The 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07)

IMPLEMENTATION ISSUES OF TURBO SYNCHRONIZATION WITH DUO-BINARY TURBO DECODING

M. Alles, T. Lehnigk-Emden, U. Wasenmüller, N. Wehn
{alles, lehnigk, wasenmueller, wehn}@eit.uni-kl.de
Microelectronic System Design Research Group
University of Kaiserslautern, 67663 Kaiserslautern, Germany

ABSTRACT

The transmission over a wireless channel results in timing, frequency and phase offsets. To circumvent the severe losses of communications performance caused by these offsets a sophisticated synchronization is mandatory. Synchronization is typically performed only once prior to the channel decoding. In this paper the authors present an FPGA implementation of a joint iterative decoder and synchronizer, which is also referred to as turbo synchronizer. We investigate the additional costs of turbo synchronization in terms of implementation complexity with a 16-state duo-binary turbo decoder. Furthermore we present the communications performance of the turbo synchronizer taking the implementation losses into account.

I. INTRODUCTION

With the invention of turbo codes [1] and the rediscovery of LDPC codes in the 1990s, iterative decoding has become a major research topic. Both codes belong to the best channel codes known today. Due to their outstanding communications performance near the Shannon limit they have become in the meantime part of a wide range of communication standards. The extension of binary turbo codes to duo-binary turbo codes [2] allows for even better bit error rates (BERs) at a given signal-to-noise ratio (SNR).

Since receiver and transmitter are not synchronized the transmission of data over a wireless channel results in timing, phase and frequency offsets. For instance the unknown delay of the transmission causes an unknown phase offset between receiver and transmitter. Even small phase or frequency offsets result in a severe loss of communications performance, hence a high performance synchronization on the receiver side is indispensable.

The performance of the synchronization strongly depends on the SNR. Due to the low SNR that is used in combination with advanced channel codes the task of synchronization on the receiver side is more challenging than with traditional channel codes. Usually so called pilot symbols are inserted into the data stream to cope with the low SNR. These pilot symbols are then used on the receiver side to perform the synchronization prior to the decoding.

Since the spectral efficiency is decreased by the pilot symbols one tries to keep their number small. This can be achieved by a joint iterative decoding and synchronization (so called turbo synchronization), which is a current topic of research, e.g. [3][4]. Turbo codes are decoded iteratively. In turbo synchronization the synchronization is performed within the iterative channel decoding loop. Thus the information of the channel decoder process can be used to perform an efficient synchronization.

To the best of our knowledge we are the first to present not only an implementation of a turbo synchronizer but also of a 16-state duo-binary turbo decoder. We investigate the implementation complexity and communications performance of the turbo synchronization using a Xilinx FPGA.

The paper is structured as follows. Section II. introduces duo-binary turbo codes briefly, while Section III. gives a short introduction of the synchronization principle. In Section IV. we discuss the implementation issues of the turbo synchronization and turbo decoder. The results are then given in Section V. Finally Section VI. concludes the paper.

II. DUO-BINARY TURBO CODES

Turbo codes in general consist of a serial or parallel concatenation of two convolutional codes. The widely used binary turbo codes like the UMTS one use two equal recursive systematic convolutional (RSC) codes, named component codes, which are parallel connected through an interleaver, see Figure 1a). The next generation of turbo codes are the duo-binary turbo codes (DB-TC) introduced by Berrou in 1999 [2]. In contrast to the binary ones, 2 bits u_1, u_2 are used simultaneously to calculate the parity bits. Duo-binary turbo codes show a better communications performance than the binary ones [5].

The performance of a code strongly depends on the polynomial for the component code and the choice of the interleaver. This gives the designer a large degree of freedom. The component code can be described by three matrices. The matrix G is the generator matrix of the linear feedback shift register. The connection matrix C defines the connections of the inputs bits with the register stages and the redundancy matrix R describes the taps for the parity bits. To guarantee a large minimum distance we use the matrices from [5] for component codes with constraint length 5 (16 state code).

