Energy Efficient Turbo Decoding for 3G Mobile

David Garrett, Bing Xu, Chris Nicol
Bell Labs Research, Lucent Technologies
Sydney, Australia
garrettd@bell-labs.com

ABSTRACT

The requirement of turbo decoding in 3G wireless standards has forced handset designers to consider power consumption issues in their implementations. The phenomenal performance of turbo codes comes at the expense of computation. Primarily this paper looks at methods of substantially reducing the power consumption for the decoding operation, making it feasible to integrate turbo decoders into a low power handset. The techniques presented include early termination of the turbo process, encoding of extrinsic information to reduce the memory size, and disabling portions of the MAP algorithm when the results will not affect the decoded output. The net result of these techniques is almost a 70% reduction in power over a fixed 6 iteration, 8-state baseline turbo decoder at 2 dB of signal to noise ratio (SNR).

Keywords

Turbo coding, low power, early termination, extrinsics.

1. INTRODUCTION

Power consumption in mobile terminals for 3G systems will certainly be driven higher by the requirement for turbo decoding data. Since their introduction in 1993 by Berrou et. al. [1], turbo codes have quickly become mainstream and have been integrated in the 3G wireless standards with several data rates [2]. The turbo algorithm itself gains its performance by blindly throwing computation power at the problem. In order for 3G mobile terminals to provide the energy efficiency that users have come to expect, significant changes to the turbo algorithm are required.

The earliest implementations of turbo decoding algorithms were based on Berrou's work with the soft output Viterbi algorithm (SOVA) as the constituent decoder [6]. Pietrobon further explored implementation issues a turbo decoders with the logMAP algorithm with his VXI chassis test platform [4]. Since then, there have been several other implementations [5][3], but only recently has published work specifically focussed on low power considerations [7][8][9][10].

The paper highlights several key techniques to significantly reduce the energy required for turbo decoding. These techniques attack not only the computational inefficiency of turbo by pruning unnecessary operations, but also by restricting memory power consumption through data encoding and reduced memory accesses. These techniques allow most of the energy to be spent where it provides the biggest gain, in the first couple iterations. Section 2 highlights seven techniques to reduce the power dissipation in the turbo decoding algorithm. Section 3 presents results estimating the power savings due to the techniques. Section 4 presents turbo simulation results to prove that the power savings techniques have little impact on the overall system performance.

2. POWER REDUCTION TECHNIQUES

2.1. Early Termination

The strength of the turbo decoding process is in allowing the two constituent decoders to share information between iterations and reinforce the symbol decision made in each decoder. Consider the wide range of channel conditions that can be experienced in turbo decoding. In many cases only two iterations are sufficient to guarantee excellent coding gain, while particularly difficult channel conditions may require many more iterations. The goal of early termination is to determine when the algorithm has converged in order to stop the excess computations that are contributing little to the final solution.

The first work on early termination stems from Haganuer et. al. definition of the cross-entropy (CE) between the decoders. CE can be used as a method to stop the overall terminations in the turbo algorithm [11]. Shao et. al. used the measure of CE to identify two simple stopping criteria, the sign change ratio (SCR) and the hard decision aided (HDA) technique for early termination [12]. Both of Shao's techniques involved monitoring the bit decisions between iterations and determining if there are any changes. The main problem with both SCR and HDA in terms of power consumption is that they require a memory to store all hard decision bits in order to compare with the next iteration. Another shortcoming of the HDA and SCR algorithms is that they require two iterations of data to compare, the previous and the current set, and thus the turbo decoder must perform an extra iteration to confirm that the bits have stabilized. There have been other papers on early termination, but they have also monitored the decision bits [9].

In this paper we present a different early termination algorithm based on CE call the soft decision extrinsic aided (SDEA) algorithm. Instead of looking for just hard decision bits out the decoder, SDEA simultaneously monitors the log-likelihood ratios (LLR) and the extrinsic information from the decoders to identify when the decisions have stabilized. SDEA is based on the idea that soft information for a particular decoder has been generated through multiple iterations meaning that it inherently contains information from previous iterations. Thus the termination can be determined with a proper analysis of information only from the current iteration.

