A high throughput Turbo Decoder for an OFDM-based WLAN demonstrator

Jochen Ertel[1], Jörg Vogt[2], Adolf Finger[3]

[1] Communications Laboratory, Dresden University of Technology, Germany, ertel@ifn.et.tu-dresden.de
[2] Communications Laboratory, Dresden University of Technology, Germany, vogt@ifn.et.tu-dresden.de
[3] Communications Laboratory, Dresden University of Technology, Germany, finger@ifn.et.tu-dresden.de

Abstract

The considerable performance gain of turbo codes compared to other conventional coding schemes makes it to a primary candidate for OFDM-based wireless LANs. This paper shows the implementation of a turbo decoder for a flexible high bit rate modem architecture. Focus is on optimization of the Max-Log-MAP decoding algorithm with respect to hardware realization. Furthermore, parallelization concepts are discussed to allow high throughput while keeping flexibility. Finally implementation results based on a FPGA solution are presented.

1 Introduction

THE constantly increasing demand for high speed and high quality mobile data transmission pushes research and development towards wireless local area network technologies incorporating adaptive and reconfigurable modem architectures. In line with this tendency, the objective of the IST project WIND-FLEX [1] is the development of a flexible high bit rate modem architecture for a wireless indoor environment at 17 GHz allowing slow mobility.

As channel coding scheme turbo coding [2] was selected due to its outstanding error correction capability. On the other hand advanced technologies allow its relative complex implementation with feasible effort. In this paper we present turbo decoder implementation aspects aiming at a FPGA solution. At first algorithms are researched to minimize hardware consumption while keeping performance constant. This includes optimization of integer arithmetic ranges and quantization aspects. Moreover, parallelization concepts are investigated as well as methods to minimize the critical path (maximize clock frequency) aiming at high throughput decoding.

2 System Overview

The WIND-FLEX modem employs OFDM with 128 subcarriers in a 50 MHz channel at 17 GHz based on TDMA/TDD as radio access scheme. The coverage of the modem ranges from 10 m for non line of sight (NLOS) up to 100 m for line of sight (LOS) radio links. Considering a 25 % overhead for signalling and control information, a maximum payload of around 100 Mbps can be achieved for a point-to-point communication.

This work is partly funded by the European Commission.

The general modem architecture has been designed to cope with environmental changes and different user requests at run time while ensuring an overall operational efficiency. The latter can be interpreted as a compromise over several requirements like data rate, bit error rate, power consumption and others. For that purpose, the so called supervisor was additionally introduced which takes the dynamic adaption of the transmit and receive parameters into account. Figure 1 shows the general modem architecture at physical layer.

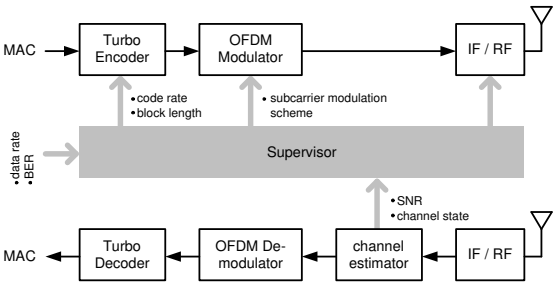

Fig. 1. WIND-FLEX physical layer concept

After turbo coding, the coded bits are mapped onto modulation symbols by BPSK, QPSK, 16-QAM and 64-QAM. These symbols are interleaved across subcarriers. According to the employed weak subcarrier excision algorithm, the strongest 92 subcarriers are dynamically selected out of the 100 active ones. Following the 128 point complex IFFT, the guard interval is added resulting in a OFDM symbol length of 3.0 μs. Then, 178 OFDM symbols

are arranged within one frame assuming that the channel state is quasi-static during this time.

Because of the supervisor concept the turbo coding scheme has to be flexible in terms of code rate and block length. Furthermore, it should offer a good compromise between performance and implementation complexity. Considering these requirements the following turbo coding scheme was selected to be applied:

- parallel concatenation of two recursive systematic convolutional (RSC) codes with polynomials $(13, 15)_{oct}$ (3GPP turbo code)
- offered code rates: 1/2, 2/3 and 3/4 (puncturing of mother code rate 1/3)
- S-random interleaving

A more detailed overview about coding and modulation aspects within the WIND-FLEX enviroment is given in [3].

