OPTIMIZED MAP TURBO DECODER

Curt Schurgers
EE Department
UCLA
Los Angeles, CA 90024
USA

Francky Catthoor
IMEC/KUL
Kapeldreef 75
3001 Heverlee
Belgium

Marc Engels
IMEC
Kapeldreef 75
3001 Heverlee
Belgium

Abstract - The very powerful turbo codes are the most recent breakthrough in coding theory. The implementation of the iterative turbo decoder however hampers their incorporation in real systems. We tackle this problem by optimizing the decoder on two levels: the separate decoder modules and the global decoder system. Our module level optimizations greatly reduce the decoding delay and energy. The main contribution of this paper is situated at the system level, where we present an optimized architecture that uses a better interleaver organization. The combination of the optimizations on both levels results in a significant decoder power (factor 3), area (factor 2) and latency (factor 2) reduction.

INTRODUCTION

With the recent advent of turbo codes [1], extremely powerful channel coding became available for all kinds of applications. In addition to deep space probes they will probably also be employed in the next generation cellular and satellite systems.

The main obstacle for integrating turbo codes in these systems, is the implementation of the decoder. Instead of pure maximum likelihood decoding an iterative scheme is used, in which information is passed between decoding modules. Two classes of turbo decoding algorithms exist, namely Maximum A Posteriori (MAP) and Soft Output Viterbi Algorithm (SOVA) [2]. We will limit ourselves to the first class since it offers a better performance and has an implementation cost that is assumed to be comparable to SOVA if the sliding window approach is used [3]. Although the MAP algorithm is highly data dominated, its memory optimization has never been treated extensively so far [3],[4]. Also the organization of the decoding modules in the global decoder system has been mostly ignored. Existing solutions use a serial concatenation [5],[6],[7] or a suboptimal feedback loop over the modules [3],[7]. We propose a more efficient structure for the entire turbo decoder by optimizing both the module and system level. Our module level optimizations result in a memory efficient algorithm organization, while the system level approach is based on an intelligent way to implement the interleaving functionality.

We first describe the general principles of turbo coding in the second section, with a focus on the decoding algorithm. Sections three and four are devoted to our module and system level optimizations respectively. The fifth section illustrates the global decoder performance gains of our two level optimization study. Finally, section six concludes this paper.

TURBO CODING

2.1. Encoder

Figure 1a shows the setup of a turbo encoder. This encoder transforms the user bits u_k into the output bits c_n that are transmitted. It consists of a parallel concatenation of relatively simple convolutional component codes (ENC1 and ENC2) separated by a pseudo random interleaver π of size N.

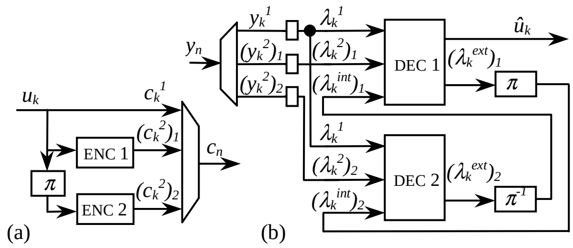

Figure 1: Turbo encoder (a) and decoder (b)

Due to this interleaver, the turbo code operates as a block code. At the beginning of the block, the encoder starts in a known state. In order to force the encoder in a known state at the end of the block, typically some extra tail bits are added [1]. Traditionally, this is only done for the first component encoder ENC1. We add some more tail bits after the interleaver to also have a known end state for the second encoder. These extra bits are not encoded by ENC1, but only by ENC2 and then sent through the channel. Another option to force both encoders in a known end state is to use precursor bits as in [8]. Both schemes reduce the throughput only very slightly, but improve the decoding performance and allow an interesting decoder module optimization as we will explain in section 3.2.

2.2. Decoder

A pure maximum likelihood decoder for a turbo code is too complex to implement. Instead, an increasingly good approximation is obtained with the iterative decoding scheme presented in figure 1b. Each decoder module (DEC1 and DEC2) performs a half iteration. It generates a correction term (called extrinsic information λ_k^{ext}), which the next module uses as a priori information (denoted as intrinsic information λ_k^{int}) after appropriate (de)interleaving.