The vector $u_i = (u_{1i}, u_{2i})^T$ contains the information bit couple (information symbol) at time step i. The state vector S_i represents the memories in the component encoder. With these matrices the parity bits $p_i^{1,2}$ and the state vector are calculated:

$$\begin{aligned} p_i^{1,2} &= \sum_{j=1,2} u_{ji} + RS_i \\ S_{i+1} &= GS_i + Cu_i. \end{aligned} \tag{1}$$

p^1 denotes the parity bits calculated from the component decoder 1, p^2 from component decoder 2 respectively.

Convolutional codes have a quasi-infinite block length. One of the best techniques to obtain a block code from a convolu-

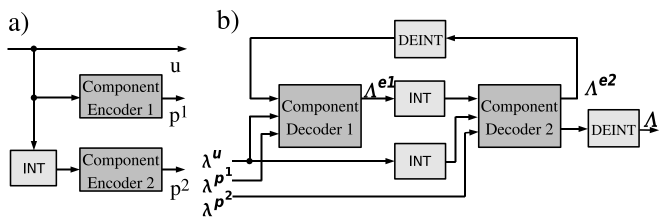


Figure 1: Structure of a) Turbo Encoder and b) Turbo Decoder

tional code is tail-biting [5]. The encoder starts and ends in the same state S for each block. This results in a circular code trellis without state-discontinuity. By using this technique no additional bits have to be transmitted to terminate the trellis, thus the code rate is not decreased and a decrease of the hamming distance by these termination bits is avoided. Tail-biting requires circular encoding what causes an additional computation complexity at the encoder, because the block has to be encoded twice. The first encoding step is needed to determine the start/end state S for each block.

In this configuration the DB-TC encoder has a code rate of $1/3$. The rate can be easily adapted by a puncturing unit. In our case we use a regular puncturing scheme, i.e., for a code rate of $1/2$ each second parity bit is punctured out.

A. *Interleaver*

The interleaver is the key to the excellent communications performance of the duo-binary turbo code. For high throughput applications parallel decoder architectures become mandatory which can yield access conflicts [6]. However many interleaver types exist which allow for a conflict free implementation of a parallel decoder. In the following, we will consider the *almost regular permutation* (ARP) interleaver [5] which is similar to the *dither relative prime* (DRP) interleaver. The interleaver process consists of two steps. The first step swaps every second data pair. In the second step, a vector $v = (v_1, ..., v_N)$ is filled linearly with data couples u. The new position i of the jth data couple is given by Equation 2. Only six parameters are necessary to describe the whole interleaver. The permutation factor P determines a global permutation of the couples over the block with the length N, while the four parameters Q_0, Q_1, Q_2 and Q_3 are responsible for a local permutation of the couples. The value i_0 is a constant offset factor.

$$i(j) = Pj + Q(j) + i_0 \;\; mod\, N, \quad j = 0, \ldots, N-1 \qquad (2)$$

$$Q(j) = \begin{cases} 0 & \text{if } j \; mod\; 4 = 0 \\ 4Q_1 & \text{if } j \; mod\; 4 = 1 \\ 4(Q_0 P + Q_2) & \text{if } j \; mod\; 4 = 2 \\ 4(Q_0 P + Q_3) & \text{if } j \; mod\; 4 = 3 \end{cases}$$

B. *Decoding Algorithm*

A possible realization of a decoder of turbo codes is given in Figure 1b). The two component decoders that decode the two component codes are connected via interleaver and deinterleaver. They use the *log likelihood ratio*s (LLR) λ^u, λ^{p^1} and λ^{p^2} of the systematic and parity information to compute the extrinsic information Λ^{e1} and Λ^{e2} on the information couples. The iterative exchange of Λ^{e1} and Λ^{e2} between these component decoders is referred to as turbo principle.

Both component decoders perform a *maximum a posteriori probability* (MAP) decoding. For implementation the suboptimal Max-Log MAP algorithm with *extrinsic scaling factor* (ESF) is suitable. In comparison to the optimal algorithm the Max-Log MAP results in a performance loss below 0.2 dB [7][8]. Moreover it was shown in [9] that, when employed in turbo decoding, one does not require knowledge of the SNR.

