required such as convolutional codes (CC), turbo codes (TC) and LDPC codes with various block lengths and code parameters. In order to support this collection of codes, a flexible multi-mode architecture for channel decoding is necessary. It is very desirable to have a common basic architecture that can be programmed to support current or future standard codes, as a result giving us high design reuse and reduced development cycles.

Several approaches have been reported for reconfigurable decoders that combine decoding of CCs and TCs, e.g. [1], but these lack in flexibility and throughput for future applications. ASIP architectures promise to greatly improve programmability while still allowing high throughput and efficiency with specialised data paths. One example is a programmable decoder [2] that offers a lot of flexibility while maintaining high speed but is still limited to only LDPC codes. Recently however emerged an ASIP architecture that also includes CCs and TCs besides LDPC codes [3] [4] and for which high throughput has been reported. However, the authors state that due to low reuse between CC/TC and LDPC decoding it is beneficial to implement separate data paths for convolutional and LDPC decoding.

In this paper we propose a more integrated approach, where operations of all decoding algorithms are processed in one data path. One functional unit (FU) handles all calculations,

enabling reuse of memories and control logic and limiting overhead. The focus lies on combining the functionality of an already well researched LDPC Decoder with the ability of trellis-based decoding as is necessary for convolutional codes. To achieve this goal we have to take a look at the decoding algorithms and how to map them to hardware in a way that produces maximum overlap.

The rest of the paper is dedicated to algorithm exploration in section II, an architecture description in section III and in sections IV and V follow implementation results and concluding remarks.

II. DECODING ALGORITHMS

We are concerned with two major types of channel codes: Trellis-based codes such as convolutional and Turbo codes and LDPC codes. Let us take a closer look at the decoding algorithms and what kind of operations are necessary to perform them. Decoding of trellis-based codes shall be represented by the well-known Viterbi Algorithm (VA) [5].

A. Viterbi Decoding

The VA is traditionally divided into three parts: branch metric computation (BMC), state metric computation (SMC) and traceback (TB). In the BMC, for every trellis step i the distance $\lambda_{jk}(i) = |y(i) - z_{jk}|^2$ between received input symbols $y(i)$ and expected values z_{jk} is calculated, where z_{jk} is the expected encoder output for a transition from trellis states S_j to S_k . This distance can be calculated as sum of the distance $\lambda_{jk,b}$ between the individual symbol elements, which are quantized as unsigned integer soft values ranging from zero to a maximum integer value y_{max} . $\lambda_{jk,b}$ is essentially the distance of y to either zero or y_{max} , which is equivalent to either the input value itself or its bitwise negations, depending on z_{jk} . The accumulated sum for one input symbol becomes:

$$\lambda_{jk}(i) = \sum_{b=1}^B \lambda_b \quad \text{with} \quad \lambda_b = \begin{cases} y_b(i) & \text{if } z_{jk,b} = 0 \\ \bar{y}_b(i) & \text{if } z_{jk,b} = 1 \end{cases} \quad (1)$$

where B is the number of elements in an input vector (i.e. $B = 2$ for a rate-1/2 code) and \bar{y}_b is the bitwise negation of y_b .

The resulting metrics $\lambda_{jk}(i)$ are used in the SMC to update and select the new path metrics in an Add-Compare-Select

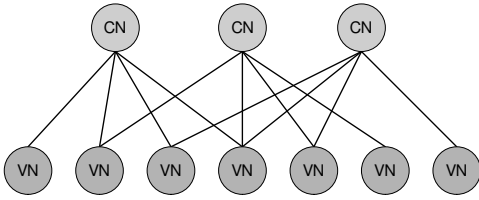


Fig. 1. Tanner graph of an irregular LDPC Code

operation:

$$\Lambda_k(i) = \min(\Lambda_j(i-1) + \lambda_{jk}(i)). \quad (2)$$

The decision $d_k(i)$ from the SMC is stored in path memory and used to trace back the maximum-likelihood path. The traceback algorithm calculates previous states by concatenating the decision bit and the right-shifted current state. Traceback can be shortened to a number of trellis steps equivalent to about five times the constraint length K since all paths converge to the ML path around this time with a high probability. Thus, after an initial delay of $5K$ steps, traceback produces one hard decision output value in every trellis step.