We have opted for the log-SISO algorithm of Benedetto *et al.* [9] as specific algorithm incorporated in each decoder module. By operating in the logarithmic domain expensive multiplications are avoided. Instead the E-operation is introduced, which boils down to taking the maximum of the operands, corrected with an optional table look up [10]. The received symbols y_k^i are transformed into log-likelihood ratios λ_k^i (for i =1,2), which are defined as:

$$\lambda_k^i \stackrel{\Delta}{=} \log\left[\frac{P(c_k^i = 1 | y_k^i)}{P(c_k^i = 0 | y_k^i)}\right] \tag{1}$$

The decoded bits $\hat{u}_k$ are calculated as:

$$\hat{u}_k = sign\ \left[\lambda_k^{int} + \lambda_k^{ext} + \lambda_k^1\right] \tag{2}$$

With an increasing number of iterations, this solution approaches maximum likelihood decoding more closely. The extrinsic information λ_k^{ext} itself is calculated based on α and β state metrics as indicated in equation (3), in which c^1 and c^2 are the output bits for a component encoder state transition from s to s'.

$$\lambda_k^{ext} = \underset{c^1=1, s\rightarrow s'}{E}\left[\delta_k(s,s')\right] - \underset{c^1=0, s\rightarrow s'}{E}\left[\delta_k(s,s')\right] \tag{3}$$

with $\delta_k(s,s') = \alpha_k(s) + \beta_k(s') + c^1 \cdot \lambda_k^1 + c^2 \cdot \lambda_k^2$

The α and β metrics are obtained through equation (4) and (5), based on a forward and a backward recursion respectively. They are initialized at the beginning (for α) or end (for β) of the block since both component encoders have a known start and end state.

$$\alpha_{k+1}(s') = \underset{s\rightarrow s'}{E}\left[\alpha_k(s) + c^1 \cdot \lambda_k^{int} + c^1 \cdot \lambda_k^1 + c^2 \cdot \lambda_k^2\right] \tag{4}$$

$$\beta_{k-1}(s) = \underset{s\rightarrow s'}{E}\left[\beta_k(s') + c^1 \cdot \lambda_k^{int} + c^1 \cdot \lambda_k^1 + c^2 \cdot \lambda_k^2\right] \tag{5}$$

Once the entire forward α recursion is finished, the first λ^{ext} (at $k = N$) can be calculated. Since the other α metrics are needed for the subsequent λ^{ext} values (decreasing k), all these metrics are stored [4]. This results in a storage of N metrics for all the states, which leads to an unacceptable cost for most practical interleaver sizes ($N = 10^3..10^4$).

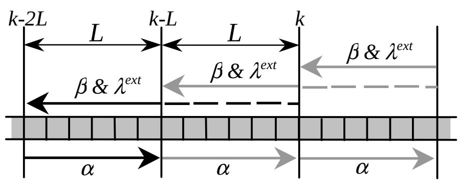

Figure 2: Sliding window SISO decoding algorithm

A solution to this storage problem, is the introduction of sliding windows [9],[10]. The β state metrics are not initialized at the end of the block, but at some point k (see figure 2). After the backward recursion over a window size of L time steps, the metrics provide an accurate approximation at time k-L. The next metrics

β_{k-L} through β_{k-2L} are calculated and used to produce the extrinsic values. The window is then shifted by a value L. This algorithm requires the storage of only L α metrics. To preserve a high throughput, two calculation units for β metrics operate in parallel. This sliding algorithm forms the basis to start our architecture oriented optimizations from.

MODULE LEVEL OPTIMIZATIONS

The first step of our optimization strategy is focused on the module level. Since the decoding algorithm is highly data dominated, a systematic memory optimization is required [11]. An extensive survey of the different transformations we have applied, and their tradeoffs, is presented in another paper [12]. Here, we briefly present the two optimizations with the most impact: double flow structure and partial state metric storage.