The way SDEA works is to use two comparators to check the magnitudes of the LLR and extrinsic information against thresholds on a per symbol basis. One counter is needed to record the number of bits in the block where the LLR and extrinsic information are both under their respective thresholds. At the end of the block, the turbo decoding will terminate if the value of the counter is below a threshold. For optimum performance with SDEA the counter threshold will be set to zero, meaning all of the LLR and extrinsic information must be above their thresholds. However, the counter threshold can be adjusted to trade further iteration reduction with a slight decrease in performance. Furthermore, because SDEA does not compare its results with the previous cycle, the turbo process can be terminated by either of the two decoders (potentially saving a half iteration). The advantage of the SDEA scheme is that it uses the instantaneous LLR and extrinsic information from one decoder, and hence no memory is required.

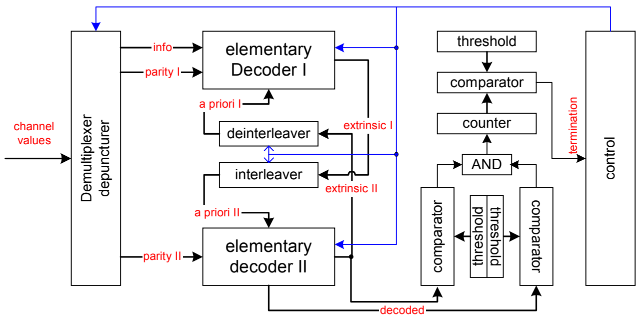

Figure 1 SDEA Architecture

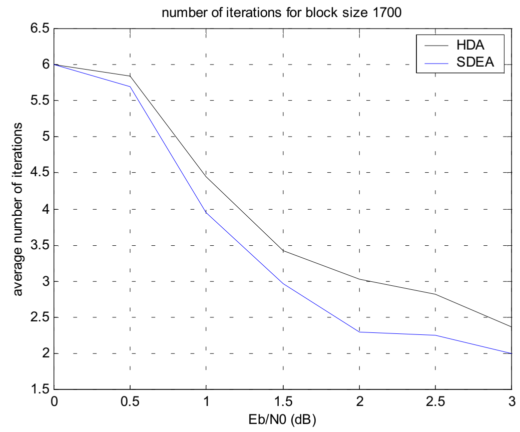

Figure 2 Early termination iterations counts.

Consider the simulation results from a rate 1/3 turbo code system with 8-bit input data, AWGN, and with a block size of 1700. Figure 2 shows a plot of the average number of iterations for the turbo system for HDA and SDEA. On average SDEA results in half an iteration savings across the simulated range of SNR. At 2 dB of SNR, SDEA early termination translates into just over a 60% power savings over a 6 iteration fixed system, almost 10% less than with HDA.

2.2. Quantization of Extrinsics

One of the properties of the logMAP algorithm is that the extrinsic information will continue to grow as each decoder reinforces its bit decision. This is costly in terms of memory size because the interleaver between the decoders must store the entire block of the extrinsic information. For an 8-bit turbo decoding system over 6 iterations, the extrinsic starts at zero but can grow to over 30,000 by the 6th iteration. This requires 16 bits of precision to represent the full range of the extrinsics. Figure 3 shows the extrinsics in the same system where the values have been clamped to 512. By the 3rd iteration, over 85% of the extrinsic values have diverged to the clamping limit. Clamping the extrinsic memory reduces the extrinsic memory width from 16 bits down to 10 bits.

Simulations have revealed no measurable performance degradation due to the clamping. In the logMAP algorithm, the extrinsic information is fed back to the branch metric calculator, which combines the input symbol histories (ISH) with the extrinsic information. With an 8-bit system, the maximum input symbol value is 127. Thus the limit of 512 works because it is sufficiently larger than the input symbol values so that it can dominate the branch metric calculation term.