B. LDPC Decoding

An elegant visualization of LDPC decoding is the tanner graph (Fig. 1), where N variable nodes connected to M check nodes represent the parity check equations given by the sparse parity-check matrix. We can use the Offset Min-Sum algorithm [6] for decoding, as it is a cost-efficient but still performant alternative to Gallagers original Belief Propagation algorithm [7]. For Min-Sum decoding, the reliability information from the recieved channel values (represented as log-likelihood ratio (LLR)) is propagated iteratively between the nodes and checked against the parity equations. This works in four basic steps:

- 1) **Initialization:** Graph edges are initialized to the received channel values y .
- 2) **CN update:** The CNs perform a parity check over the connected VNs V and propagate the result together with the corresponding LLRs back to the VNs (excluding intrinsic information from the respective nodes). The update from a CN C_m to a VN V_n can be expressed as:

$$L_{mn} = \left(\prod_{n' \in V_m \setminus n} \text{sign}(L_{n'm}) \right) \cdot \max\left(\min_{n' \in V_m \setminus n} |L_{n'm}| - \beta, 0 \right). \quad (3)$$

β serves as normalization offset to prevent performance degradation, the max function around the min-term and 0 simply indicates β shall not be used on values smaller than itself.

- 3) **VN update:** The VNs collect the LLRs from the connected CNs C and updates them with the new values (again, minus intrinsic information):

$$L_{(nm)} = y_n + \sum_{m' \in C_n \setminus m} L(r_{m'n}) \quad (4)$$

TABLE I
LIST OF OPERATIONS USED BY DECODING ALGORITHMS.

	operation
LDPC CN update	compare, sub, XOR
LDPC VN update	add, sub
VA BMC	add
VA SMC	add, compare
VA TB	shift, mem

- 4) **Decision:** Steps 2 and 3 are repeated iteratively until a stopping criterion is reached. Now the soft outputs of step 3 (this time including intrinsic information) are subjected to hard decision.

Both the VA and the Offset Min-Sum algorithm need to execute many similar, yet independent operations in the different steps of the algorithms, which makes them well suited for reaching high decoding throughputs by implementing them in a parallelized fashion. In the next section we are going to define an architecture of parallel processing units for this purpose.

III. ARCHITECTURAL IMPLICATIONS

A. Processing unit

Now let us take a look at the main operations needed to process the two algorithms discussed. For the VA, we need addition and comparison operations for SMC and BMC and shift operations combined with a path memory for TB. The Offset Min-Sum algorithm requires us to perform comparison (min search), subtraction (normalization) and bitwise XOR (parity check) in the CN update step. In the VN update, there is addition and subtraction. An overview of these relationships is given in table I. As we can clearly see, this collection is dominated by the fairly similar addition, subtraction and comparison operations.

These operations are shared between the two algorithms and should be used to build a basic processing unit that is able to execute both algorithms. Including the XOR operation, which is only needed for LDPC decoding, does not introduce a significant increase in complexity. The TB on the other hand works radically different than the other parts of decoding, since a large amount of path memory accesses is necessary for every decoding step. It is more feasible to implement this in a separate FU.

Figure 2 shows a block schematic of a FU that realizes the above specifications. This ALU unit processes the decoding algorithms serially, one input operand per clock cycle. Processing is separated into two stages that can be connected via a programmable interconnect. For LDPC decoding, the min-term and sign computation (as XOR) from equation 3 are realized within the 'Min Search' block in stage one. The cancellation of intrinsic information coming from the currently processed edge V_n is done in stage two. The edge has been buffered in

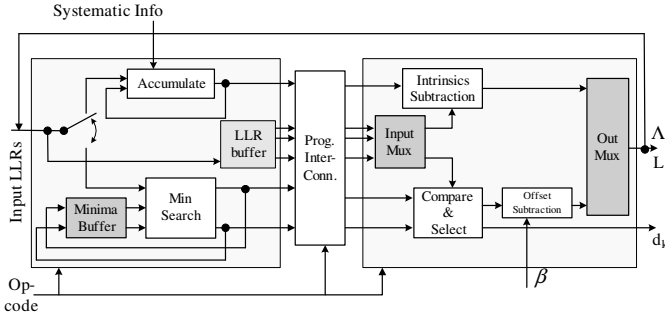


Fig. 2. Two-stage FU capable of sequentially processing operations needed for both Viterbi and LDPC decoding

the LLR buffer and is now compared to the minimum value in 'Compare & Select' and exchanged for the second smallest value, if necessary. Finally, the offset β is subtracted. The VN update LLR accumulation (eq. 4) is performed in the first ALU stage and intrinsic information removed in stage two, again using the LLR buffer. The VA is processed as follows: First, the sum and negation from equation 1 are executed in stage one box 'Accumulate', taking into account the systematic information about the expected input. The ACS addition (eq. 2) is also executed here, then the intermediate result is buffered. Afterwards the same calculation is done for the second trellis branch that is connected to the current node. Then the two candidates are compared in stage two. Output from the ALU is the updated metric, which can be fed back as input for the next trellis step, and a decision bit, which indicates the branch selection of the current Trellis step to the TB unit (TBU).