The Max-Log MAP algorithm consists of a forward and a backward recursion along the trellis graph with the time step k and the state m of the component code. It computes for each possible information or parity symbol $\boldsymbol{d}_k = (d_{1k}, d_{2k})$ an *a posteriori probability* (APP) LLR. The APP LLRs can be expressed using three metrics, whereas two of them refer to the encoder states $S_k^{(m)}$: the *state metrics* α_k^m and $\beta_{k+1}^{m'}$. The third metric is the branch metric $\gamma_{k,k+1}^{m,m'}$ which describes the transition from the state m to the following state m' in the trellis depending on the received symbol. The α- and β-metrics are gathered in a *forward* and *backward recursion*, respectively:

$$\alpha_{k+1}^{(m')} = \min_{\forall m} \left(\alpha_k^{(m)} + \gamma_{k,k+1}^{m,m'} \right) \qquad (3)$$

$$\beta_k^{(m)} = \min_{\forall m'} \left(\beta_{k+1}^{(m')} + \gamma_{k,k+1}^{m,m'} \right). \qquad (4)$$

The LLR computation of the received symbol d_k then turns into:

$$\begin{aligned} \Lambda_{\boldsymbol{d}_k}^{(i)} &= \ln \frac{\Pr\{\boldsymbol{d}_k = 0|\boldsymbol{y}\}}{\Pr\{\boldsymbol{d}_k = i|\boldsymbol{y}\}} \qquad (5) \\ &= \min_{\forall(m,m')} \left(\gamma_{k,k+1}^{m,m'}(\boldsymbol{d}_k = i) + \alpha_k^{(m)} + \beta_{k+1}^{(m')} \right) \\ &\quad - \min_{\forall(m,m')} \left(\gamma_{k,k+1}^{m,m'}(\boldsymbol{d}_k = 0) + \alpha_k^{(m)} + \beta_{k+1}^{(m')} \right), \end{aligned}$$

with $i \in \{0, \ldots, 3\}$. The hard decoded symbol is the index i given by the minimum of $\Lambda_{\boldsymbol{d}_k}^{(i)}$.

III. Synchronization

The synchronization consists of the estimation of the unknown parameters of timing, frequency and phase offset, and the elimination of all possible negative influences introduced by these parameters. We focus on the frequency and phase synchronization of bursts with Quadrature Phase Shift Keying (QPSK) modulation in conjunction with turbo decoding. We assume, that the steps of gain control, timing and burst detection are properly carried out. The received sample sequence r is given in the complex baseband according to Equation 6:

$$r(l) = s(l) \cdot e^{j(2\pi f_o l + \Phi)} + n(l) \qquad l = 0, 1, \ldots, L-1 \quad (6)$$

The sample sequence r with L elements is based on QPSK symbols s with one sample per symbol and symbol duration T, and is disturbed by a noise sequence n. In Equation 6 the frequency offset f_o is annotated as a fraction of the symbol rate $1/T$. The frequency offset f_o and phase offset Φ have to be estimated and corrected. They are considered to be fixed during an estimation interval. The synchronization is done in two main steps. Initially a coarse synchronization is carried out with the help of pilot symbols. Afterwards fine synchronization is done iteratively with the additional use of tentative decoder decisions after each decoder iteration.

A. *Coarse Synchronization*

According to [4] pilot blocks with L_p pilot symbols are uniformly inserted in the stream of coded symbols. Depending on the block length of the turbo code one or more segments are thus created with the structure of a preamble followed by L_c coded symbols and a postamble. A measure for the average phase of the ith pilot block is calculated by modulation removal for the received symbol subsequence $r_{p,i}$ of r corresponding to the known pilot symbol sequence $a_{p,i}$ of the ith pilot block in the so called data aided way:

$$Z_p(i) = \sum_{l=1}^{L_P} r_{p,i}(l) \cdot a^*_{p,i}(l). \tag{7}$$

With the results of Equation 7 the frequency offset $\tilde{f}_0$ and the phase offset $\tilde{\phi}$ for each segment is estimated:

$$\tilde{f}_0 = \frac{arg(Z_p(i+1) \cdot Z^*_p(i))}{2\pi(L_p + L_c)} \tag{8}$$

$$\tilde{\phi} = arg(Z_p(i+1) + Z_p(i)) - (2L_p + L_c) \cdot \tilde{f}_0 \cdot \pi. \tag{9}$$

