Multi Core Implementation of a Trellis Based Syndrome Decoder with Adaptive Complexity

Klaus Hueske, Jan Geldmacher and Jürgen Götze

Information Processing Lab
Department of Electrical Engineering and Information Technology, TU Dortmund University
Otto-Hahn-Str. 4, 44227 Dortmund, Germany
klaus.hueske@tu-dortmund.de

***Abstract*—This paper investigates the implementation of a trellis based syndrome decoder on symmetric multiprocessor (SMP) platforms. Two advantages of the proposed approach will be exposed: First, compared to conventional parallel Viterbi decoder implementations, the syndrome decoder achieves a higher parallel efficiency in terms of speedup on the SMP platform. This is realized by reducing the computational overhead of the parallel Viterbi algorithm implementation. Second, it offers an adaptive complexity, i.e. the number of decoding operations decreases with improving transmission conditions. This property can be exploited to reduce the average energy consumption of a radio receiver. Measurement results are shown for two SMP platforms: ARM's MPCore and Intel's Core i7.**

I. Introduction

The paradigm of energy efficiency of digital baseband algorithms is no longer limited to mobile devices where the battery capacity is a constraining factor. Due to increased energy costs and growing awareness of users it has become a major topic also for appliances that do not have such constraints, like consumer electronics or communication infrastructure (often referred to as ”Green IT”). The energy consumption of a baseband modem is significantly influenced by the complexity of the used algorithms itself and the way they are implemented on a specific signal processing platform.

To reduce the average computational complexity (and with it the energy consumption of a receiver), the algorithms should be able to adapt to the current transmission conditions, i.e. utilize all available processing power in worst case conditions, but adaptively reduce the complexity for more favorable transmission conditions.

An investigation regarding the complexity of typical baseband processing tasks has revealed that the forward error correction (FEC) algorithms are amongst the ones with highest computational complexity [1]. Common algorithms used in FEC are the Viterbi Algorithm (VA) for decoding convolutional codes and the BCJR algorithm for Turbo decoding [2]. Both algorithms are based on the trellis representation of the respective encoder. As their structures are related, we will focus on the VA for decoding convolutional codes. However, the results can be extended to BCJR decoding.

To achieve a sufficient decoding throughput, FEC decoders are often realized as parallel implementations, either as dedicated hardware or as some kind of processor. In our work we assume a symmetric multi processor platform (SMP) to measure the efficiency of the parallel implementation. Measurement results are provided for two different platforms: ARM's MPCore embedded multi core processor [3] and Intel's Core i7. The efficiency of a parallel implementation will be measured in terms of speedup, i.e. by which factor the processing time is reduced by using multiple processing units.

A common approach for Viterbi decoding on a parallel platform is to separate the received sequence into overlapping blocks and to decode these blocks independently. This general principle has been called overlap-add Viterbi algorithm or sliding block Viterbi decoder [4].

Two drawbacks can be identified regarding the overlapping VA: First, an overhead is introduced by decoding overlapping parts of the received sequence. This overhead will limit the efficiency of a parallel implementation. Second, the whole received sequence has to be processed, even if few or no errors occurred during transmission. The decoding complexity remains constant, i.e. it cannot be easily adapted to changing transmission conditions.

These drawbacks are addressed in an alternative decoder concept based on syndrome decoding (SD) [5]: With SD error-free parts of the received sequence can be identified in advance and the VA is only applied to erroneous parts of the sequence. This significantly reduces the number of decoding operations for good transmission conditions, i.e. high SNR. Given a constant throughput the reduced number of decoding operations can be exploited by dynamic voltage/frequency scaling (DVFS) methods [6] to reduce the energy consumption of the receiver. When applying DVFS the processor clock is lowered and the voltage is adjusted in a suitable manner to reduce the power consumption of the device.

Parallel decoding benefits from SD as well, because it allows to partition the received sequence without any overlapping overhead, which clearly improves the efficiency of the parallel implementation. While alternative approaches like the T-Algorithm [7] reduce the decoding complexity in high SNR regions as well, the avoidance of overlapping relies on a specific property of the syndrome decoder.