3.1. Partial State Metric Storage

As discussed previously, the main motivation to introduce the sliding window approach is the reduction of the state metric storage. We present a way not only to further reduce the size of the state metric memory, but also to limit the number of transfers to this memory since they represent the energy bottleneck in a decoder module (around 70%, see [12]).

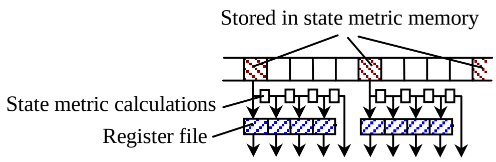

Figure 3: Partial state metric storage

The idea is to store only a fraction $1/\theta$ of the state metrics in a RAM, e.g. α_1, $\alpha_{1+\theta}$, $\alpha_{1+2\theta}$, etc. The missing metrics are recalculated when they are needed. Figure 3 illustrates this principle where the recalculated metrics are temporarily stored close to the data-path in a register file. This decreases the size of the state metric RAM by a factor θ. Also the number of transfers from this RAM is reduced by the same factor as they are replaced by less energy consuming register accesses. The latency and the number of data-path operations, on the other hand, are increased and extra registers need to be provided, leading to a tradeoff.

3.2 Double flow structure

Since the α and β state metrics have a known initial condition for both component codes (see section 2.2), equations 4 and 5 are perfectly equivalent. Decoding can therefore be performed by either applying the sliding window on the β metrics (as is done traditionally [3],[9],[10]) or on the α metrics, both resulting in what we call a single flow structure.

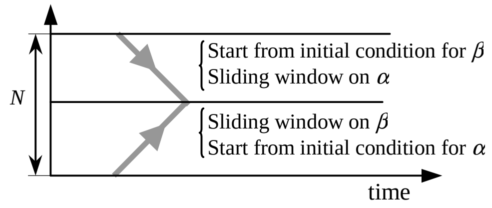

Figure 4: Double flow structure

We introduce yet another option which reduces the decoding time of one data block of size N. For the first half of the data block, the α metrics are calculated starting from their initial condition while a sliding window is applied on the β metrics. For the second half of the data block, it is the other way around. Both calculations can however be started at the same time and run in parallel. This results in the 'meet in the middle structure' of figure 4, which we will refer to as double flow structure. The benefit is a greatly reduced decoding time (almost a factor 2), at a cost of slightly increased hardware resources (details see [12]).

3.3. Results

We have used area/energy models of the different components [11],[12] in order to evaluate the gain of our two module level optimizations. Figures 5 and 6 show the results for a rate ½, 16 state turbo code with an interleaver size N of 5,000 and a window size L of 100.

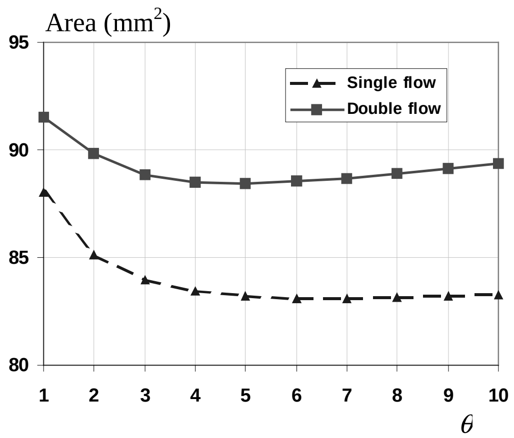

Figure 5: Decoder module area as a function of θ

From figure 5 we conclude that introducing the double flow structure only results in a small area penalty. Changing the parameter θ has no real impact on the area of a module. Figure 6 however clearly indicates that the optimal value of θ (for our parameters and models in the range 3..5) provides a substantial energy reduction for types of flow structures. The normalized throughput exhibits a gain of 85 % when going from a single to a double flow structure, which is close to the expected factor of 2. The decoding delay for one data block is inversely proportional to this throughput. Table 1 compares a double flow structure with $\theta = 4$ to the traditional sliding window organization (single flow and $\theta = 1$, [3],[9],[10]).