The received sequence r is corrected segment by segment. The LLR values λ^u, λ^{p^1} and λ^{p^2} of the transmitted bits for the decoder are calculated on base of the corrected sequence.

B. *Fine Synchronization*

Each decoding iteration produces LLR values of the transmitted coded bits according to Equation 5. The hard decoded symbols can be calculated with these LLR values and provide a tentative decision of the transmitted symbols. For pure decoding the hard decoded symbols must only be calculated for the systematic bits after the last decoder iteration. For the purpose of iterative synchronization, however, the hard decoded symbols of systematic and parity bits have to be calculated after each decoder iteration. This tentative estimate of the codeword is used for synchronization purposes as a known block of symbols. Simulations showed that it is sufficient to make a fine correction of the phase offset. A measure for the average phase of the coded part of a segment is given by

$$Z_c(i) = \sum_{l=1}^{L_c} r_{c,i}(l) \cdot \tilde{a}^*_{c,i}(l)), \tag{10}$$

where $\tilde{a}_{c,i}$ denotes the estimated coded symbol sequence in the ith segment and $r_{c,i}$ the corresponding received symbol subsequence of r. With the results of Equation 10 and the results of the pilot blocks (Equation 7) the phase offset $\tilde{\phi}$ can be iteratively estimated.

$$\tilde{\phi} = arg(Z_p(i) + Z_c(i) + Z_p(i+1)). \tag{11}$$

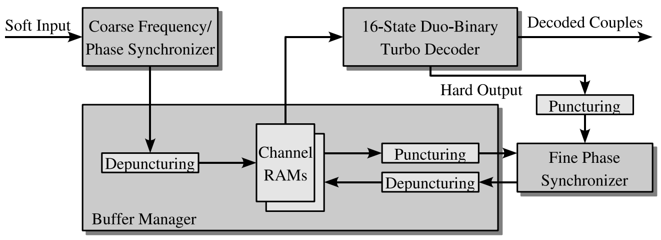


Figure 2: Turbo Synchronizer Architecture

IV. Implementation Issues

The challenge of turbo synchronization is the mutual exchange of information between decoder and synchronizer. As the decoder needs synchronized information and the synchronizer needs decoded information the components in the system have to communicate a lot with each other. Hence the data bandwidth of each single component is the key for an efficient implementation.

In Figure 2 the turbo synchronizer architecture is depicted. It consists of four building blocks, the coarse frequency and phase synchronizer, a buffer manager, the duo-binary turbo decoder and the fine phase synchronizer. We use Xilinx FPGAs and efficiently exploit the dual-ported RAMs (BRAMs) offered by the FPGA.

The system works as follows. After coarse synchronization the received data stream is depunctured and stored in a channel RAM. The data in the channel RAM is then copied to the turbo decoder. After a decoder iteration is carried out hard decisions of the codeword couples are available. The fine synchronizer uses these hard decoded couples and the channel values stored in the buffer manager to perform its operation. Additionally it is necessary to puncture the information of the decoder for the synchronization. After synchronization depuncturing of the fine synchronized data must be performed again.

Fine synchronizer and turbo decoder work in parallel to achieve a high throughput with turbo synchronization. Once the content of the channel RAM is updated completely and the decoder has finished its iteration the decoder stops and reads the turbo synchronized data for the following decoding iteration from the channel RAM. Double buffering in the buffer manager allows to perform a coarse synchronization even while the turbo synchronization of a previous codeword is still in progress.

