A Configurable Application Specific Processor for Turbo Decoding*

Pablo Ituero‡ Marisa López-Vallejo‡ Syed Aon Mujtaba†

‡ETSI Telecomunicación (UPM)
Ciudad Universitaria s/n
28040 Madrid, Spain
{pituero,marisa}@die.upm.es

† Wireless Systems Research
Agere Systems
New Jersey, USA
mujtaba@agere.com

Abstract

Turbo codes provide an astonishing performance, however their complex decoder structure entails a power and area consuming VLSI implementation. To overcome this problem we present an application specific processor architecture that clearly outperforms previous implementations. In particular, a simpler normalization scheme for the state metrics is used and a higher degree of concurrency is achieved with little hardware overhead thanks to the optimized use of the butterfly pair structure. Moreover, the resulting architecture is characterized by a great flexibility and programmability —Log-MAP and Max-Log-MAP algorithms, direct procedure and sliding windows mechanism— accomplishing with several industrial standards such as UMTS and CDMA2000 among others. The architecture has been prototyped in a VirtexII FPGA.

1 Introduction

Turbo codes were introduced in 1993 [2] and during the last years the scientific community has made a great effort towards their development and practical implementation. Turbo codes exhibit an astonishing performance and have been adopted in several industrial standards. The main drawback of these codes is the complex decoder structure which entails a power and area consuming VLSI implementation.

Among all algorithms that can compute the Turbo decoding, the MAP algorithm [1] provides the best performance at low E_b/N_0 levels. This algorithm is implemented by means of a Soft-Input Soft-Output (SISO) decoder, whose implementation causes the complexity of the Turbo decoder. Thus, the optimization of area and performance is a foundamental goal in any MAP architecture. Moreover, given that it is used in third generation wireless devices, power optimization becomes a serious constraint.

On the other hand, an additional characteristic that can make more interesting a Turbo decoder is flexibility. A Turbo decoder for cellular phones or satellite communications ideally can execute several standards, is capable to adapt itself to external conditions and can be implemented under different hardware constraints.

In this paper we present an Application Specific Instruction Set Processor (ASIP in the following) architecture that implements a SISO decoder; the main purpose behind it consists on achieving flexibility, low power consumption as well as a significative area reduction. The control of the processor is microprogrammed, allowing a great adaptability to all the parameters of the Turbo code and the decoding procedure. In the program memory several programs can be stored so as to execute different standards under various environmental constraints. A trade-off between accuracy and frequency, as well as between area and execution time can be selected when synthesizing the decoder, this implies adaptability to the hardware constraints. Our approach clearly outperforms previous implementations and is capable to accomplish with several industrial standards —UMTS, CDMA2000, W-CDMA and IEEE 802.16.

Important work has been carried out in the hardware implementation of MAP-based turbo decoders. Recent publications put special emphasis on energy-efficient architectures [9, 6] or on FPGA implementations [11, 7]. Regarding configurability, most approaches are based on the replication of functional units to decode in parallel [3, 11, 4]. To the best of our knowledge, no previous MAP-based turbo-decoder has reached the degree of configurability that is described in this paper.

The structure of this paper is the following: the next section details the theoretical basis of the MAP algorithm; then section 3 approaches its implementation issues; sections 4 and 5 describe the datapath and control of the processor, respectively; section 6 puts forward the synthesis results; and finally we present the conclusions of this work.

2 The MAP algorithm

The MAP algorithm computes the decoding with a very high reliability. The main parameters involved in the algorithm are depicted in figure 1. The

*This work was funded by the CICYT project TIC2003-07036

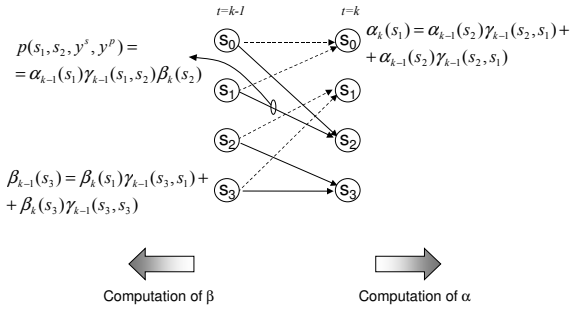

Figure 1: Parameters involved in the decoding process.

transmitted word is represented by u_k. The probability that a transition was produced is given by $p(s', s, y^s, y^p) = \alpha_k(s')\gamma_k(s', s)\beta_{k+1}(s)$, where s' and s are the origin and destination states, respectively; y^s and y^p are the noisy received systematic and parity data respectively; $\alpha_k(s')$ is the forward path metric; $\beta_{k+1}(s)$ is the backward path metric; and $\gamma_k(s', s)$ is the branch metric. A SISO decoder implements this algorithm; it receives the noisy data along with the *a priori* information of the previous decoder L^e_{in}. The SISO decoder yields the *log-likelihood ratio* LLR and the extrinsic information for the next decoder L^e_{out}.