3 Max-Log-MAP Turbo Decoding

The MAP algorithm is the optimal symbol-by-symbol SISO (soft-input soft-output) decoding scheme to decode the RSC codes of a turbo code. It is also called the (modified) Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [4]. So, a turbo decoder based on the MAP algorithm has the best performance but is of a high complexity. To reduce its complexity suboptimal MAP algorithms were developed [5]. One of these suboptimal algorithms is the Max-Log-MAP that is applied within the introduced system. The resulting performance loss due to this suboptimality can be compensated by scaling the Max-Log-MAP extrinsic value output during the iterative turbo decoding process [6]. To minimize memory consumption of an implementation and to make it independent from the coding block length the so called sliding window version of the Max-Log-MAP algorithm is employed [7].

Another SISO decoding algorithm is the SOVA, an extended Viterbi algorithm [8]. Improved versions of this algorithm reach a similar performance as MAP-based algorithms [9]. But with respect to hardware implementation the Max-Log-MAP algorithm is more suitable [10].

In the following, the Max-Log-MAP algorithm shall be described in a way aiming at implementation. Therefore integer arithmetic is assumed. Equations describing the algorithm are formed in a way that integer ranges become optimal from an implementation point of view. Furthermore, no clipping or quantization loss shall be caused within the Max-Log-MAP implementation. Clipping and quantization shall be allowed to keep extrinsic information (exchanged between turbo decoding iterations) in a feasable integer range only. The resulting performance loss is negligible compared to the loss caused by quantization of decoder soft-input within the demodulator.

3.1 Branch metric calculation

Let y_k^s and y_k^p be the systematic and parity (redundancy) RSC code part at decoder input with $k \in \{1, \ldots, N\}$. They are the decoder soft-input in form of Log-Likelyhood-Ratios (LLRs). In our application they are two's complement integers with 4 bit resolution provided by the demodulator. Let $L_k^{e,in}$ be the input extrinsic information from a previous decoding iteration step. $L_k^{e,in}$ is represented by a 6 bit integer and covers the range $\{-24, \ldots, +24\}$ only (see section 3.5). Now, the branch metrics are calculated as follows:

$$\gamma_k(s', s) = x_k^s(s', s) \cdot (L_k^{e,in} + y_k^s) + x_k^p(s', s) \cdot y_k^p \quad (1)$$

x_k^s and x_k^p are the code bits of a state transition from state s' to s, whereby $x_k^s, x_k^p \in \{-1, +1\}$. Because of calculating the branch metrics in that manner, only two different absolute values of them exist per trellis step k. Furthermore, the RSC code trellis can be separated into 4 butterfly elements. All 4 branch metrics associated with such a butterfly element are the same concerning its absolute value, they differ only in sign (see figure 2).

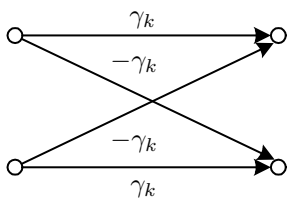

Fig. 2. Butterfly element of RSC code $(13, 15)_{oct}$ trellis

From an implementation point of view this can be used to reduce wiring effort. Only two instead of four different branch metrics have to be routed to and within forward/backward recursion units. But instead of additions only, subtractions have also to be performed inside them. On the other hand this method to calculate branch metrics (equation 1) doubles the integer range compared with what is minimal needed. This problem can be solved as will be shown in next subsection.