A. *Duo-Binary Turbo Decoder*

The architecture of the duo-binary turbo decoder given by Figure 1b) is depicted in Figure 3. A local channel RAM is used to store scaled channel values (systematic and parity couples). Scaling is performed using the channel reliability factor (CRF). Furthermore the decoder includes a MAP unit that acts as component decoder, an ARP interleaver and an interleaver table that perform interleaving, two extrinsic memories that realize the exchange of extrinsic information between the component decoders and, finally, a memory that stores hard decoded couples.

One turbo decoder iteration is split into two half iterations. During the first half iteration the MAP acts as component de-

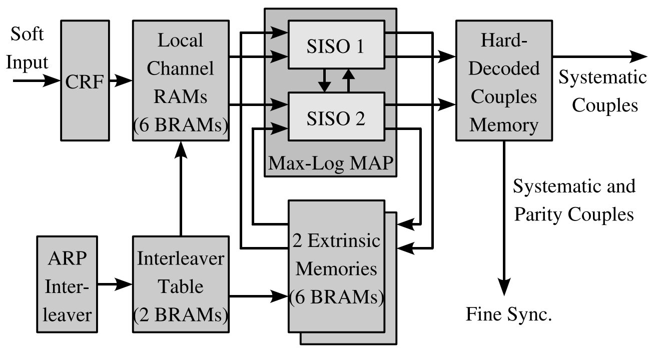

Soft Input
CRF
Local Channel RAMs (6 BRAMs)
SISO 1
SISO 2
Max-Log MAP
Hard-Decoded Couples Memory
Systematic Couples
Systematic and Parity Couples
Fine Sync.
ARP Inter-leaver
Interleaver Table (2 BRAMs)
2 Extrinsic Memories (6 BRAMs)

Figure 3: Architecture of the Duo-Binary Turbo Decoder

coder 1. Values from the channel RAM and one of the extrinsic RAMs are read in a linear manner to perform MAP decoding. Afterwards the newly computed extrinsic information is written back in a linear manner to the second extrinsic memory. Furthermore hard decisions of the p^1 couples are stored in the memory which is connected to the fine synchronizer.

In the second half iteration the component code 2 is processed, respectively. Since it is now necessary to use interleaved data as input for the MAP, addressing of the channel RAM and the extrinsic RAM needs to be done by the ARP interleaver. The updated extrinsic information is written deinterleaved to the extrinsic RAM whereas the hard decisions of the systematic and p^2 couples are written deinterleaved to the hard decision memory.

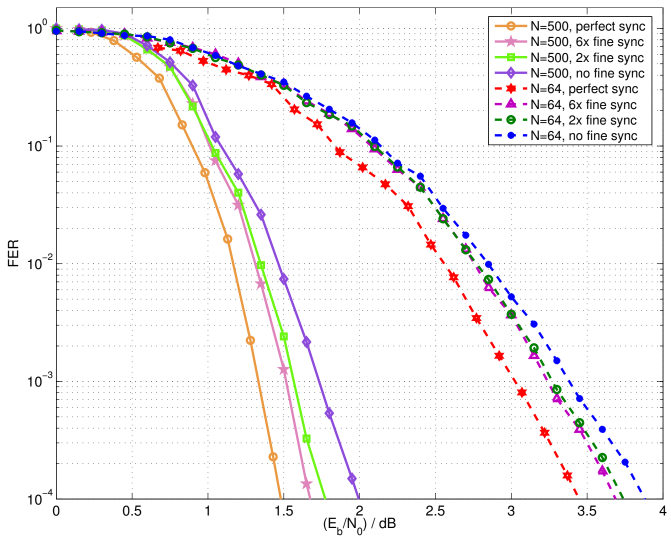
To increase throughput two soft-input soft-output decoders (SISOs) are used to realize the Max-Log MAP algorithm. The codeword is hence split into two equal sub-blocks which are then distributed to the SISOs. After an initial latency each SISO computes the extrinsic information of one information couple per clock cycle, which is possible when three recursion units for the state metric calculations are used. A detailed description of such a SISO for binary turbo codes is given in [6].

