

ECE 734 Project Proposal
Syed Gilani

Implementation of a parallel turbo decoder for IEEE 802.16e (Wimax) standard.

1. Turbo codes introduction:

Turbo coding is used as a channel coding method in almost all the current wireless communication standards. Turbo codes are basically convolutional codes concatenated in series or in parallel. The IEEE802.16e uses a parallel concatenated convolutional code for turbo coding. Although, turbo codes are built by small constraint length convolutional codes, they can achieve error performances much better than standalone convolutional coding. The increase in bit-error-rate performance stems from the fact that the same data is encoded twice with different bit arrangements. This rearrangement of bits to re-encode the data is according to a pre-defined permutation pattern and helps to make the original and rearranged bitstreams independent. The resulting coded data becomes more resilient to channel fading as the receiver's decision for any given bit takes into account both the interleaved and non-interleaved versions of the received bits. The IEEE 802.16e standard uses *duo-binary* turbo codes in which a pair of bits is used for encoding in both the regular and interleaved coding iterations. A block diagram of the Wimax turbo encoder is shown in figure 1.

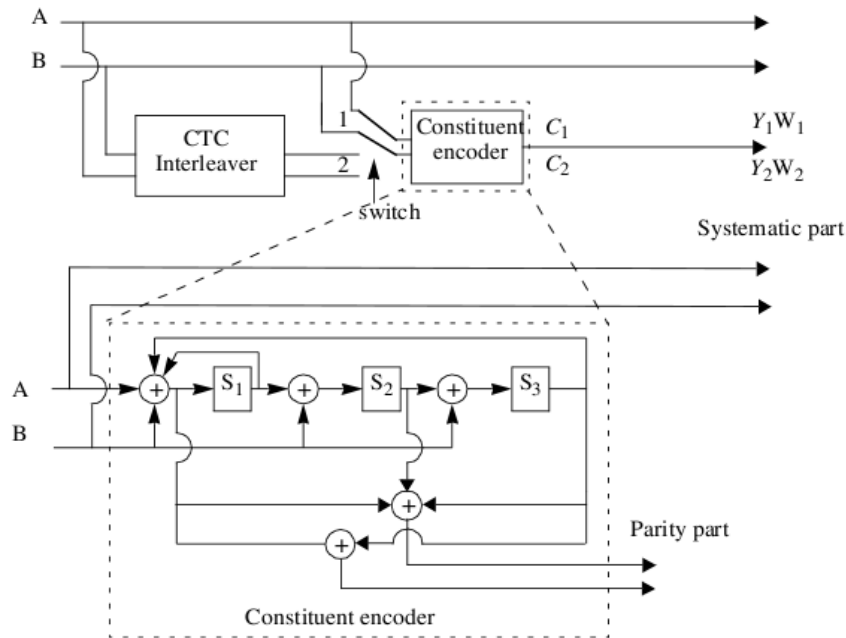


Figure 1: Convolutional turbo encoder for Wimax[3]

2. Algorithm:

Turbo decoding is based upon an iterative *maximum a-posteriori probability* (MAP) algorithm in which the log-likelihood ratios of each bit are calculated in every iteration and forwarded to the next interleaved iteration. The max-log-MAP algorithm is an approximation of the MAP

algorithm that avoids complex logarithmic functions without severe performance degradation [4].

2.a MAP algorithm:

Maximum A posteriori (MAP) algorithm used for decoding turbo coded data determines the following log-likelihood ratio for each bit.

$$\Lambda(c_t) = \log \frac{P_r(c_t=1)}{P_r(c_t=0)}$$

where c_t is the decoded bit at time t given a received sequence r . The algorithm requires a forward and a backward recursion over the state trellis and uses the path metrics from both recursions to determine $\Lambda(c_t)$. For the purpose of explanation, the path metrics for forward recursion at time t are denoted as α_t and the path metrics for backward recursions are denoted as β_t . The branch metric for each transition is denoted as γ_t . A brief summary of the MAP algorithm is given below [2]:

1. Compute the branch metrics for transitions due to inputs 0 and 1 as:

$$\gamma_t^i(l, l') = \begin{cases} p_t(i) \exp\left(-\frac{\sum_{j=0}^{n-1} (r_{(t,j)}^i - x_{(t,j)}^i(l))^2}{2\sigma^2}\right) & \text{for } (l, l') \in B_t^i \\ 0 & \text{otherwise} \end{cases}$$

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 $S_{(t-1)}=l \rightarrow S_{(t)}=l'$ that are caused by the input $i \in \{0, 1\}$. The equation can be simplified by performing these calculations in the logarithmic domain. The subsequent algorithm is called *log-MAP algorithm*. The rest of the steps in the algorithm include:

1. Compute the forward path metrics $\alpha_t(l)$ as:

$$\alpha_t(l) = \max\{\alpha_{(t-1)}(l') + \gamma_t^i(l, l')\} \quad \text{for } 0 \leq l' \leq M_s - 1, i \in \{0, 1\}$$

2. Compute backward path metrics $\beta_t(l)$ as:

$$\beta_t(l) = \max\{\beta_{(t+1)}(l') + \gamma_t^i(l, l')\} \quad \text{for } 0 \leq l' \leq M_s - 1, i \in \{0, 1\}$$

3. The log-likelihood can be computed as:

$$\Lambda(c_t) \approx \max\{\gamma_t^1(l', l) + \alpha_{(t-1)}(l') + \beta_t(l)\} - \max\{\gamma_t^0(l', l) + \alpha_{(t-1)}(l') + \beta_t(l)\}$$

2.b Interleaving address generation:

The interleaving addresses are generated according to the five parameters, P0, P1, P2, P3 and the block length. The parameters P0, P1, P2 and P3 are shown for some block lengths below, for a complete list refer to tables 326 and 327 in [1,3]:

Block Length(bits)	P0	P1	P2	P3
24	5	0	0	0
36	11	18	0	18

48	13	24	0	24
72	11	6	0	6
96	7	48	24	72
108	11	54	56	2
120	13	60	0	60
144	17	74	72	2
180	11	90	0	90
192	11	96	48	144
216	13	108	0	108
240	13	120	60	180

Table 1: Parameters for interleaving

Interleaving is performed in two steps which are different from the data interleaving steps performed outside the turbo encoder.

Step 1:

For an input sequence $u_0 = [(A_0, B_0), (A_1, B_1), (A_2, B_2), (A_3, B_3), \dots, (A_{(n-1)}, B_{(n-1)})]$

for $i=0$ to $n-1$

if $(i \bmod 2 == 1)$ $(A_i, B_i) = (B_i, A_i)$

else $(A_i, B_i) = (A_i, B_i)$

Step 2:

The address of a couple (A_j, B_j) is determined by $P(j)$, which is determined as follows:

switch($j \bmod 4$)

case 0: $P(j) = (P_0 * j + 1)_{(mod N)}$

case 1: $P(j) = (P_0 * j + 1 + N/2 + P_1)_{(mod N)}$

case 2: $P(j) = (P_0 * j + 1 + P_2)_{(mod N)}$

case 3: $P(j) = (P_0 * j + 1 + N/2 + P_3)_{(mod N)}$

3. Bottlenecks:

1. Multiple Iterations

The decoding algorithm is an iterative process which requires several passes of bits before final decoded output is available. A serial implementation can not satisfy the high throughput requirements of Wimax protocol which necessitates a parallel implementation to increase throughput.

2. Memory accesses

For a parallel implementation of the decoder, conflict free memory access is of great importance to ensure that the memories do not require an impractical number of ports. Conflict free access needs to be ensured both when a processing element (PE) is reading log-likelihood ratios of the previous iteration from memory or writing new log likelihood ratios of the current iteration. Both of these memory accesses occur simultaneously and for a parallel implementation several processing elements will be accessing memories simultaneously as well.

3. *Interleaving address generation:*

The interleaving algorithm explained in the previous section, requires complex multiplication and modulus operations. Particularly for a parallel implementation of turbo decoder, it becomes infeasible to have separate multipliers and dividers to generate addresses separately for each processing element (PE). The obvious option of storing all the bit addresses in ROM can result in excessive memory requirements as the the Wimax standard proposes a vast range of block lengths from 24 bits to 2400 bits. Storing interleaving and deinterleaving addresses for all the bits in all the possible block lengths significantly increases the memory requirements of the decoder well-beyond the practical considerations.

4. **Constraints and design methodology:**

For the project I propose to do an ASIC implementation of a parallel turbo decoder for IEEE 802.16e standard. The set of constraints for the decoder design include *high throughput* and *memory reduction*. The parallel turbo decoder will have a variable number of processing elements to achieve different decoding throughputs. The implementation is expected to borrow ideas from *systolic arrays* and employ *look-ahead transformation* and *loop transformations* on the trellis. The interleaver design will explore concepts from *multiplier-less digital filters* to employ a multiplier and divider less interleaver that does not need to store interleaving addresses but generates them on-the-fly.

5. **Results**

The final results will include design's area, throughput comparisons with different number of processing elements (PE), bit-error-rate and SNR graphs and comparisons of memory reduction after employing different techniques.

6. **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) Vucetic, B. and Yuan, J. 2000 *Turbo Codes: Principles and Applications*. Kluwer Academic Publishers.
- 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 Bjärmark, Marco Strandberg, "*Hardware Accelerator for Duo-binary CTC decoding : Algorithm Selection, HW/SW Partitioning and FPGA Implementation*", November 2006