3.2 Forward and backward recursion

The forward and backward state metrics are calculated as follows:

$$\alpha_k(s) = \max_{s' \in S} \left(\alpha_{k-1}(s') + \frac{1}{2} \cdot \gamma_k(s', s) \right) \quad (2)$$

$$\beta_k(s') = \max_{s \in S} \left(\beta_{k+1}(s) + \frac{1}{2} \cdot \gamma_{k+1}(s', s) \right) \quad (3)$$

Looking at equation 1, it can be seen that the four branch metrics of a trellis step k are either all even integers or all odd ones, never both. This kind of redundancy within the branch metrics is eliminated by division by 2 in equations 2 and 3. This division is implemented in hardware by shifting a bit vector. In case of odd two's complement values, which are normally not divisible by 2, this implementation method is equivalent with subtraction of 1 and succeeding division by 2. Note that the inherent subtraction of 1 from all state metrics α_k and β_k respectively leads to no degradation of Max-Log-MAP performance.

3.3 State metric normalization

Equations 2 and 3 are recursive ones. The state metrics increase with increasing k due to maximum operation. Regarding soft value calculation, only the difference between state metrics at step k is of interest. Because of the RSC code trellis structure this difference is bounded [11]. Based on a branch metric range of $\{-40, \ldots, +40\}$ and equations 2 and 3, the state metric difference is bounded to 72.

There exist two main approaches for solving the normalization problem. To prevent an increased critical path in hardware implementations the so called modulo-normalization is often used. Its principle is described in [12]. But the integer range needed for state metrics is double of metric difference bound. The second and commonly used approach is the normalization of state metrics by conditional subtraction of a variable or fix value. This additional operation usually causes an increased critical path in hardware implementations. However, this can be circumvented by subtracting a fixed value that is a power of 2.

What we have implemented is subtraction of 32 from all state metrics α_k or β_k of a trellis step k if one of them is greater or equal than 104. This guarantees with respect to a metric difference bound of 72 that all state metrics are positive values. Because subtraction is done in the following trellis step, the state metric range reaches up to 123. So, what we need to store state metrics, are only 7 bit per value. See also figure 3.

The basic implementation unit that performs forward and backward recursion is the so called Add-Compare-Select (ACS) unit. It realizes equations 2 and 3. Because of its recursive character it is the bottleneck concerning the maximum clock frequency of the turbo decoder implementation. The normalization method introduced does not cause an increased critical path within the ACS unit.

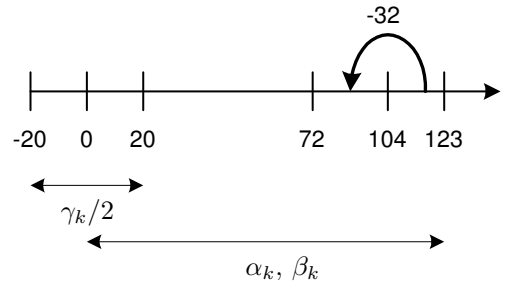

Fig. 3. Integer ranges at forward and backward recursion

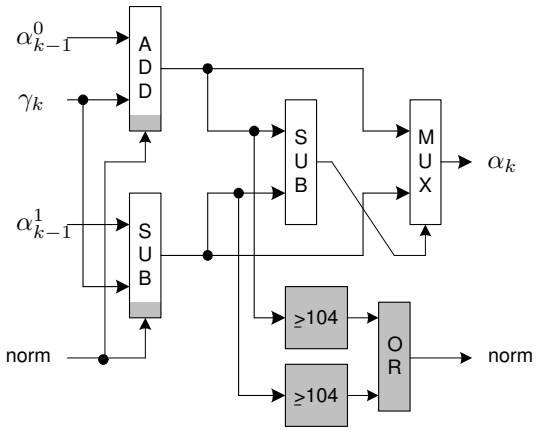

Fig. 4. Normalization within ACS unit (forward recursion)

As can be seen in figure 4, the comparison whether a state metric is greater or equal 104 is done in parallel with the state metric comparison (subtraction) and with multiplexing. Furthermore, the conditional subtraction of 32 can be done by manipulating some MSBs of the adders without increasing the critical carry chain inside.