B. *Synchronization*

The coarse synchronizer is based on direct implementation of the equations as given in the previous section.

The fine synchronizer performs phase correction during the iterations of the decoding process. The phase synchronization is done in the same way as in the coarse synchronizer with the difference that the reference pilots and the hard decoded bits from the decoder are used to estimate the phase offset. Two scheduling procedures can be considered:

1. The fine synchronization process takes place between two decoder iterations. That means that either the decoder or the synchronizer is working. This serial scheduling results in an increased system latency and a suboptimal hardware utilization.
2. Fine synchronization is carried out in parallel to the decoding process. This scheduling scheme is employed in our system. The system throughput is only slightly affected if the fine synchronization takes about the same amount of time as one decoder iteration. To fulfill this constraint the fine synchronizer corrects up to four symbols per clock cycle.



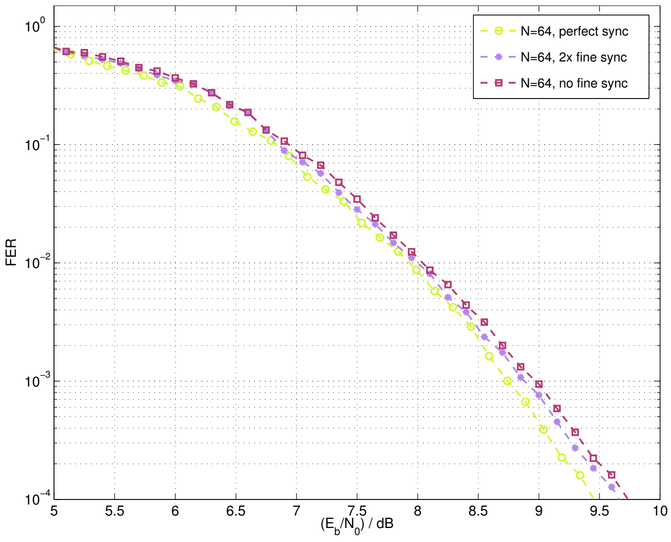


Figure 4: Communications Performance with and without Fine Synchronization, Codes Rates $^1/_3$ (top) and $^6/_7$ (bottom)

V. Results

A. *Communications Performance*

Simulations show the advantage of the turbo synchronization in contrast to the coarse synchronization only and in comparison to an ideal frequency and phase estimation. The simulation are carried out with the bit true models of the hardware units to take into account the quantization losses. As mentioned before we use the component code from [5]. Furthermore the interleaver parameters are calculated by the algorithm proposed in [5]. The number of turbo decoder iterations is 8.

The top of Figure 4 shows codes with 128 and 1000 information bits and rate $^1/_3$. Simulations are carried out with a frequency offset of $f_0 = 10^{-3}$ and a phase offset of $\Phi = 115°$. For the code with rate $^1/_3$ the block contains 15% pilot symbols for the short block and 3% for the long block. For this

Decoder	16-State Duo-Binary Turbo Decoder			
Algorithm	Max-Log-MAP with ESF			
Information Couples	64-2048			
Code Rate	$^1/_3$-$^7/_8$			
DB-TC Iterations	8			
Sync.	Coarse		Turbo	
Sync. Iterations	0		2	
Throughput	7.0 - 27.0 Mbps		6.9 - 25.6 Mbps	
Comm. Performance	see Figure 4			
XC4VLX80-12 FPGA @ 120 MHz				
Component	Slices	BRAMs	Slices	BRAMs
Coarse F/P Sync.	1,450	3	1,450	3
Buffer Manager	1,021	12	1,904	14
DB-TC Decoder	16,873	14	20,295	14
Fine P Sync.	–	–	1,369	2
Overall	18,424	29	24,410	33

Table 1: Implementation Results

code rate an improvement of the communications performance up to 0.3 dB by the turbo synchronization can be observed for both codes. By increasing the number of fine synchronizations to 6 the perfect synchronization performance is missed by 0.2 dB. For the higher code rate of $^6/_7$, see bottom of Figure 4, we are operating at a high SNR region where the coarse synchronization already has a good performance. For this code rate a block contains 30% pilot symbols. The turbo synchronization results in a performance gain of 0.1 dB only. The gap to the perfect synchronization is below 0.1 dB.