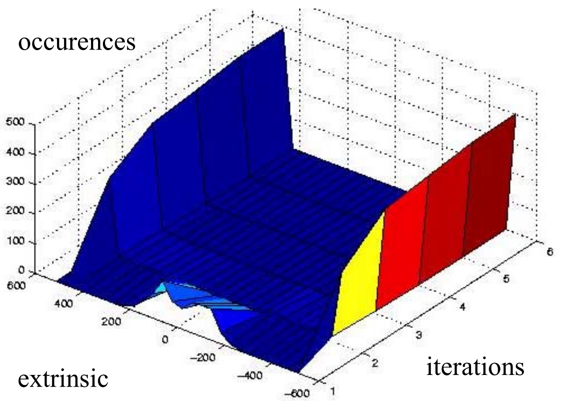

Figure 3 Extrinsics clamped to 512 over 6 Turbo iterations

Clamping the extrinsic can reduce the memory requirement to 10 bits, but there are also further reductions that are possible. When the extrinsic is small, each of the values are important to swing the balance of the turbo operation, but as the extrinsic increases in value, it overwhelms the calculation and only an approximation is needed. Based on this idea, we experimented with encoding the extrinsic data to further reduce the memory requirements of the interleaver.

A simple but effective encoding scheme that accomplishes high precision at small values, and less precision at the larger values is to floor the extrinsic value to the closest power of two number.

With a maximum value of 512, the requantized extrinsic will be an element of the set {0, 2^x} where *x* has a range of [0,9]. With this encoding set, the 10 bit extrinsic can be requantized into a 5-bit sign magnitude number, where the lower 4 bits represent that 11 possible values of the extrinsic from 0 to 512. This provides an extremely simple and fast encoding and decoding circuit, further shrinks the extrinsic memory. Similar work for max-log-MAP was recently published in [14].

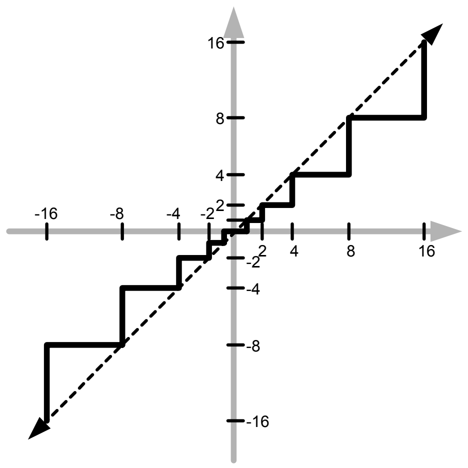

Figure 4 Extrinsic companding function

The companding function has additional benefits not identified in [14] in that in some cases, the encoding provides **better** performance than the regular extrinsic system. Figure 5 demonstrates a complete system simulation of a turbo system with and without the companded extrinsics (block size of 840, 6 iterations, and 8-bit input symbols).

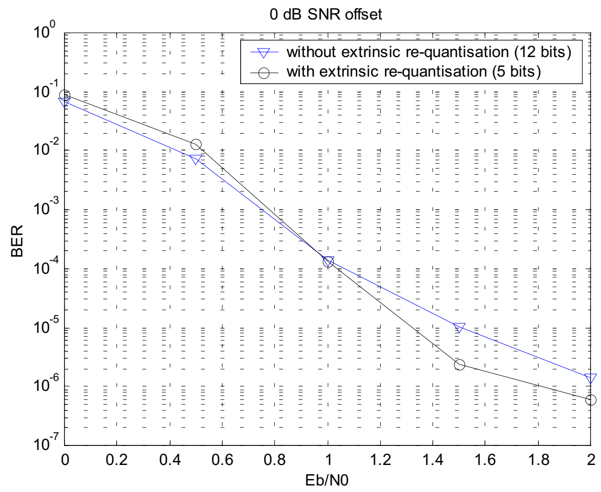

Figure 5 Turbo performance with encoded extrinsics

For SNR higher than 1 dB, the encoding provides an average of 0.2 dB over the conventional system. We hypothesize that this relates the optimality of the extrinsic information. Several papers have discussed scaling of the extrinsics, and have reported better performance in the turbo system by reducing the value of the extrinsic [15]. This seems analogous to any feedback system where reducing the system gain can allow for better performance. Inherently the floor function of our extrinsic encoder provides scaling of the extrinsic value (an extrinsic of 275 would be reduced to 256). Although the scaling is not uniform across the range of extrinsics, it still provides the advantage of extrinsic scaling.