3.4 Soft value calculation

The calculation of Max-Log-MAP decoder soft-output is done in the following way, whereby S^0 and S^1 indicate all state transitions caused by an information bit 0 and 1, respectively.

$$L_k = \max_{S^1} \left(\alpha_{k-1}(s') + \frac{1}{2} \cdot \gamma_k(s', s) + \beta_k(s) \right) - \max_{S^0} \left(\alpha_{k-1}(s') + \frac{1}{2} \cdot \gamma_k(s', s) + \beta_k(s) \right) \quad (4)$$

The sign bit of L_k represents the decoded bit. The extrinsic information that is needed for further turbo decoding iterations is calculated as follows:

$$L_k^{e,out} = L_k - L_k^{e,in} - y_k^s \quad (5)$$

The resulting extrinsic values $L_k^{e,out}$ are two's complement integers with 9 bit resolution. Note, that during all soft-value calculations no clipping is done and no other performance loss is caused.

3.5 Processing of extrinsic information

Extrinsic information exchanged between decoding iterations grows with successive iterations. Therefore the integer range of implemented arithmetics and memory consumption would become larger. The design of a Max-Log-MAP decoder covering the whole range needed by extrinsic information is not necessary because the extrinsic information range and resolution can be limited without significant performance degradation. Further details are presented in [13] and [14].

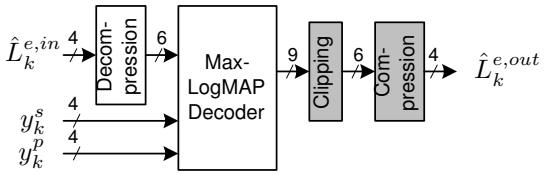

Fig. 5. Extrinsic information within SISO decoder

As can be seen in figure 5, extrinsic information is clipped to a 6 bit value at first. From BER performance point of view, clipping here leads to an earlier flattening of BER curves. But the selected range does not influence BER performance down to 10^{-6} which is required. At a second step, extrinsic information is mapped (compressed) by a look-up table to a 4 bit value. This step includes two effects. At first a nonlinear quantization and at second a scaling of extrinsic information.

Limitation of range and resolution of extrinsic information are design parameters and have to be selected as a compromise between BER performance loss and reduction of hardware consumption (arithmetic and memory word widths).

4 Parallelization Concepts

The implementation of high speed turbo decoders strongly depends on parallelization concepts. For turbo decoders based on sliding window MAP algorithms 5 different parallelization levels can be classified as shown in figure 6 (see [15]).

Depending on the levels that are used for parallelization, a turbo decoder implementation can be adapted to meet the requirements of throughput, flexibility and hardware consumption. The levels highlighted in grey (figure 6) are the ones where parallelization is applied in the introduced system.

4.1 Recursion level

Parallelization at recursion level is mandatory for high speed applications. This means the parallel calculation of all state metrics of a trellis step. Here, 4 double ACS-units are employed. A double ACS-unit performes the ACS operations of a butterfly element

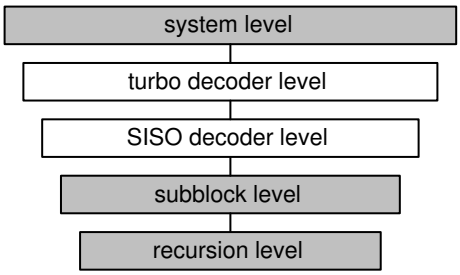

Fig. 6. Parallelization levels of turbo decoder implementations

as presented in figure 2. Advantageous here is the reduced wiring effort concerning branch metrics (see section 3.1).

4.2 Subblock level

Subblock level operations base on the sliding window principle described in [7]. Max-Log-MAP decoding is done subblock by subblock, each of length N_{sb}. There exists a third recursion type here called aquisition. This calculates initial values for backward recursion. The sliding windows principle is depicted in figure 7.