B. Routing network

The other important part of the algorithms besides the actual calculations is the connectivity of the Tanner and Trellis graphs, which also has to be represented through the architecture. To improve performance, usually many processing elements are used in parallel and the data has to be routed between them via interconnections. From [2] we can see that for a parallel structure of FUs the Tanner Graph connections can be easily realized with a barrel shifter. For the VA, the interconnect structure changes only with the number of trellis states and is static otherwise (assuming the usual standard feedforward encoders). Therefore it should be sufficient to use an interconnect structure that can be switched between certain different configurations depending on the number of states. Figure 3 shows the p_t ALU stages in parallel with such an interconnect in between and the barrel shifters (Shuffle units) on the outside.

C. TBU

The traceback algorithm requires a large number of memory accesses with very little computation. Therefore, it is impractical to include it in the ALU unit. Instead, processing it in parallel in a separate FU can be used to speed up the decoding process. The TBU's path memory needs to store the last $5K$ decisions for the 2^{K-1} trellis states.

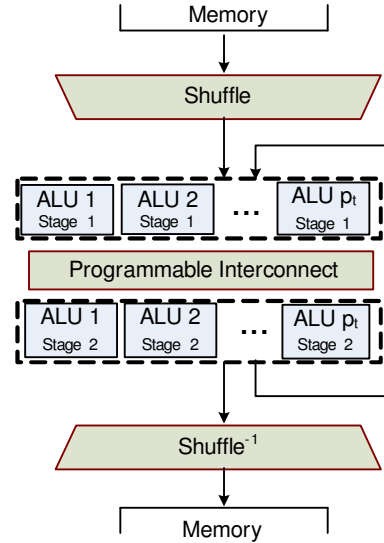


Fig. 3. Array of parallel ALU FUs with interconnects

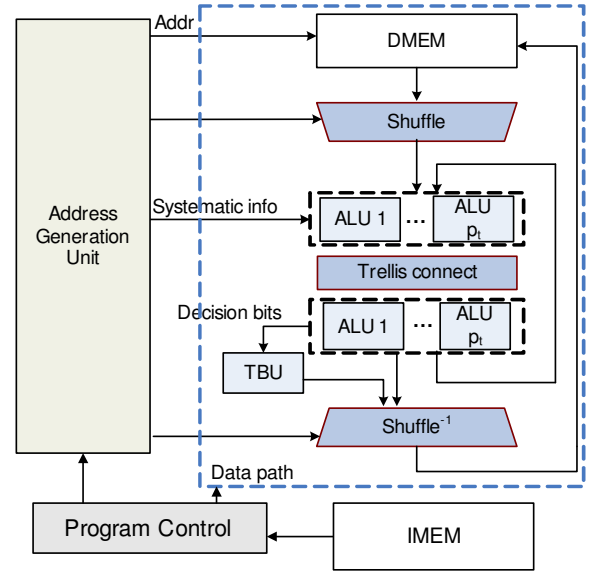


Fig. 4. Architecture overview

D. ASIP Decoder Architecture

Now we apply the proposed concepts to an architecture based on the SIMD (single instruction, multiple data) principle. Figure 4 shows a Decoder ASIP based around a parallel data path. Additionally, it consists of an address generation and control logic block (AGU), data and instruction memories and program control logic. All functional units are programmable via Very Long Instruction Words (VLIW), which are stored in IMEM.

IV. RESULTS

A decoder with 64-fold parallelism has been implemented using the STA low-power design paradigm [8]. The ALU has been configured to allow processing of equations with up to

16 CN or VN values (LLR buffer size = 16) at a quantization of 5 bit. The programmable interconnect supports 8-, 16-, 32- and 64-state trellis connections between ALUs. Codes with larger trellises can still be decoded by buffering state metrics in memory, although this impacts throughput, of course. The data memory has a size of 40kByte, enabling block lengths of 16K bits for LDPC and 64k bits for Viterbi decoding. The decoder was implemented in Verilog and synthesized with SYNOPSIS tools for a 130 nm CMOS process at a target clock frequency of 200 MHz. The area and power consumption can be found in Table II. As we can see, the design is dominated by memory. The only FU that is dedicated to just one algorithm, the TBU, takes up 8.6% of the decoder, giving us a reuse factor of over 91%. The additional cost of implementing the VA in the ALU compared to an older version that supports only LDPC codes is about 18 %.