2.3. Optimal Window Sizing

The window approach to implementing logMAP trades the total amount of memory required in the system for extra processing power. The original logMAP algorithm must process the entire forward pathmetrics, α (which requires storing pathmetrics for every state and time slot in the trellis), before computing the backward pathmetrics, β, and the LLR. The window approach takes a small block of information, and the starts the backwards recursion a small distance away from the block in order that the βs have converged to a reasonable distribution for processing on the sub-block (this metric is typically 4-5 times the encoder memory size plus one). Along the same lines, the approach in [7] was to store only incremental portions of the β values to reduce memory cost, and then use logic to regenerate the sub-block β values. The goal for minimum energy in a window logMAP decoder is to maximize the window size (which reduces computation overhead) without increasing the read and write power of the pathmetric memory and other components of the system. Equation (1) represents the average power dissipation for the decode operation, represented by the average energy for each phase, the α calculation, β calculations and the dummy β calculations, multiplied by the clock frequency required to maintain the same throughput as the block logMAP algorithm (no windows or dummy βs).

$$P_{avg}(w) = \left(\frac{E_\alpha(w) + E_\beta(w) + \frac{5(m+1)}{w} E_{dummy\beta}(w)}{2} \right) f_{block} \qquad (1)$$

Consider two important data rates for turbo coded channels in 3G wireless systems, 384 kbs and 2 Mbs. The windowed logMAP algorithm needs approximately 2.15 cycles/bit to process the logMAP algorithm with a window size of 128. With 2 decoders running at 6 iterations, this requires 25.8 cycles/bit for the complete decoder operation. Using this metric along with an overhead of 10%, the system with a window size of 128 must run at 11 MHz and 57 MHz for the 384 kps and 2 Mbs data rates respectively, roughly 8% faster than the non-window logMAP algorithm.

2.4. Branch Metric Cache

The logMAP algorithm is different from the traditional trellis-based convolutional codes in that it performs both a forward and backward recursion over the trellis. Consider the logMAP branch metric equation as seen in equation (2), where *i* represents the path (0 or 1), *d* and *c* are the expected data and parity bits, and y_s, y_p and L_e represent the soft information for data, parity and extrinsic information respectively.

$$\gamma_k^i = d^i\left(y_s + Le\right) + y_p c^{i,k} \tag{2}$$

With a rate ½ trellis code, there are only 4 possible combinations of this equation given the input soft information: (y_s+L_e+y_p), (y_s+L_e-y_p), (-y_s-L_e+y_p), and (-y_s-L_e-y_p). Of the four combinations, two can be generated from the other two.

During the forward recursion stage, both the input symbol and extrinsic memory must be accessed to compute the branch metric, γ. In order to reduce the memory access power, the first two products, (y_s+L_e+y_p) and (y_s+L_e-y_p), can be computed, and then can be written into a local γ memory which stores the values over the window size. When the backward recursion starts, the branch metric can be retrieved from the small local memory, thus preventing a memory access to the two large input symbol history and extrinsic memories. The approach in [5] was to also precompute the γ values, but this was not done in a reduced form. In that design, the same memory bank was re-used as an 8-state pathmetric storage, thereby not realizing the power savings from the small, dedicated γ cache.

2.5. Reduced LLR computation

The extrinsic encoding from Section 2.2 provides an opportunity for further power reduction. It was observed that as the turbo iterations progress, the extrinsics do not decrease once they have reached their highest clamped value. Therefore once the individual extrinsics have reached the clamped value, there is no reason to keep updating their values in memory because the new extrinsics will also be the clamped value. The output extrinsic is computed by subtracting the input extrinsic from the output LLR for each bit in the block. Unless the LLR information is needed outside of the turbo decoding block, it does not need to be computed for the extrinsics that are already clamped. Based on the clamping, the decoder can reduce power through two mechanisms:

- If the extrinsic is clamped still calculate the forward α trellis computation, but no longer store the α results for that particular bit.
- Compute the backward recursion for β, and whenever the corresponding extrinsic is clamped, disable computation of LLR.