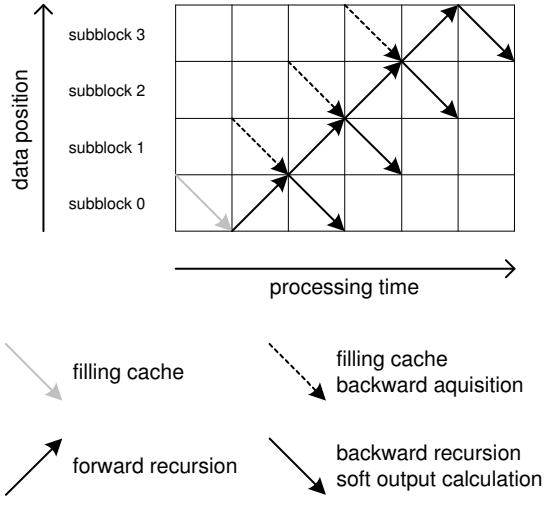

Fig. 7. Sliding window data processing

Figure 8 shows the structure of the SISO decoder. All 3 recursion types are processed in parallel. This is the highest parallelization at subblock level. It assumes a three times higher input memory access rate. To prevent this, two cache memories are introduced. Each of them stores N_{sb} branch metric sets of 2 times 7 bit. Forward state metrics are buffered until they are fed to soft value calculation. The latter is carried out in backward direction. The size of this forward state metric memory is

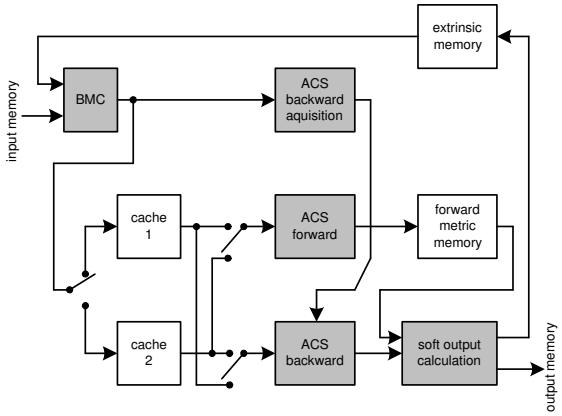

Fig. 8. SISO decoder structure

N_{sb} values of 7 bit width. Extrinsic information is buffered for one block to exchange themselve between SISO decoding steps. All buffers are dual port ones because they have to be read and written at the same time.

4.3 SISO decoder level

As described in the previous subsection, only one subblock is processed by ACS recursions at one time in the introduced sytem. Parallelization at SISO level would lead to processing of several subblock at the same time. This would require a higher memory access rate and complicates implementation due to the need for large word width buffers and due to interleaving.

The SISO processing time for one code block which is half of iteration time can be calculated as shown below:

$$T_{SISO} \approx \frac{N + 2N_{sb}}{f_{clock}} \tag{6}$$

As it can be seen, the SISO processing time per bit that corresponds to throughput depends on block length N. Highest throughput is 1 bit per clock providing $N \gg N_{sb}$.

4.4 Turbo decoder level

The turbo decoder implementation is based on the structure presented in figure 9.

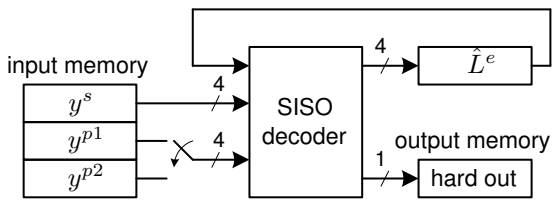

Fig. 9. Basic turbo decoder structure

At turbo decoder level only one SISO decoder is implemented. There exist 3 input memories containing the soft values y^s, y^{p1} and y^{p2} of a block of length N. The SISO decoder alternately processes the RSC code 1 and 2 iteration by iteration. Interleaving and deinterleaving is achieved by read and write operations of y^s and $\hat{L}^e$ memories.

Parallelization at turbo decoder level would mean to implement more than one SISO decoder. These are arranged in a pipelined manner. This principle fixes the number of decoding iterations. So, it can not be changed at runtime. Due to the need of flexibility, parallelization at this level has not been considered.

4.5 System level

The increase of throughput at system level leads to implementation of more than one turbo decoder. The decoder introduced here employes 6 turbo decoders which are implemented in parallel. Because of hardware limitations they are not completely independent from each other. SISO decoder and turbo decoder controlling is implemeted only ones. It includes also interleaver patterns stored in lookup tables. Therefore, all 6 turbo decoders synchronously process the incoming data blocks. This results in an increased decoding delay, because 6 blocks have to be received before decoding can start. However, throughput is not decreased. The input and output memories exist twice. So, if turbo decoders are active, input and output shadow memories can be filled and read out respectively by neighboring units.