Two aspects will be evaluated in this paper: The efficiency of the SMP based syndrome decoder implementation in terms of speedup and the possible reduction of clock frequency exploiting adaptive complexity.

The paper is organized as follows: In Section II, the main principles of overlapping Viterbi decoding are reviewed. The trellis based syndrome decoder will be presented in Section III. In Section IV we will focus on the implementation details and the processor platforms that are used for performance evaluation in terms of speedup/clock frequency reduction in Section V. Final conclusions are drawn in Section VI.

II. Viterbi Decoding

A. Decoding Problem

Assume an information sequence $\mathbf{u}$ is encoded with a generator matrix $\mathbf{G}$ and transmitted over a channel. With $\mathbf{e}$ the channel error, the received sequence can be expressed as

$$\mathbf{r} = \mathbf{uG} \oplus \mathbf{e} = \mathbf{v} \oplus \mathbf{e}, \tag{1}$$

where $\mathbf{v}$ is the encoded sequence and $\oplus$ the *xor*-operation. The decoding problem can then be written in terms of an optimization problem

$$\mathbf{v}^* = \arg\min_{\hat{\mathbf{v}}} \|\mathbf{r} \oplus \hat{\mathbf{v}}\|, \tag{2}$$

i.e. to find an estimate of the code sequence $\mathbf{v}^*$ with minimum distance to $\mathbf{r}$. In Viterbi decoding this optimization problem is solved by applying the VA to search the trellis of the encoder $\mathbf{G}$ for the code sequence $\mathbf{v}^* = \mathbf{u}^*\mathbf{G}$ with minimum distance to the received sequence $\mathbf{r}$.

B. Overlapping Viterbi Decoder

If parallel decoding is desired, a possible approach is to separate the received sequence $\mathbf{r}$ into blocks and distribute these block to parallel processors. However, the encoder introduces continuous dependencies between successive code symbols, i.e. realizing a partitioning is not straight-forward.

To make the blocks independent from each other and to achieve optimal decoding performance in maximum likelihood sense, the blocks have to overlap to a certain extend [4]. The required minimum overlapping length is governed by two parts: The acquisition depth A and the truncation depth D.

The acquisition depth is the number of decoding steps that are necessary to make the metric independent of the unknown initial metrics. This is likely the case after $A = 5\nu$ steps [8], where ν is the constraint length of the code. The traceback depth $D = 5\nu$ is the number of steps after that all paths have most likely merged when performing traceback [2].

Depending on the length of the decoded block N the additionally required $A + D = 10\nu$ received symbols cause a considerable computational overhead, which can significantly reduce the efficiency of a parallel implementation. In an optimum parallel implementation $S_{\text{Par}}(P) = t_1/t_P = P$ holds, i.e. the processing time t_1 using one processor is P times the processing time t_P using P processors. However, using the overlapping VA with a frame of length B that is decoded using P overlapping blocks of size $N = B/P$ with P parallel processors, the maximum speedup is limited by

$$S_{\text{Par,Max}} = \frac{t_1}{t_P} = \frac{B}{\frac{B}{P} + \frac{(P-1)(A+D)}{P}} = \frac{P}{1 + \frac{(P-1)10\nu}{B}}. \tag{3}$$

Here it is assumed that the frames are terminated, i.e. $A = 0$ for the first block and $D = 0$ for the last block. Two drawbacks can be identified regarding the overlapping VA: First, the overhead limits the achievable speedup and causes additional computational load. Second, the whole received sequence has to be processed, even if few or no errors occurred during transmission. One approach to address these drawbacks is based on syndrome decoding.