B. Implementation

The architecture was implemented as a synthesizable VHDL model. Table 1 gives an overview of the FPGA resources. Block lengths from 64 to 2048 information couples in steps of two are supported by the decoder. The eleven different code rates range from $^1/_3$ to $^7/_8$. An 8 bit input quantization is used for the coarse synchronization. The decoder uses a 6 bit quantization for channel values.

We implemented both, the system with and without turbo synchronization, to measure the additional costs of the turbo synchronization in terms of implementation complexity. The overall slice count increases by 33% from 18,424 slices to 24,410 slices, while the additional memory requirement increases by 14%. There are three reasons for this increase:

1. An additional unit is necessary to perform the fine phase synchronization.
2. The buffer manager has to send and receive soft information to and from this additional synchronizer, hence its design becomes more complex. Also additional RAMs are required to store the pilot symbols.
3. The turbo decoder has to compute not only hard decoded couples of the systematic bits but also of the parity bits. This is not necessary when turbo synchronization is not performed. Furthermore these parity bits need to be stored in the memory for hard decoded couples and communication is required between turbo decoder and fine synchronizer.

The clock frequency of 120 MHz is mainly determined by routing congestions on the FPGA. For the longest blocks we achieve a throughput of 27.0 Mbps when turbo synchronization is not used. Performing two fine synchronization iterations, the throughput decreases slightly to 25.6 Mbps. This is due to the fact that decoder and fine synchronization work in parallel and thus it is only necessary to stall the decoder when its channel values have to be updated after the fine synchronization.

VI. Conclusion

To the best of our knowledge we are the first to present not only an implementation of a turbo synchronizer but also of a 16-state duo-binary turbo decoder. With turbo synchronization it is possible to approach the communications performance of a perfect synchronization and we demonstrated the implementation complexity.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," in *Proc. 1993 International Conference on Communications (ICC '93)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[2] C. Berrou and M. Jezequel, "Non-Binary Convolutional Codes for Turbo Coding," *Electronic Letters*, vol. 35, no. 1, pp. 39–40, January 1999.

[3] V. Lottici and M. Luise, "Embedding Carrier Phase Recovery Into Iterative Decoding of Turbo-Coded Linear Modulations," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 661–669, Apr. 2004.

[4] S. Godtmann, A. Pollok, N. Hadaschik, W. Steinert, G. Ascheid, and H. Meyr, "Joint Iterative Synchronization and Decoding Assisted by Pilot Symbols," in *IST Mobile & Wireless Communications Summit*, Myconos, Greece, July 2006.

[5] C. Douillard and C. Berrou, "Turbo Codes with rate-m/(m+1) constituent convolutional codes," *IEEE Transactions On Communications*, vol. 53, no. 10, pp. 1630–1638, oct 2005.

[6] M. J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A Scalable System Architecture for High-Throughput Turbo-Decoders," *Journal of VLSI Signal Processing Systems (Special Issue on Signal Processing for Broadband Communications)*, vol. 39, no. 1/2, pp. 63–77, 2005, springer Science and Business Media, Netherlands.

[7] P. Robertson, E. Villebrun, and P. Hoeher, "A Comparison of Optimal and Sub-Optimal MAP decoding Algorithms Operating in the Log-Domain," in *Proc. 1995 International Conference on Communications (ICC '95)*, Seattle, Washington, USA, June 1995, pp. 1009–1013.

[8] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding," *European Transactions on Telecommunications (ETT)*, vol. 8, no. 2, pp. 119–125, March–April 1997.

[9] A. Worm, P. Hoeher, and N. Wehn, "Turbo-Decoding without SNR Estimation," *IEEE Communications Letters*, vol. 4, no. 6, pp. 193–195, June 2000.