5 Results

As result of our work the turbo decoder described was implemented in hardware. It was completely written in VHDL language and synthesized for an Altera APEX FPGA type EP20K600. It consumes 53% of logic and 57% of memory resources of this FPGA.

The turbo decoder is flexible in terms of block length and number of decoding iterations. These parameters can be changed at run time. Block length N has to be a multiple of N_{sb}, where N_{sb} is variable up to 32. Interleavers are implemented in form of lookup tables. Maximum block length and maximum number of interleavers depend on memory resources (input, output and interleaver memory). Our implementation supports block lengths up to $N = 512$ and includes 9 different S-random interleaver pattern.

Decoder throughput depends on block length N (see section 4.3) and on the number of decoding iterations. Based on a system clock of 50 MHz for $N \gg N_{sb}$ we reach a throughput of about 135 Mbps per full iteration. Therefore, the number of

decoding iterations is limited down to 1 iteration to handle highest system throughput of about 100 Mbps. For lower throughput modes more iterations are performed.

6 Conclusions

In this paper a turbo decoder implementation for a high throughput OFDM-based WLAN demonstrator was presented. Aiming at an optimized implementation of algorithms, it was shown how the Max-Log-MAP algorithm has to be modified to minimize the word width of arithmetic operations as well as the consumption of memory.

To reach highest throughput, the state metric normalization procedure within the ACS-unit was optimized in a way that the critical path is not increased. Furthermore, different parallelization concepts have been discussed and finally realized.

References

[1] A. Polydoros et al., "Wind-flex: Developing a novel testbed for exploring flexible radio concepts in an indoor enviroment", *IEEE Communications Magazine*, pp. 116–122, July 2003.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes", *ICC '93*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[3] M. Benedix, J. Ertel, and A. Finger, "Turbo coding for an ofdm-based wireless lan at 17 ghz", *7th International OFDM-Workshop*, Hamburg, September 2002, pp. 137–141.

[4] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Transactions on Information Theory*, pp. 284–287, March 1974.

[5] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain", *ICC*, Seattle, June 1995, pp. 1009–1013.

[6] J. Vogt and A. Finger, "Improving the max-log-map turbo decoder", *IEE Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, November 2000.

[7] A.J. Viterbi, "An intuitive justification and a simplified implementation of the map decoder for convolutional codes", *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 260–264, February 1998.

[8] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications", *GLOBECOM*, Dallas, November 1989, pp. 1680–1686.

[9] Z. Wang and K.K. Parhi, "High performance, high throughput turbo/sova decoder design", *IEEE Transactions on Communications*, vol. 51, no. 4, pp. 570–579, April 2003.

[10] J. Vogt, K. Koora, A. Finger, and G. Fettweis, "Comparison of different turbo decoder realizations for imt-2000", *GLOBECOM*, Rio de Janeiro, December 1999, pp. 2704–2708.

[11] P. Siegel, C. Shung, T. Howell, and H. Thapar, "Exact bounds for viterbi detector path metric differences", *ICASSP*, May 1991, pp. 1093–1096.

[12] A.P. Hekstra, "An alternative to metric rescaling in viterbi decoders", *IEEE Transactions on Communications*, vol. 37, no. 11, pp. 1220–1222, November 1989.

[13] J. Vogt, J. Ertel, and A. Finger, "Reducing bit width of extrinsic memory in turbo decoder realizations", *IEE Electronics Letters*, vol. 36, no. 20, pp. 1714–1716, September 2000.

[14] J. Vogt, *Beiträge zur effizienten Decodierung von Turbo-Codes*, Ph.D. thesis, Technische Universität Dresden, 2002.

[15] M.J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A scalable system architecture for high-throughput turbo-decoders", *Workshop on Signal Processing Systems*, 2002.