III. Syndrome Decoding

A. Basic Principles

The adaptive complexity decoder is based on Schalkwijk's syndrome decoder presented in [5]. Instead of searching for $\mathbf{v}^*$ directly, the syndrome decoder searches for a sequence $\mathbf{e}^*$, which corrects the errors in $\mathbf{r}$. The according constrained optimization problem can be stated as

$$\mathbf{e}^* = \arg\min_{\hat{\mathbf{e}}} \left\{ \|\hat{\mathbf{e}}\| \mid (\mathbf{r} \oplus \hat{\mathbf{e}})\mathbf{H}^T = \mathbf{0} \right\}, \tag{4}$$

where $\mathbf{H}^T$ is the syndrome former matrix. The constraint makes sure that the estimated error sequence $\mathbf{e}^*$ results in a valid code sequence when applied to $\mathbf{r}$. For that the syndrome former $\mathbf{H}^T$ is defined to be orthogonal to the code's generator matrix, i.e. $\mathbf{GH}^T = \mathbf{0}$, and thus to all code sequences. With $\mathbf{rH}^T = \mathbf{b}$ the syndrome of $\mathbf{r}$, the constraint can also be formulated as $\mathbf{b} = \hat{\mathbf{e}}\mathbf{H}^T$, i.e. the estimated error $\mathbf{e}^*$ has to cause the same syndrome as $\mathbf{r}$. Furthermore, it holds that

$$\mathbf{b} = \mathbf{rH}^T = (\mathbf{v} \oplus \mathbf{e})\mathbf{H}^T = \mathbf{uGH}^T \oplus \mathbf{eH}^T = \mathbf{eH}^T$$

and it is obvious that the syndrome only depends on the error sequence. The equivalence to (2) can be easily verified by inserting $\hat{\mathbf{e}} = \mathbf{r} \oplus \hat{\mathbf{v}}$ into (4).

The optimization problem (4) is solved by applying the VA to search the trellis of the syndrome former $\mathbf{H}^T$ for the error sequence $\mathbf{e}^*$, which satisfies $\mathbf{e}^*\mathbf{H}^T = \mathbf{b}$ and has minimum weight[1]. The corresponding information sequence is then obtained by multiplication by the right inverse $\mathbf{G}^{-1}$ of the generator matrix, $\mathbf{u}^* = (\mathbf{r} \oplus \mathbf{e}^*)\mathbf{G}^{-1}$.

We summarize the following important properties of the syndrome decoder:

First, as mentioned above the syndrome only depends on the transmission errors. Error-free parts in $\mathbf{r}$ will lead to zero sequences in $\mathbf{b}$. Thus, error-free periods in $\mathbf{r}$ can be estimated by detecting parts of consecutive zeros with sufficient length in $\mathbf{b}$. This allows the detection of error-free periods prior to decoding the sequence, i.e. only erroneous parts of $\mathbf{r}$ have to be processed by the VA. This enables an adaptive reduction of decoding complexity for good transmission conditions, i.e. high SNR [11].

Second, the additional operations for syndrome computing and application of $\mathbf{G}^{-1}$ are simple XOR-operations and thus of negligible complexity compared to the VA. Consequently,

[1]While the original paper [5] only considered hard-decision decoding, the modification of the decoder metric allows also soft-decision decoding. Furthermore, the syndrome decoding approach can be extended to enable MAP decoding [9] [10].

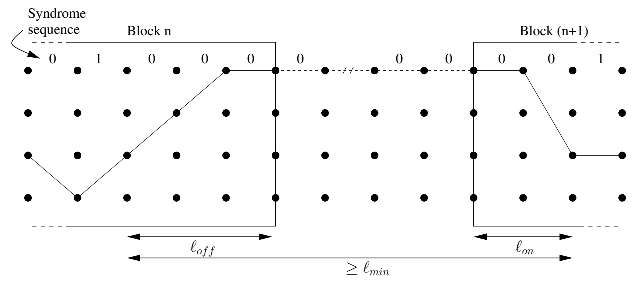

Fig. 1. Design parameters of the block processing scheme.

the worst case complexity of the SD is basically identical to the complexity of the conventional Viterbi decoder.

Third, if we assume that all transmitted information sequences $\mathbf{u}$ have equal probability, then all code sequences $\mathbf{v}$, all paths through the encoder trellis and consequently all trellis states will have equal probability. The SD is based on the idea of estimating error sequences instead of code sequences. This leads to decoding with unbalanced state probabilities because the state probabilities now depend on transmission errors and no longer on the information sequence. In error-free parts the decoding path traverses the "zero-state" state with high probability. This state has all zero register contents and a leaving edge with input/output bits all zero.

