

Design and implementation of Turbo Decoder for IEEE-802.16e and LTE standards

Syed Z. Gilani

May 14, 2009

1 Introduction

Convolutional Turbo encoding is employed for channel coding in most modern communication standards. It has been shown to provide better bit error rate performance than convolutional encoding but this increase in performance comes at a considerable cost in implementation complexity to achieve required decoding throughput. Convolutional Turbo codes (CTC) are constructed by a parallel concatenation of two or more component codes. The component encoders operate on the interleaved versions of the same data and reduce the channel fading effects by making the encoded data sequences independent of each other. The IEEE-802.16e (WiMax) standard defines

constituent encoder first encodes the input bit stream and then encodes an interleaved version of it. The interleaving is performed in a two step process in which an input sequence of block length N is first interleaved as follows:

for $i = 0 \rightarrow N - 1$

$$\begin{aligned} \text{if } (\text{mod}(i, 2) == 0) \quad & (A_i, B_i) = (B_i, A_i) \\ \text{else} \quad & (A_i, B_i) = (A_i, B_i) \end{aligned} \quad (1)$$

In step 2, the interleaved address for each pair of bits in the data sequence is generated according to the following expression:

for $j = 0 \rightarrow N - 1$

$$\begin{aligned} & \text{switch}(j \text{ mod } 4) \\ & \text{case0} : P(j) = (P_0 \cdot j + 1)_{\text{mod } N} \\ & \text{case1} : P(j) = (P_0 \cdot j + 1 + N/2 + P_1)_{\text{mod } N} \\ & \text{case2} : P(j) = (P_0 \cdot j + 1 + P_2)_{\text{mod } N} \\ & \text{case3} : P(j) = (P_0 \cdot j + 1 + N/2 + P_3)_{\text{mod } N} \end{aligned} \quad (2)$$

where the values of P_0, P_1, P_2 and P_3 depend upon the block size of data and are defined in the standard. The turbo encoder structure for LTE standard is shown in figure 2. The encoder outputs a systemic bit(input bit) and two parity bits for each bit of encoded data. One parity bit is generated by the interleaved sequence and the other by the deinterleaved sequence. The interleaving function for the LTE standard is shown in 3. It uses two parameter f_1 and f_2 whose values depend upon the block size, N , of encoded data. The block sizes vary from 40 bits to 6144 bits.

$$P(i) = (f_1 i + f_2 i^2)_{\text{mod } N} \quad (3)$$

The rest of this paper is organized as follows, section II describe the Turbo decoding algorithm, section III

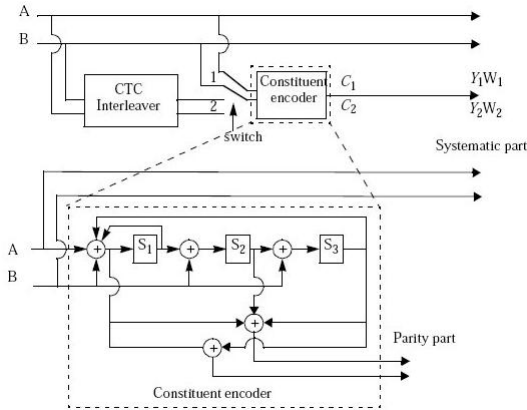


Figure 1: IEEE 802.16e Turbo Encoder

a duo-binary turbo encoding scheme with two component encoders. The term duo-binary refers to the fact that each of the component encoding is done on a pair of bits rather than a single bit. A block diagram of the IEEE-802.16 convolutional turbo encoder is shown in 1. The polynomials defining the component convolutional encoders are $1 + D + D^3$ (for the feedback branch) and $1 + D^2 + D^3$ (for the parity branch). The

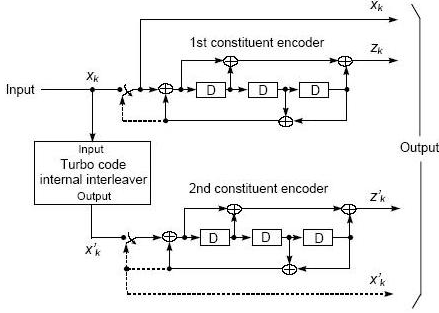


Figure 2: LTE Turbo Encoder

provides implementation details and section IV provides synthesis and simulation results.