Although the local pathmetric memory only has a depth equal to the window size, it has a wide input word in order to store the 8-states simultaneously, so this is effective in reducing the write power. The LLR calculation uses two sets of logsum trees to compute the LLR0 and LLR1 results and disabling them results in further savings in the logic power. Table 1 shows the percentage of extrinsic that were clamped in the turbo system (rate 1/3, with extrinsic companding) on a per iteration basis. In the later iterations, most of the pathmetric memory writes and the LLR computations can be disabled.

	Signal to noise ratio				
Iterations	0.0 dB	0.5 dB	1.0 dB	1.5 dB	2.0 dB
1	0.000	0.000	0.000	0.000	0.002
2	0.000	0.002	0.008	0.073	0.318
3	0.000	0.011	0.089	0.552	0.880
4	0.001	0.078	0.370	0.879	0.963
5	0.004	0.234	0.631	0.944	0.970
6	0.005	0.486	0.775	0.953	0.970

Table 1. Percentage of Clamped Extrinsics

2.6. 3GPP Interleaver Address Computation

The standard for turbo decoding for 3GPP systems involves a complex address generation algorithm for interleaving between constituent decoders [2]. The interleaver algorithm is based on a block interleaver with intra-row and inter-row permutations of the patterns based on a set of prime numbers. The most straightforward method of implementing the address interleaving is to generate all the possible addresses, and store the interleaver address table in a memory. The memory requires 5114 by 13 bits. A better solution for low power operation is to build a dedicated address interleaving datapath to generate the addresses on the fly. Instead of storing the entire address list, the interleaving simply needs to store the intra-row and inter-row permutation patterns. The intra-row pattern, *S*, can have up to 256 entries with 8-bit indices, while the inter-row permutation pattern can have up to 20 entries with 5-bit indices. The actual computation logic is minimal compared with the table memories.

Table 2 shows the configuration overhead and the estimated power consumption for the two methods of obtaining interleaver addresses for a 5114 turbo block. The address datapath saves 54% of the power of the full address table, while the overhead needed to update the tables when the block sizes change is just 0.22% of the entire decoder operation.

Interleaver Addressing options	**Configure overhead (% of cycles)**	**Power dissipation (µW/MHz)**
Address table in SRAM	4.16%	60.3
2 register files + MOD operation	0.22%	27.5(-54%)

Table 2. Address interleaving overhead and power

In the second decoder, the interleaved address is used in two places, first to retrieve the second interleaved parity from the input symbol history, and then to store the new extrinsic back into the de-interleaved extrinsic memory. In order to further reduce power, the extrinsic addresses can be stored in a local cache once they are calculated for this first operation. Then, instead of recomputing the addresses to write the extrinsic information back into the block, they can be retrieved in reverse order from the local cache. The cache is only the size of the window, so its power consumption will be even lower than the interleaver address datapath unit.

2.7. LogMAP Correction Factor

The logMAP correction factor is important is recovering some of the performance loss due to calculating numbers in the logarithm domain [16]. The difference between max-logMAP and logMAP is the addition of an SNR dependent correction factor for the max operation in the add-compare-select (ACS) circuit. We have found that a single value (scaled based on SNR) is sufficient in recovering a significant portion of the performance loss of max-logMAP and is within 0.05 dB of a full 8-entry logMAP correction table. The logMAP ACS is modified to compare the absolute value of the pathmetric difference in order to decide whether or not to inject a single value into the output pathmetric. Figure 6 demonstrates the low power logMAP ACS unit. It saves the logic of a full 8-entry LUT and yet achieves similar performance.

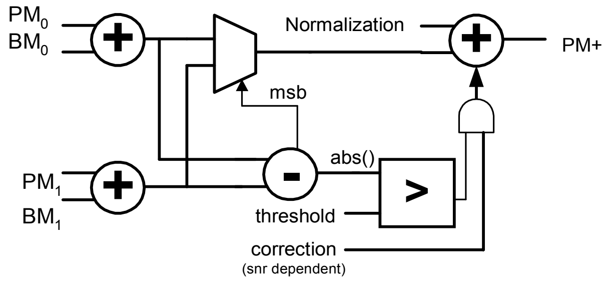

Figure 6 ACS unit with one-value logsum correction

3. POWER ESTIMATES

All of the power savings techniques presented in the previous sections were evaluated by generating a model of the power dissipation for the system using the memory and register file power dissipation numbers for the system, along with an estimate of the average power dissipation for the logic [13]. For the datapath units, the power was estimated by synthesizing the basic cells and then using the cell count and average gate capacitance to extrapolate to the full 8-state ACS unit and logsum trees power dissipation. The average signal activity for the logic was estimated to be 0.20.