The mentioned properties are exploited in the Block Syndrome Decoding (BSD) approach proposed in [11], which is described in the following section.

B. Block Syndrome Decoder

The principle of the BSD for parallel decoding can be outlined as follows: First the syndrome sequence is analyzed and parts with a sufficient number of consecutive zeros are identified ("all-zero parts"). Using this information, the received sequence is separated into blocks with initial and final state being the zero-state, i.e. no overlapping is required. The resulting blocks can then be distributed to the parallel decoders.

A critical part of the BSD is how to decide from the syndrome sequence at which points a separation of the received sequence is possible. Therefore, the following design parameters are defined: The parameter ℓ_{min} is defined to be the minimum number of consecutive *0*s in the syndrome sequence required for a separation. That is, whenever a sequence of consecutive *0*s with length greater or equal to ℓ_{min} is detected, the received sequence is separated at this point. The parameters ℓ_{off} and ℓ_{on} define where exactly the previous block ends and where the next block starts: ℓ_{off} denotes the number of *0*s in the syndrome sequence at the end of a block, i.e. the number of stages until it can be assumed with sufficient probability that the ML path has returned to the zero-state. The third parameter ℓ_{on} is defined to be the number of *0*s in the syndrome sequence from the beginning of a block to the first *1*. Figure 1 illustrates the meaning of these parameters for a four state trellis, where values are chosen as $\ell_{off} = 3$ and $\ell_{on} = 2$.

The selection of the parameter set has an impact on the performance in terms of bit error rate (BER) and parallelization efficiency: Small values for ℓ_{on}, ℓ_{off} or ℓ_{min} will degrade the BER. Large values for ℓ_{min} reduce the amount of possible separation points, resulting in longer blocks. Large values for ℓ_{on} and ℓ_{off} reduce the savings, that result from avoiding the decoding of error-free parts between two consecutive blocks. Suitable parameter sets depend on the used code and can be determined from simulation results. Results for different code rates and constraint lengths can be found in [12] and [11], respectively.

Compared to the overlapping VA described in Section II-B, the block length in the BSD is not fixed, but depends on the channel error: Good transmission conditions will allow separations more frequently and thus result in shorter block lengths, while severe transmission conditions (more errors) will result in longer blocks. Hence, for implementation the block length has to be limited by a suitable upper bound, which can be easily achieved by implementing overlapping as a fallback. This extension of the BSD is referred to as BSD/OL in the following [11].

IV. Multi Core Implementation

A. Processor Platforms

For evaluation the presented approaches were implemented as "C" program and the performance was determined on an ARM MPCore and on an Intel Core i7 920 SMP platform. As the focus of this work is on the evaluation of the parallel efficiency of the proposed algorithm in terms of relative speedup, the used code is not optimized to achieve high absolute throughputs on a specific platform. However, a throughput of 3.2 Mbit/s could be reached using one of the Core i7 cores.

The MPCore architecture is based on 32-bit integer RISC ARM 11 cores with a clock frequency of 200 MHz. The evaluation platform consists of 4 identical cores. Each core can use 32 kB of dedicated level-1 cache for data and 32 kB for instructions. A shared second level cache of 1 MB, which runs at core frequency, allows fast data exchange between the processors. For both scenarios a Linux based operating system with SMP kernel is used for resource management and scheduling. Both architectures provide DVFS techniques, named *Intelligent Energy Manager (IEM)* [3] in case of ARM or *Speedstep* in case of Intel.

B. Implementation

The parallel implementation is based on the *pthreads* library [13]. Multiple decoders are created as P different threads. In case of overlapping Viterbi decoding overlapping parts of the received sequence are assigned to each decoding thread. Each time a thread finishes decoding, a new overlapping part is assigned. After decoding, the inner parts of each decoded block are extracted and reassembled to obtain the final decoded output sequence. The received frames of length B are divided into P parts and each part is decoded as an independent thread. The traceback is realized within the thread.