2 TURBO DECODING ALGORITHM

The turbo decoding requires soft-input-soft-output (SISO) decoders to provide bit-level loglikelihood ratios for all the encoded bits. These log-likelihood ratios (LLR) are determined as:

$$\Lambda(c_t) = \log \frac{P_r[c_t = 1]}{P_r[c_t = 0]} \quad (4)$$

where c_t is the decoded bit at time t given a received sequence r . Since LTE turbo encoding encodes each bit separately, unlike Wimax which encodes a pair of bits together, the LLR values of LTE turbo decoding represent the log-likelihood ratios of 0 and 1. For Wimax, four LLR values need to be determined corresponding to each possible pair of bits, 00, 01, 10 and 11. The SISO decoders can be implemented optimally using maximum-a posteriori (MAP) algorithm or more generally with approximations of the MAP algorithm to reduce implementation cost and complexity. Turbo decoding requires several iterations over a block of data to improve its bit estimates using the log-likelihood ratios generated by the SISO decoders in the previous iteration.

2.1 MAP algorithm

The MAP algorithm performs forward and backward recursions on the state trellis of constituent encoders and compares the metrics of each recursion to estimate its LLR. The algorithm can be summarized in the following steps. First, for each pair of encoded

bits, compute the branch metrics (γ) according to 5 for all trellis paths corresponding to previous state l' and current state l .

$$\gamma_t^i(l', l) = \left[p_t(i) \exp \frac{\sum_{j=0}^{n-1} (r_{t,j}^i - x_{t,j}^i)}{2\sigma^2} \text{for } (l', l) \in B_t^i \right] \quad (5)$$

where $p_t(i)$ is the a priori probability of $c_t = i$ and $x_{t,j}^i(l)$ is the encoder output associated with the transition $S_{t-1} = l' \rightarrow S_t = l$ and input $c_t = i$. B_t^i is the set of transitions that are caused by the input $i \in 00, 01, 10, 11$ (Wimax) or $i \in 0, 1$ (for LTE). For actual implementation (5) can be calculated in log-domain, and the term $2\sigma^2$ can be ignored. The resulting approximate algorithm is referred to as the log-MAP algorithm. The forward path metrics $\alpha_t(l)$ and backward path metrics $\beta_t(l)$ are computed as:

$$\alpha_t = \max \left(\alpha_{t-1}(l) + \gamma_t^i(l', l) \right) \quad (6)$$

$$\beta_t = \max \left(\beta_{t-1}(l) + \gamma_t^i(l', l) \right) \quad (7)$$

In the final step, LLR of each bit pair is determined as:

$$\Lambda^i \approx \max_{\text{for all } i} \left(\gamma_t^i(l', l) + \alpha_{t-1}(l') + \beta_t(l) \right) \quad (8)$$

These steps remain unchanged for both LTE and Wimax standards however, for LTE $i \in 0, 1$ while for Wimax decoding $i \in 00, 01, 10, 11$. The final LLR value of an iteration for LTE is positive/negative if the decoded bit is 0/1. For Wimax however, all four LLR values need to be forwarded to the next iteration.

2.2 Bottlenecks and constraints

The turbo decoding algorithm performs several iterations over the received data block to improve the log-likelihood ratio of decoded bits. Each iteration is composed of processing of the complete data block according to the MAP algorithm defined by equations (6), (7) and (8). Since the algorithm requires both backward and forward processing of the trellis, where each trellis metric depends upon its value at the previous trellis step. In order to decode a complete block of data, the two metrics, forward and backward, need to be calculated and stored in the memory and then compared to provide the log-likelihood ratio. However, this approach can require large amounts

of memory particularly for large data sizes, as supported by both Wimax and LTE standards. For example, for a block size of 6144 bits, $2 \times 6144 = 12288$ entry forward and backward metric memory will be required to store the state metrics. Even if each entry is only 1 byte wide, the total memory required just by the α and β metrics is more than 12KB. Moreover, since gamma values are also needed for the final LLR calculation and for the alpha and beta metrics, it needs to be stored in a separate memory. For Wimax, there are 16 possible values of gamma for each trellis step. Storing all these values for all the data block can result in gamma memory as huge as $16 \times 1 \text{ byte} \times 6144 = 98304$ bytes. Other than memory usage in MAP decoder, the complete Turbo decoding algorithm works on blocks of data, the LLR values of the complete block need to be stored in interleaved and deinterleaved memories from where the decoded data can be read and written to. Since the data passes through multiple iterations where each iteration is reading an interleaved version of data, both interleaved and deinterleaved memories are required. Finally, the multiple iterations of data during