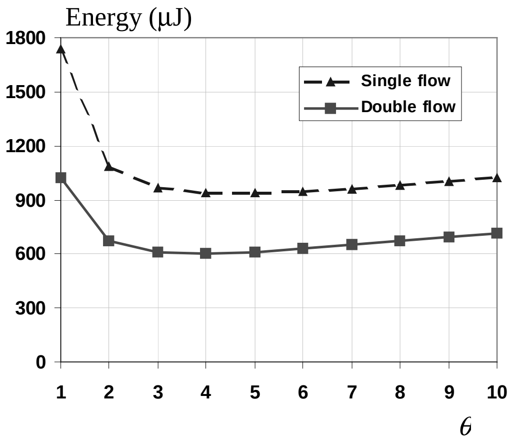

Figure 6: Decoder module energy as a function of θ

Parameter	Multiplied with
Area (A_{module})	1.05
Energy (E_{module})	0.35
Throughput (T_{module})	1.85
Delay (D_{module})	0.55

Table 1: Module level improvements

SYSTEM LEVEL OPTIMIZATIONS

To arrive at a good turbo decoder implementation, not only the separate decoder modules need to be efficient, but also the way they are interacting. This interaction between the modules is mainly dictated by the organization of the interleaving functionality. In this section, we first present an intelligent interleaving setup and expand it towards an efficient decoder architecture.

4.1. Interleaving

As mentioned in section 2.2, the extrinsic information of one decoder module is (de)interleaved (π or π^{-1}) and used as intrinsic information in the next module. A traditional implementation of this interleaving is depicted in figure 7a [3]. The outputs λ_k^{ext} of the first module (corresponding to DEC1 of figure 1b) are written to M1. At the same time the next module (corresponding to DEC2) is working on the previous data block and reads from M2 in the interleaved order. For the next half iteration, the functionality of M1 and M2 is swapped. This setup requires two buffer memories, but the number of transfers is reduced compared to explicitly performing interleaving.

Our improved organization is presented in figure 7b. Once a particular output λ_k^{ext} is generated, the corresponding input λ_k^{int} is no longer needed and its memory location can therefore be reused. The functionality of DEC2 is multiplexed on the same physical hardware module. The interleaving is taken care of by reading the inputs in the interleaved order. The two drawings in figure 7b therefore represent the same piece of hardware for two different time instances (once with the functionality of DEC1, once with that of DEC2). The next half iteration uses the same hardware module as well and reads the inputs in the normal order again. In this way, all the iterations (each split in 2 halves) on one data block are performed on the same hardware module. Our setup reduces the required interleaving memory by a factor two compared to the traditional approach. By appropriately scheduling the read/write accesses, this interleaver memory can still remain a single port RAM, which is crucial for power and area reasons.

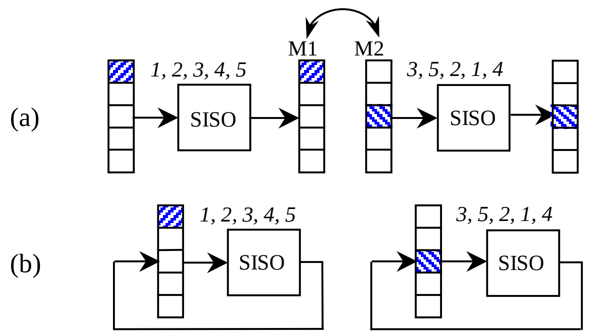

Figure 7: Original (a) and improved (b) Organization of the interleaver functionality

4.2. Parallel Decoder Architecture

Our interleaver setup results in a structure where one decoder module performs all the iterations on a specific data block. To increase the throughput of the decoder, a number (called *h*) of these modules can be placed in parallel as shown in figure 8. The different modules are decoding subsequent data blocks and the λ^{int} memories take care of (de)interleaving the extrinsic information. The received symbols are transformed into log likelihood ratios and stored in one of the input memories (λ^1,λ^2). One more such input memory is needed compared to the number of decoder modules, since the incoming symbols need to be buffered when all the modules are still busy (meaning they still need the stored λ^1 and λ^2 values). The switch provides the appropriate inputs to each decoder module.