In case of the BSD the syndrome analysis has to be performed before starting the actual decoding process. Starting from the beginning of the syndrome sequence, a counter is incremented for each detected zero in the syndrome sequence and reset for each detected one. If ℓ_{min} consecutive zeros are detected, the thread will start decoding the detected block. If the maximum allowed block length is reached without finding ℓ_{min} consecutive zeros, the thread will decode the block using the overlapping method (BSD/OL fallback) [11]. In this case a flag is set to prevent the next active thread from starting in zero state. A pointer is used to identify the last decoded element of the received sequence. To avoid indeterministic behavior, the access to the pointers and flags are protected by *mutexes*, i.e. only one thread can access these resources at a given time. It should be emphasized here, that only the computationally simple syndrome analysis has to be performed sequentially, while the decoding process itself can be performed concurrently.

V. Performance Evaluation

The multicore performance of the presented approaches in terms of S_{Par} is evaluated using the implementation from Section IV-B.

The simulation parameters are set as follows: The convolutional code has a rate of $R = 1/2$ and $\nu = 6$ registers, which corresponds to 64 states. The frame length is set to $B = 1632$ received symbols. These parameters were chosen according to the DVB-T standard [14]. The BSD parameters are $\ell_{min} = 18$, $\ell_{on} = 6$ and $\ell_{off} = 6$. The received sequence was partitioned into P blocks, where P is the number of threads.

A. Bit Error Rate and Additional Computations

As stated in Section III-B the BSD parameters have to be chosen as a trade-off between reduction of decoding operations and BER performance. Figure 2 shows that the BER is basically identical to that of the conventional Viterbi decoder over a wide range of SNR (both decoders using soft bit inputs).

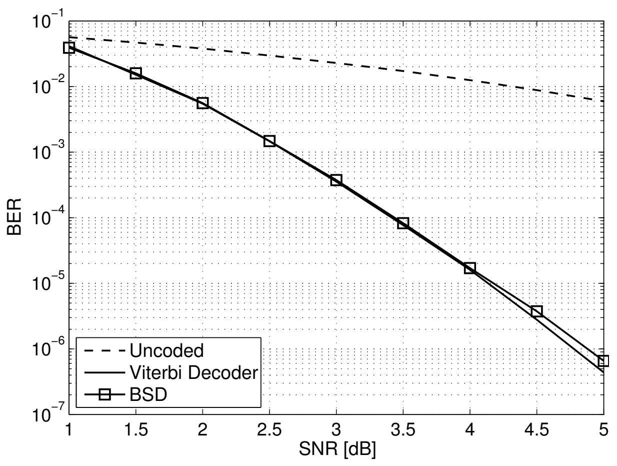

Fig. 2. BER over SNR for conventional Viterbi decoder and BSD.

As discussed in Section III, the BSD requires additional computations for syndrome calculation and application of the inverse generator matrix. The computational effort can, however, be considered as insignificant compared to the VA: From the total runtime of the decoder, only 1.48% are consumed for syndrome calculation and another 1.30% for the application of the inverse generator matrix.

B. Parallel Efficiency

In Figure 3 the parallel speedups S_{Par} for the two decoder concepts running on the ARM MPCore and Core i7 platforms are depicted.

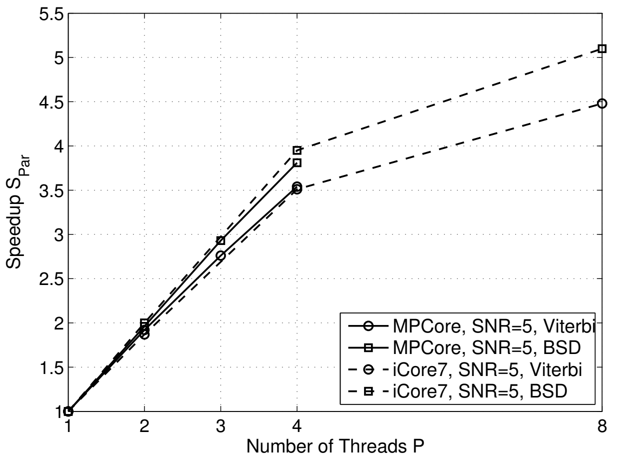

Fig. 3. Speedup by parallel implementation of BSD and Viterbi over P.