Once the parameterized power model was in place, the optimal window size for the window logMAP algorithm was determined by balancing the size of the memories with the computational overhead as seen in equation (1). For this particular system, the optimal window size was found to be 128. Thus, the actual sizes of the memories were set to handle the maximum block size of 5114 and the window size of 128.

Figure 7 shows the overall effectiveness of the power reduction techniques. The first line represents the baseline system with full size memories for extrinsics and ISH operating with the window logMAP algorithm, and using a LUT for the interleaver address. The next variant changes the extrinsic encoding from 16 bits to 5 bits, which results is an 7% overall power savings in the decoder. When the branch metric caching is added on top of the extrinsic encoding, this results in a further 11% power reduction. The address interleaver datapath reduces power by another 1%. The next chart represents the power savings due to the stopping the LLR computation (using the SNR of 2 dB data), which increases the overall power reduction in the system to 61% of the original system without early termination. Without even attempting early termination, these techniques significantly reduce system power consumption. Finally, the early termination with SDEA significantly drops the overall system power dissipation down to 33% of the original system. It is interesting to note that by combining early termination with the other techniques provides a 7% further decrease in power consumption over early termination alone (at 2dB SNR). Early termination tends to limit the effectiveness of techniques like the reduced LLR computation which have their highest power savings in the later iterations.

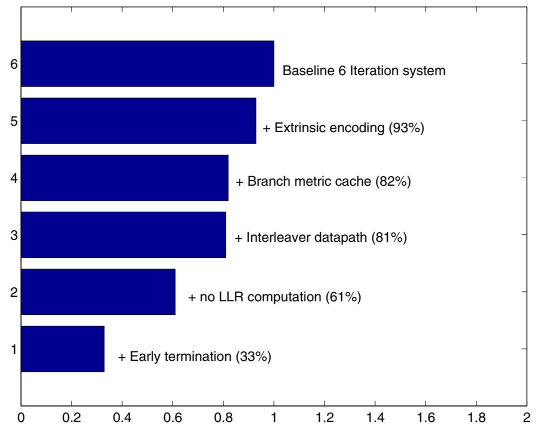

Figure 7 Distribution of power savings

4. SIMULATIONS

While some of the techniques like branch metric caching, and interleaver address calculation have no impact on the algorithm, techniques like clamping the extrinsic, early termination and one-value logMAP correction can affect the system performance. In order to validate the power reduction techniques, all the ideas were incorporated into a system simulation in order to compare with the floating point logMAP algorithm, and the system with an 8-entry LUT. The simulation used a rate 1/3 code with a block size of 840, and 8-bits of input quantization. The simulations use the SDEA early termination method to restrict the number of iterations. As can be seen in Figure 8, there is only a slight overall degradation from the full floating point logMAP performance, even with all the power savings techniques incorporated into the system.

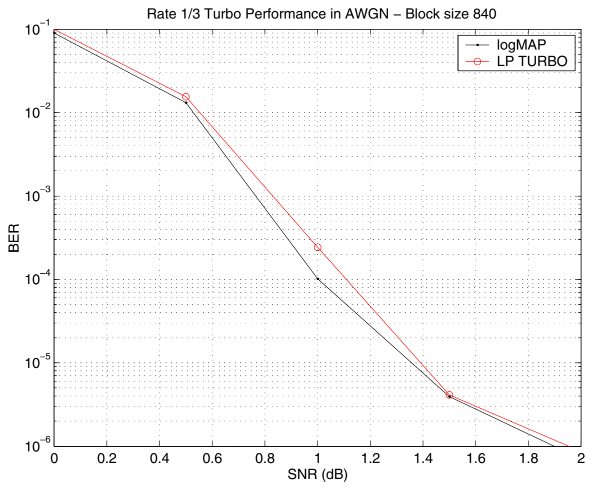

Figure 8 BER performance comparisons among systems.