Our interleaving setup reduces the size of the λ^{int} memory size by a factor two. For a turbo code with a minimum rate of 1/3, the input memory (λ^1,λ^2) is approximately three times the size of that for λ^{int}. Our architecture therefore reduces the total interleaver memory area by about 13%. This boils down to an area reduction of about 8% (the interleavers consume approximately 60% of the total area [12]).

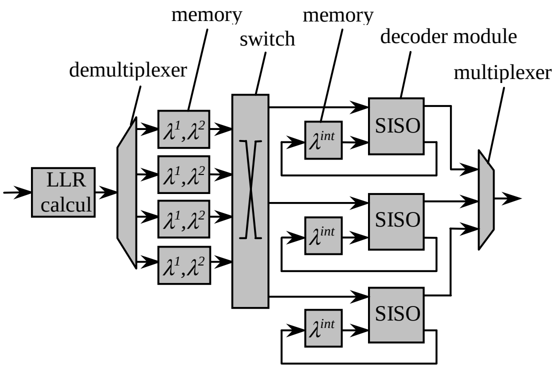

Figure 8: Decoder system level architecture

Next to the efficient interleaving, our architecture also has another advantage compared to a serial concatenation of decoder modules. When all the decoder modules can not be placed on the same piece of silicon, λ^1, λ^2 and λ^{int} need to be passed between two chips in the serial setup. This results in energy-expensive extra off chip transfers, which are completely avoided in our architecture (at a cost of an extra output calculation unit per chip).

PERFORMANCE GAINS AND TRADEOFFS

The combination of the module and system level optimizations of the two previous sections results in an improved turbo decoder compared to traditional organizations. In this section, we evaluate these improvements by expressing them in terms of system level performance measures.

The relevant performance measures are the user data rate R_{tot}, the total decoding power P_{tot}, the total decoder area A_{tot}, and the decoding latency for one data block D_{tot}. Equations 5 through 8 describe them as a function of the module level parameters of table 1 and the decoder settings. These settings are the number of decoding modules in parallel h, the number of half iterations i and the block size N.

$$R_{tot} = \frac{h}{i} \cdot T_{module} \tag{5}$$

$$P_{tot} = i \cdot \frac{E_{module}}{N} \cdot R_{tot} \tag{6}$$

$$A_{tot} = h \cdot A_{module} \tag{7}$$

$$D_{tot} = i \cdot D_{module} \tag{8}$$

Depending on the design goal, our improvements can easily lead to a reduction of h compared to traditional implementations. We therefore like to eliminate this parameter from the above equations, in order to be able to make a fair comparison.

The other parameters (N and i) are determined by the maximum desired decoding performance and are therefore assumed to be fixed. Substituting h of equation (5) in (7), results in (10) and equation (6) can be rewritten to (9).

$$\frac{P_{tot}}{R_{tot}} = \frac{i}{N} \cdot E_{module} \quad (9)$$

$$\frac{A_{tot}}{R_{tot}} = i \cdot \frac{A_{module}}{T_{module}} \quad (10)$$

Equations (8), (9) and (10) relate the module level parameters directly to the system level parameters, only as a function of the desired decoding performance. When choosing the same settings that led to table 1 (double flow structure with $\theta = 4$) and taking into account the extra 8% area reduction due to the interleaver, the system level performance increase compared to the traditional sliding window implementation is indicated in table 2.