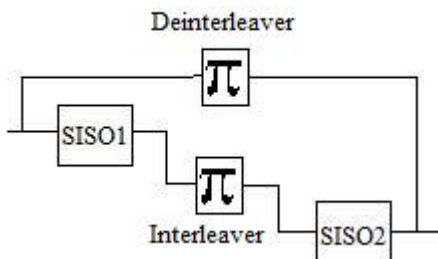


Figure 3: Serial Architecture

turbo decoding can severely reduce the data throughput. A serial turbo decoding architecture is shown in figure 3. The encoded data is passed between the two decoders several times after passing through the deinterleaver/interleaver. Such a serial architecture can considerably reduce the throughput of the system. Consider the case of a turbo decoder architecture synthesized to function at 250MHz, then in order to perform 8 iterations over a data block of 6144 bits, the maximum throughput is upper bounded by $6144 \times 8 \times 4 \text{ ns} = 31 \text{ Mbps}$. The data rates for 4G wireless communication standards can be as high as 100-300Mbps.

3 IMPLEMENTATION

3.1 Parallelization

Since multiple iterations through SISO decoders are required in turbo decoding before decoded output becomes available, decoding throughput becomes a primary bottleneck necessitating a parallel approach to decoding. This parallelization is achieved by allowing several processing elements (PE) to traverse the state trellis simultaneously by dividing the trellis into different parts. A block diagram of the turbo decoder is shown in the figure ???. For a block length of 2400, and 8 PEs, the 2400 time steps of the trellis are broken to 8 trellises of 300 time steps each and each PE computes separate sets of LLRs. The number of PEs is parametrizable and can be 2, 4 or 8 corresponding to a throughput increase of 2X, 4X and 8X respectively. Internally, each PE implements the

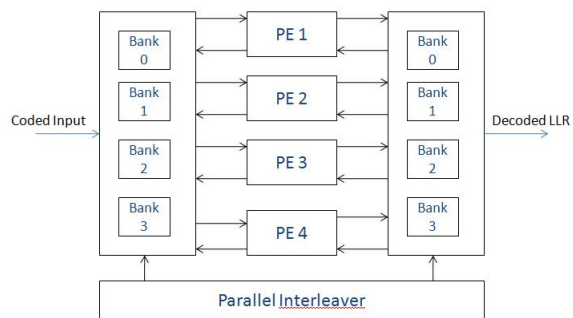


Figure 4: Top level Turbo decoder architecture for 4 PEs

log-MAP algorithm and is made up of gamma memory, alpha memory, branch metric computation unit (BMU), forward recursion units (alpha) and backward recursion units (beta) as shown in the figure 5. The alpha and beta units implement (2) and (3) respectively. As there are four possible paths leading into each state in the trellis, both alpha and beta units need to compute four metrics and select the best one from them. The LLR unit implements (4) for all four possible combinations of input. It requires computation of metrics for all 8 states for all four transitions and selecting the best metric for each of those transitions (LLR00, LLR01, LLR10, LLR11) and eventually determine the largest of the four metrics and subtracting it from all others in order to scale the LLR values. Since the LLR unit performs a large number of computations, it is pipelined into

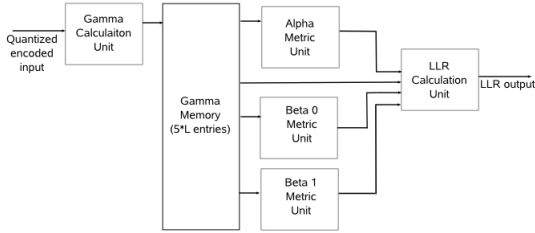


Figure 5: Internal PE architecture

4 stages. The deep pipelining removes the critical path of the design from the LLR unit to the traditional add-compare-select units α and β .

3.2 MAP memory reduction

In order to avoid large memories to support large block sizes, the MAP decoders implement the sliding window MAP algorithm. The algorithm breaks the trellis into smaller trellises and decodes them serially. Thus, the complete trellis is first broken up into different parts and each part is assigned to a different PE. Each assigned trellis is decoded window by window, where the length of the window is changeable. For a window length L , initially the BMU continuously computes the branch metrics at each time step, however the alpha and beta 1 and beta 2 units remain idle. After $2*L$ timesteps, the alpha unit starts processing the trellis from timestep 0 in the forward direction while the beta 1 starts backward recursion on the trellis from timestep $2*L$. After $3*L$ timesteps, beta 1 has computed beta metrics for time step L while alpha unit has calculated forward recursion metrics for all time steps up to L . From timestep $3*L$ to $4*L$, the beta metrics from beta 1 are forwarded to the LLR unit along with the alpha and gamma metrics for corresponding time step to calculate LLR values. Thus LLR values for timesteps $L-1$ to 0 are calculated during time $3*L$ to $4*L$. Moreover, the beta2 unit also starts processing the trellis backward from the time step $3*L$. From time step $4*L$ to $5*L$ beta metrics from beta 2 are forwarded to the LLR unit along with corresponding LLR and gamma metrics. The LLR unit thus produces the LLR values for bit pairs from time $2*L$ to L . Each of the beta units starts its backward recursion L time steps before its metrics are actually used by the LLR unit to ensure that the beta metrics have gained sufficient confidence level after passing through trellis stages. A graphical description