TABLE II

AREA ESTIMATE FOR THE DECODER COMPONENTS AND POWER ESTIMATE FOR VITERBI DECODING.

Module	Area [Gates]	%
Top Level	745 K	100
Memories	461 K	62
Functional Units	284 K	38
- Vector ALUs (1 ALU FU)	136 K (2.1 K)	18 (0.3)
- Address & Control Logic	33 K	4.4
- TBU	64 K	8.6
- Interconnects	35 K	4.7
- Interfaces	8 K	1.1
	Power [mW]	%
Top Level	129	100
Memories	26	20
Functional Units	103	80

To measure the performance, simulations were carried out with an instruction set simulator and on RTL level with ModelSim. For Viterbi decoding a rate-1/2 (171,133) convolutional code with a block length of 1024 bit was used. A throughput of 30 Mbit/s with an SNR of 4.25dB at $BER = 10^{-4}$ was reached. The corresponding power consumption of 130 mW was calculated with SYNOPSIS PrimeTime. The LDPC results were simulated using a tailbiting LDPC convolutional code of rate $R = 0.4$, block length $N_B = 3200$ bit at 10 decoding iterations, reaching 53 Mbit/s throughput. A comparison to published state-of-the-art is presented in table III. Throughput values from [4] have been normalized in respect to our results to make comparing easier and area scaled-down to 65 nm has been estimated.

TABLE III

COMPARISON TO OTHER PUBLISHED WORK

	this paper	[4] -normalized
- Area [mm ²]	0.95 ¹	0.62
- Throughput VA [MBit/s] ²	30	22
- Throughput LDPC [MBit/s] ³	53	62
- Power [mW]	130	n/a
- Logic Reuse [%]	91	63

While LDPC decoding does not quite match the speeds from [4], Viterbi decoding speed for a 64-state CC surpasses their results (numbers originally published in [3]), but at the cost of a larger area. There are no power consumption values available to compare against and the logic reuse in [4] is lower since only the memories are shared and different data paths are used for LDPC and Viterbi decoding.

V. CONCLUSION

In this paper we presented an ASIP for channel decoding that exhibits state-of-the-art performance and is the first to integrate functionality for LDPC and convolutional decoding in one processing unit. The high flexibility this architecture provides is a prerequisite for modern outer receivers to enable easier adaption of new communication standards, higher design reuse and faster application development. The decoder reaches a throughput of 30 MBit/s for Viterbi decoding at $f = 200$ MHz with an area of 745 kGates and power consumption of 130 mW. Future research could investigate the potential to extend this architecture to include Turbo decoding.

ACKNOWLEDGMENT

This research is part of the MxMobile project supported by Bundesministerium für Bildung und Forschung (BMBF).

REFERENCES

- [1] M. A. Bickerstaff, D. Garrett, C. Thomas, T. Prokop, B. Widdup, G. Zhou, C. Nichol, and R. Yan, "A unified turbo/viterbi channel decoder for 3GPP mobile wireless in 0.18 um CMOS," in *Proc. IEEE Int. Solid-State Circuits Conference (ISSCC'02)*, San Francisco, CA, Feb. 2002, pp. 124–451.
- [2] M. Bimberg, M. Tavares, E. Matus, and G. Fettweis, "A high-throughput programmable decoder for LDPC convolutional codes," in *Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Montreal, Canada, July 2007.
- [3] T. Vogt and N. Wehn, "A reconfigurable application specific instruction set processor for convolutional and turbo decoding in a sdr environment," in *Proc. Design, Automation and Test in Europe (DATE'08)*, Munich, Germany, Mar. 2008.
- [4] M. Alles, T. Vogt, and N. Wehn, "Flexichap: A reconfigurable ASIP for convolutional, turbo, and LDPC code decoding," in *Proc. Int. Symposium on Turbo Coding (TURBO CODING'08)*, Lausanne, Switzerland, Sept. 2008.
- [5] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm." *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, Apr. 1967.
- [6] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.
- [7] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
- [8] G. Cichon, P. Robelly, H. Seidel, E. Matus, M. Bronzel, and G. Fettweis, "Synchronous transfer architecture STA," in *Proc. Int. Symposium on Systems, Architectures, Modeling and Simulation (SAMOS)*, Samos, Greece, June 2004, pp. 126–130.

¹scaled to 65 nm assuming a scaling factor of 4

²normalized to $f = 200$ MHz

³normalized to $f = 200$ MHz and $R = 0.4$