With two threads both algorithms perform almost perfectly and reach a speedup close to two. While the BSD scales almost linearly for an increasing number of threads, the gap between overlapping VA and BSD increases. This is caused by the additional overhead due to overlapping. From (3) the maximum speedup for the VA running on 4 threads can be computed as 3.60, which conforms to the measurement results.

The results for 8 threads running on the Core i7 are also shown. Although the Core i7 has 8 logical cores, these cores are based on 4 physical cores using hyperthreading technology. Consequently the increase of speedup is not linear anymore.

The ability of the BSD to separate the received sequence into blocks with known initial and final states strongly depends on the number of transmission errors. This has an impact on the parallel efficiency as shown in Figure 4. Note that only the parallel speedup is considered here. The savings due to adaptive complexity are not yet included and will be discussed in Section V-C.

For low SNR only few and long blocks can be detected, which forces the BSD into the overlapping fallback mode (BSD/OL). In this case the speedup is identical to that of the overlapping Viterbi decoder. For higher SNR more suitable blocks can be found and the speedup approaches the theoretical limit. For very high SNR large parts of the received sequence are already error-free, i.e. only few small blocks have to be decoded. As the decoding complexity is already very low in this case, the relative amount of time spent for analyzing the syndrome becomes more significant compared to the time used for the actual decoding. This decreases the parallel speedup, as the syndrome analysis is a sequential process that cannot benefit from the parallel implementation (cf. Section IV-B).

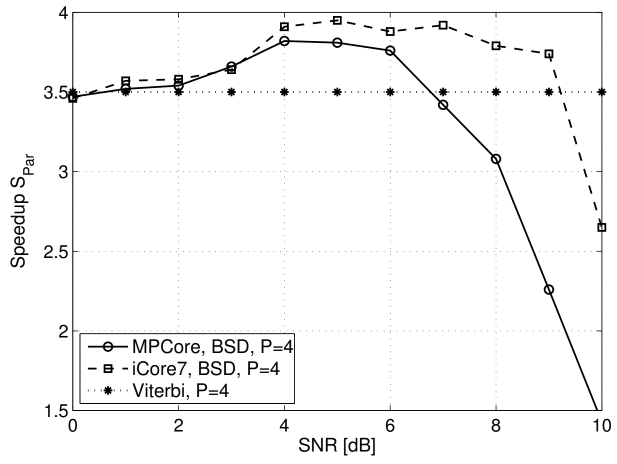

Fig. 4. Speedup by parallel implementation of BSD and Viterbi over SNR.

This decrease, however, is insignificant, because the savings in decoding operations by the adaptive complexity of the BSD clearly outweigh the reduced parallel efficiency, as shown in the following section.

C. Adaptive Complexity

The reduction of clock frequency to exploit the saved decoding operations in a scenario with constant throughput can be described by a factor $S_{\text{Dyn}} = f_{\text{Vit}}/f_{N\,dB}$. Here f_{Vit} denotes the clock frequency of the standard Viterbi decoder and $f_{N\,dB}$ the clock frequency of the BSD decoder at an SNR of NdB. To simplify the measurements we determined the quotient of the execution time t_{Vit} of the conventional Viterbi decoder and the execution time $t_{N\,dB}$ of the BSD at an SNR of NdB instead of modifying the processor's clock frequency. As both methods are equivalent, this yields

$$S_{\text{Dyn}}(N) = \frac{t_{\text{Vit}}}{t_{N\,dB}} = \frac{f_{\text{Vit}}}{f_{N\,dB}} \tag{5}$$

The results are shown in Figure 5. For increasing SNR the

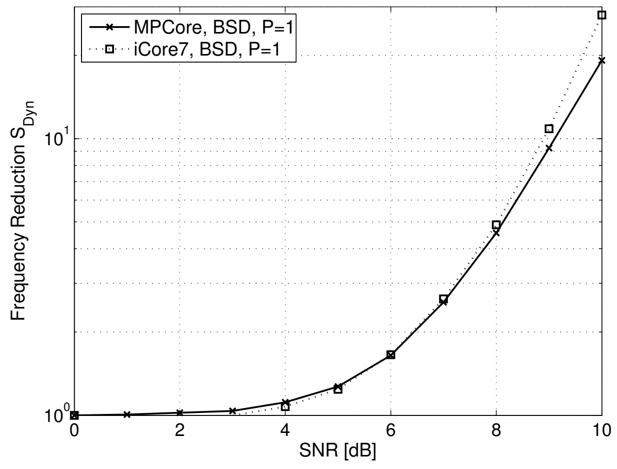

Fig. 5. Possible reduction of clock frequency using BSD.