5. CONCLUSIONS

The turbo decoding algorithm allows significant coding gain in the wireless channel, but at the cost of computation. We have shown that we can achieve significant power savings by both trimming unnecessary computations (early termination using SDEA, no LLR calculation, optimal window sizing), as well as restructuring the actual decoder architecture (γ cache, address interleaving and cache, extrinsic encoding, one-value LUT). The net result of all these techniques is almost a 70% decrease in power consumption for the 6-iteration turbo decoding for 3G wireless data services.

6. ACKNOWLEDGEMENTS

The authors would like to thank Mark Bickerstaff, Linda Davis and Graeme Woodward for all the interesting discussions on turbo coding.

7. REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding,” *International Conference on Communications*, 1993, pp. 1064-1070.

[2] *3rd Generation Partnership Project: Technical Specification Group Radio Access Network: Multiplexing and channel decoding (FDD)*, Technical Specification, Release 1999.

[3] C. Berrou, P. Combelles, and B. Talibart, “An IC for Turbo-Codes Encoding and Decoding,” *IEEE International Solid-State Circuits Conference*, February 1995, pp. 90-91.

[4] S. Pietrobon, “Implementation and Performance of a Turbo/MAP Decoder,” *International Journal of Satellite Communications*, **16**, pp 23-46, 1998.

[5] G. Masera, G. Piccinini, M. Roch, M. Zamboni, “VLSI Architectures for Turbo Codes,” *IEEE Transactions on VLSI Systems*, Vol 7, No. 3, September 1999, pp. 369-379.

[6] M. Jezequel, C. Berrou, C. Dillard, and P. Penard “Characteristics of a Sixteen-state Turbo-Encoder/Decoder (TURBO4),” *International Symposium on Turbo Codes*, 1997, pp. 280-283.

[7] Franky Cathour, “Energy Efficient Data Transfer and Storage Organization for a MAP Turbo Decoder Module”, *International Symposium on Low Power Electronics and Design*, August 1999, pp. 76-81.

[8] D. Garrett, M. Stan, “A 2.5 Mbp/s, 23 mW SOVA Traceback Chip for Turbo Decoding Applications”, *International Symposium on Circuits and Systems*, May 2001.

[9] H. Suzuki, Z. Wang, and K. Parhi, “A K=3, 2Mbps Low Power Turbo Decoder for 3rd Generation W-CDMA Systems,” *Proceedings for Custom Integrated Circuits Conference*, May 2000, pp. 39-42.

[10] O. Leung, C. Yue, C. Tsui, R. Cheng, “Reducing Power Consumption of Turbo Code Decoder Using Adaptive Iteration with Variable Supply Voltage”, *International Symposium on Low Power Electronics and Design*, 1999, pp. 36-41.

[11] J. Haganauer, E. Offer, L. Papke, “Iterative Decoding of Binary Block and Convolutional Codes,” *IEEE Transactions on Information Theory*, Vol. IT-42, March 1996, pp. 429-445.

[12] R. Y. Shao, S. Lin, and M. P. C. Fossorier, “Two Simple Stopping Criteria for Turbo decoding”, *IEEE Trans. Comm.*, vol. COM-47, pp. 1117-1120, Aug. 1999.

[13] *LV160C 1.5V Standard-Cell Library Databook*, Lucent Microelectronics, Sept. 2000.

[14] J. Vogt, J. Ertel, and A. Finger, “Reducing bit width of extrinsic memory in turbo decoder realisations”, Electronic Letters, 28th September 2000, Vol. 36, No. 20.

[15] Z. Wang, H. Suzuki, and K. Parhi, “Efficient Approaches to Improving Performance of a VLSI SOVA-Based Turbo Decoders,” *IEEE International Symposium of Circuits and Systems*, May 2000, pp. I287-I290.

[16] R. Robertson, E. Villebrun, and P. Hoeher, “A Comparison of Optimal and Suboptimal map decoding algorithms operating in the log domain,” *International Conference on Communications*, 1995, pp. 1009-1013.