of the the scheduling of these units is shown in figure 6. As a result of these approaches, the gamma memory size is reduced to $5*L$ entries while the alpha memory size is reduced to $3*L$ entries. The beta units no longer require separate memories and their values are consumed as soon as they are produced. Both

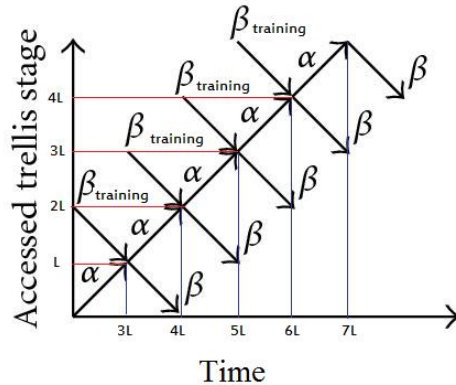


Figure 6: Sliding window MAP scheduling

the gamma and alpha memories are written and read circularly to minimize the size of these memories.

3.3 Metric Normalization

Since the alpha and beta metrics grow at each time step, they can easily overflow. In order to avoid this, the metrics can either be normalized by subtracting the minimum metric from all the states after a few time steps. However, this requires comparators and additional subtractor in the critical path. Since high throughput is one of the constraints of the design, any reduction in the critical path is highly desirable. Consequently, modulo normalization is implemented to minimize the overflow detection and prevention overhead. In modulo normalization, instead of overflow prevention, all the metrics are allowed to overflow but the bitwidths of metrics is adjusted such that the relative difference of the metrics can still be computed. This requires computing the metrics with adders that are two-bits wider than the storage [4] and saving an additional extension bit with each metric. The extension bit determines if the metrics read from the storage should be sign extended or extended by ones. The calculation of this extension bit is based upon the comparison of the 2 MSBs of all the 8 metrics in forward or backward recursions. The two MSBs of all eight α or β metrics are compared to determine the free quadrant. The metrics

are then zero or sign extended according to figure 7

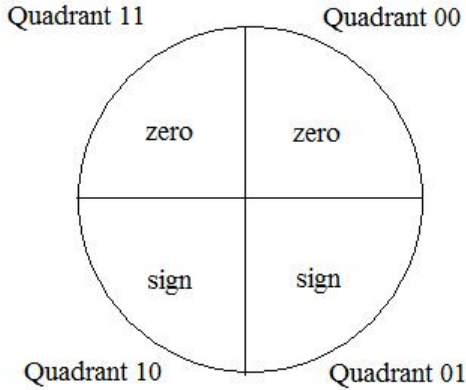


Figure 7: Quadrant based metric normalization

3.4 Interleaver Design

The LLR values computed by PEs are stored at their interleaved/deinterleaved address depending upon whether its an even/odd iteration. The address generation for all the PEs is done by a separate interleaver. However, the design of interleaver (2) and (2) needs to be simplified to avoid multiplications and division operations. One option is to store all the possible interleaving and deinterleaving addresses in a ROM. However, LTE alone supports 40 different block lengths, each with different parameters. The largest of these block sizes is 6144 bits. Which implies storing 6144*2 interleaving and deinterleaving addresses for this one block size alone. Clearly, storing addresses in memory is not a feasible option when low memory is a design constraint. Consequently, the digital filter for interleaving functions defined by (2) and (3) are changed to IIR filters for a simplified serial implementation. A serial implementation of the interleaver in (2) can avoid high complexity by iterative calculation of addresses every clock cycle by adding P_0 to an accumulator every clock cycle and adding the rest of the expression according to the value of modulus 4. In order to support LTE standard, interleaving address generation according to (3) is simplified as follows:

$$P(i) = (f_1i + f_2i^2)_{modN} \quad (9)$$

$$P(i + 1) = (P(i) + f_1 + f_2 + 2f_2i)_{modN} \quad (10)$$

Table 1: Throughput comparison

Iterations	PEs	Parallel throughput(Mbps)	Serial Throughput(Mbps)
2	2	490	243
2	4	909	243
2	8	1666	243
4	2	245	122
4	4	455	122
4	8	833	122
8	2	122	60
8	4	228	60
8	8	417	60