savings in decoding operations result in a significant possible clock frequency reduction compared to the conventional VA, whose complexity is constant for all SNR. This can result in considerable energy savings in higher-than-average transmission conditions when applying DVFS methods. As mentioned before, the efficiency of the parallel BSD implementation decreases for very high SNR. However, from the measurement results it is obvious that the savings in decoding time clearly outweigh this effect. Note that, instead of frequency reduction, a successive deactivation of processor cores can be performed to exploit the adaptive complexity and to reduce power consumption.

VI. Conclusions

The predicted parallel performance of the trellis based syndrome decoding algorithm was verified by measurement results obtained from current symmetric multiprocessor implementations. The block syndrome decoder shows two beneficial properties in this context: Due to its avoidance of overlapping overhead it can achieve a higher parallel efficiency in terms of speedup. Furthermore, due to its adaptive complexity it allows the reduction of the receiver's energy consumption under above-average transmission conditions. It should be mentioned here that the savings by reduction of decoding operations can be further increased by choosing smaller values for ℓ_{min}, e.g. if only a specific BER is required as in [12] or a code with shorter constraint length is chosen [11].

References

[1] K. Hueske, J. Geldmacher, J. Götze, and E. Coersmeier, "Multi Core Processing for Software Radio Channel Decoder," in *5th Karlsruhe Workshop on Software Radios (WSR'08)*, Karlsruhe, Germany, 2008.

[2] S. Lin and D. J. Costello Jr., *Error Control Coding, 2. edition*. Pearson Prentice Hall, 2004.

[3] ARM, *ARM11 MPCore Processor Technical Reference Manual (ddi0360f) - http://infocenter.arm.com*.

[4] P. Black and T.-Y. Meng, "A 1-Gb/s, four-state, sliding block viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 6, pp. 797–805, June 1997.

[5] J. Schalkwijk and A. Vinck, "Syndrome decoding of binary rate-1/2 convolutional codes," *IEEE Transactions on Communications*, vol. 24, no. 9, pp. 977–985, Sept. 1976.

[6] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 2007, p. 43.

[7] S. Simmons, "Breadth-first trellis decoding with adaptive effort," *IEEE Transactions on Communications*, vol. 38, no. 1, pp. 3–12, 1990.

[8] G. Fettweis and H. Meyr, "Feedforward architectures for parallel viterbi decoding," *The Journal of VLSI Signal Processing*, vol. 3, no. 1-2, pp. 105–119, June 1991.

[9] M. Tajima, K. Shibata, and Z. Kawasaki, "Relation between encoder and syndrome former variables and symbol reliability estimation using a syndrome trellis," *IEEE Transactions on Communications*, vol. 51, no. 9, pp. 1474–1484, Sept. 2003.

[10] J. Geldmacher, K. Hueske, S. Bialas, and J. Götze, "Adaptive Low Complexity MAP Decoding for Turbo Equalization," in *6th International Symposium on Turbo Codes & Iterative Information Processing*, 2010.

[11] J. Geldmacher, K. Hueske, and J. Götze, "Syndrome based block decoding of convolutional codes," in *IEEE International Symposium on Wireless Communication Systems (ISWCS'08)*, 2008, pp. 542–546.

[12] ——, "An adaptive and complexity reduced decoding algorithm for convolutional codes and its application to digital broadcasting systems," in *International Conference on Ultra Modern Communications (ICUMT2009)*, 2009.

[13] B. Nichols, D. Buttlar, and J. P. Farell, *Pthreads Programming. A POSIX Standard for Better Multiprocessing*. O'Reilly Media, 1996.

[14] "DVB; Framing structure, channel coding and modulation for digital terrestrial television (ETSI EN 300 744)," 2004.