Parameter	Decrease with factor
P_{tot}/R_{tot}	2.85
A_{tot}/R_{tot}	1.90
D_{tot}	1.85

Table 2: Decoder performance improvements

Different tradeoffs can be considered depending on which parameter is kept fixed. In the common case of a fixed data rate corresponding to a particular application, the power, area and latency reduction are directly given by the numbers in table 2. From equation (7) it can be derived that this means that 1.8 times less modules are required.

CONCLUSIONS

In order to find an efficient turbo decoder implementation, we have followed a two step optimization strategy. On the module level, we introduce a double flow structure and partial metric storage. This results in a new sliding window MAP algorithm organization, which greatly reduces the decoding delay and the energy consumption with only a small area penalty. On the system level, we have presented a parallel ordering of decoder modules, which limits the interleaver memory and potentially also the number of costly off chip transfers. The combination of these module and system level optimizations results in considerable decoder efficiency improvements. For a constant user data rate, the decoding power, total area and latency are reduced by a factor 2.8, 1.9 and 1.8 respectively compared to the traditional sliding window turbo decoder. Furthermore, it has to be stressed that all our optimizations do not make any assumption about the target platform of the decoder arithmetic. Our novel decoder architecture is therefore generic as it can be targeted towards custom hardware or embedded software (on any type of processor, i.e. RISC, DSP or VLIW).

REFERENCES

[1] C. Berrou, A. Glavieux, P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” *Proc. International Conference on Communications (ICC’93)*, Geneva, Switzerland, May 1993, pp. 1064-1070.

[2] P. Robertson, E. Villebrun, P. Hoeher, “A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain”, *Proc. International Conference on Communications (ICC’95),* Seattle, June 1995, pp. 1009-1013.

[3] S. Halter, M. Öberg, P.M. Chau, P.H. Siegel, “Reconfigurable Signal Processor for Channel Coding & Decoding in Low SNR Wireless Communications”, *Proc. IEEE Workshop on Signal Processing Systems (SiPS’98),* Cambridge, Massachusetts, Oct. 1998, pp. 260-274.

[4] S. Hong, J. Yi, W.E. Stark, “VLSI Design and Implementation of Low-Complexity Adaptive Turbo-Code Encoder and Decoder for Wireless Mobile Communication Applications”, *Proc. IEEE Workshop on Signal Processing Systems (SiPS’98),* Cambridge, Massachussetts, Oct. 1998, pp. 233-242.

[5] C. Berrou, P. Combelles, P. Pénard, B. Talibart, “An IC for turbo-codes encoding and decoding”, *Proc. IEEE International Solid-State Circuits Conference (ISSCC’95),* San Francisco, California, Feb. 1995, pp. 90-91.

[6] E. Hall, S. Wilson, “Stream-Oriented Turbo Codes”, *48th IEEE Vehicular Technology Conference (VTC’98),* Ottawa, Canada, May 1998, pp. 71-75.

[7] S. Pietrobon, “Implementation and Performance of a Serial MAP Decoder for use in an Iterative Turbo Decoder”, *Int. Symp. on Inform. Theory and its Applications,* Sidney, NSW, Nov. 1994, pp. 1073-1077.

[8] A. Barbulescu, S. Pietrobon, “Interleaver design for turbo codes”, *Electronics Letters*, Vol.30, No.25, Dec. 1994, pp. 2107-2108.

[9] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, “A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes”, *JPL TDA Progress Report 42-127,* Nov. 1996.

[10] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, “Algorithm for continuous decoding of turbo codes”, *Electronics Letters*, Vol.32, No.4, Feb. 1996, pp. 314-315.

[11] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vandecappelle, “Custom Memory Management Methodology -- Exploration of Memory Organisation for Embedded Multimedia System Design”, *Kluwer Academic Publishers,* Boston, 1998.

[12] C. Schurgers, M. Engels, F. Catthoor, “Energy Efficient Data Transfer and Storage Organization for a MAP Turbo Decoder Module”, *1999 International Symposium on Low Power Electronics and Design (ISLPED’99)*, San Diego, California, Aug. 1999.