Thus, using an accumulator to shift and add every time step can reduce the complexity of the LTE interleaving. However, for the case of parallel decoding, interleaved addresses need to be generated for all the PEs, which are processing trellis at different locations. Although the addresses required for different PEs are not in a serial manner but they can be generated by determining the bank and bit address of the first PE alone. It is observed that for all the block lengths defined in Wimax and LTE the all the interleaved/deinterleaved addresses have the same bit address as long as the number of PEs is a power of 2. Furthermore, for each block length in Wimax and LTE the bank addresses are permuted in a circular way. Consequently, if the bank address of the first PEs LLR value is known, all the other bank addresses can be determined by circularly shifting and initial vector of bank addresses accordingly. As the interleaver calculates the interleaving addresses it also stores the corresponding deinterleaving bank and bit addresses in memory for using during next iteration. This storage of deinterleaving addresses is also for the bank and bit address of a single PE which can be used to determine the addresses of all the PEs.

3.5 Lookahead Transformation

In order to increase the throughput of the design, lookahead transformation was implemented for the calculation of α and β metrics. For the case of Wimax, there are four possible paths leading into each state before lookahead transform is implemented. However, applying one-level lookahead increases the possible paths from 4 to 16. This requires at least $\log_2(16)=4$ comparators in the critical

path. Consequently the designs operating frequency suffers greatly (300MHz) with an accompanying increase in area due to addition logic. For LTE, however, the prospect of lookahead transformation looks more promising, particularly since it will also enable increased hardware sharing between Wimax and LTE as Wimax already has four input paths per stage while LTE after lookahead transformation will have four paths, allowing an increased throughput for LTE without any additional cost. However, it requires modifications in the LLR values calculation and storage as each bit's LLR needs to be stored at a different interleaved address and applying lookahead transform will produce one LLR for a pair of bits. Due to time constraints, the algorithm transformation required to make lookahead transform feasible for LTE on the combined hardware could not be completed.

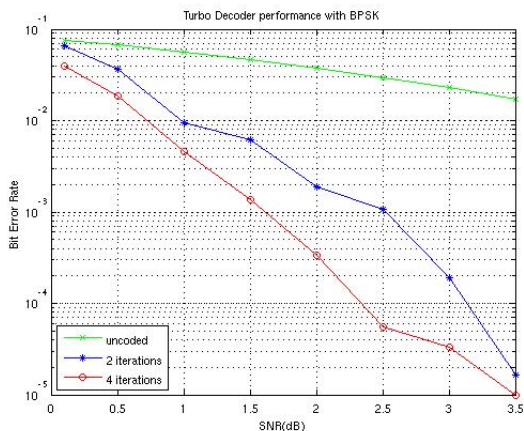


Figure 8: Comparison of implemented turbo decoder with uncoded

Table 2: Synthesis Results(Area)

Number of PEs	Map Decoders	Interleaver	Gamma Memory(bits)
2	439585.2	196341.1	16000
4	640153.4	120888.8	32000
8	1029516.4	57278	64000

4 RESULTS

The design was simulated and tested using Matlab and Modelsim simulations and was synthesized for 500MHz clock using 65nm TSMC technology library.

The throughput comparisons for different number of iterations and PEs are shown in table 1 for 500MHz serial and parallel architectures. The design increases the throughput by 2X, 4X and 8X with as many PEs.

The synthesis results of the design are shown in table 2. The increase in throughput of the decoder by adding more PEs comes at the cost of increased memory requirements for the gamma and alpha memories. However, the size of the interleaver reduces with increasing parallelism as less deinterleaving bit addresses need to be stored (for N block size and n PEs, N/n bit address need to be stored for deinterleaving). Figure (??) shows the bit error rate performance of the decoder for 2 and 4 iterations. As expected, the decoder performs much better than the uncoded BPSK over AWGN channel. Moreover, the performance of the decoder improves with increasing iterations.

5 References

1. IEEE 802.16-2004, IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE, Oct.1, 2004
2. IEEE 802.16-2004, IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE, Oct.1, 2004
3. IEEE Std 802.16e-2005, "IEEE standard for local and metropolitan area networks Part 16: Air interface for fixed broadband wireless access systems Amendment 2: Physical and medium access control layers for combined fixed and mobile operation in licensed bands," Feb. 2006
4. Joakim Björmark, Marco Strandberg, Hardware Accelerator for Duo-binary CTC decoding : Algorithm Selection, HW/SW Partitioning and FPGA Implementation, November 2006