The MAP algorithm can be simplified by working in the logarithmic domain. In this domain multiplications become additions and additions can be computed by means of the *Jacobian logarithm* ($\ln(e^a + e^b) = max\{a, b\} + f(|a - b|)$). If the latter term —known as the correction term— is considered, the algorithm is referred to as the Log-MAP algorithm; in contrast if the correction term is neglected, the algorithm is referred to as the Max-Log-MAP algorithm [10]. In the logarithm domain, the metrics are calculated by the following expressions[1]:

$$\overline{\gamma}_k(s', s) = \frac{1}{2}u_k\left(L^e_{in}(u_k) + L_c y^s_k\right) + \frac{1}{2}L_c y^p_k x^p_k \quad (1)$$

$$\overline{\alpha}_k(s) = \underset{s' \in S_{k-1}}{max^*}\{\overline{\alpha}_{k-1}(s') + \overline{\gamma}_{k-1}(s', s)\} \quad (2)$$

$$\overline{\beta}_{k-1}(s') = \underset{s \in S_k}{max^*}\{\overline{\beta}_k(s) + \overline{\gamma}_{k-1}(s', s)\} \quad (3)$$

$$LLR(u_k) = \underset{S^+}{max^*}\left\{\overline{\alpha}_k(s') + \overline{\gamma}_k(s', s) + \overline{\beta}_{k+1}(s)\right\} - \underset{S^-}{max^*}\left\{\overline{\alpha}_k(s') + \overline{\gamma}_k(s', s) + \overline{\beta}_{k+1}(s)\right\} \quad (4)$$

$$L^e_{out}(u_k) = LLR(u_k) - L_C y^s_k - L^e_{in}(u_k) \quad (5)$$

These equations are the base of the architecture described in this paper. There are several ways to execute the algorithm which entails a trade-off between area and latency. A straightforward procedure computes the alphas in the forward recursion and the betas, the LLR and the L^e_{out} in the backward recursion; this procedure achieves a low total execution time, however it requires a whole block of memory to store the alphas and presents a high latency. To overcome this problem some mechanisms have been proposed such as the *Sliding Windows Mechanism* [12] that divides the computation into small blocks which reduces significantly the storage needs and the latency.

[1] To simplify the equations, $\ln x$ is denoted by $\overline{x}$. max^* stands for the Jacobian expression $max\{a, b\} + f(|a - b|)$.

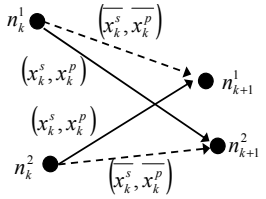

Figure 2: A Butterfly pair.

3 Implementation issues

The idea of an ASIP for Turbo decoding was first introduced in [8]. The novelty of our work is based on three improvements: (a) a simpler normalization scheme for the state metrics is used which allows greater flexibility, (b) a higher degree of concurrency is achieved with little hardware overhead thanks to the optimized use of the butterfly pair structure and (c) both Log-MAP and Max-Log-MAP are implemented. Next paragraphs detail these assets:

The state metrics get bigger and bigger as the decoding algorithm proceeds, to avoid this explosion some normalization scheme must be adopted. In [8] a very complex scheme was employed which meant dealing with delayed un-normalized metrics; this lessened greatly the flexibility of the system. In [13] a new normalization scheme was proposed: at each time instant, all the alphas or betas are compared with 2^{q-2} —being q the number of bits of the state metrics—, if any of them is greater than 2^{q-2}, all the alphas or betas are subtracted 2^{q-2}, otherwise they remain untouched. This method was chosen for this work, it simplifies importantly the normalization since just a one-bit flag must be stored along with the state metrics of a time instant to express the normalization state, furthermore it is very efficient since simple combinational logic can be used to implement it.

As stated in [14] "good" RSC encoders for Turbo codes generate a trellis which can be grouped into 2^{m-1} butterfly pairs, each of them determined by a unique substate —m denotes the memory of the encoder. A butterfly pair is illustrated in figure 2, where x^s_k and x^p_k represent the systematic and the parity outputs of the encoder, respectively. In a trellis section half of the pairs have codewords $(-1, -1)$ and $(1, 1)$, the other half have codewords $(-1, 1)$ and $(1, -1)$. If the gamma computation of equation 1 is considered, it is clear that for a time instant k, everything remains constant except x^s_k and x^p_k so we can rewrite the

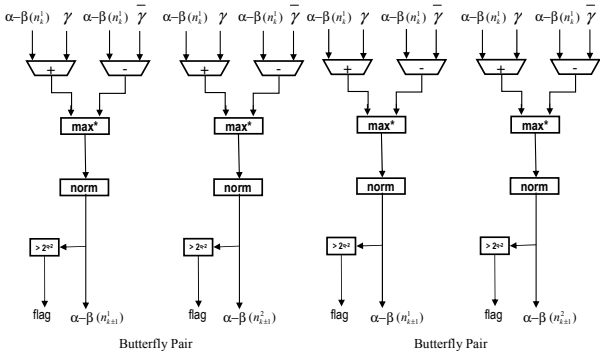

Figure 3: Implementation of the Alpha-Beta Computation.

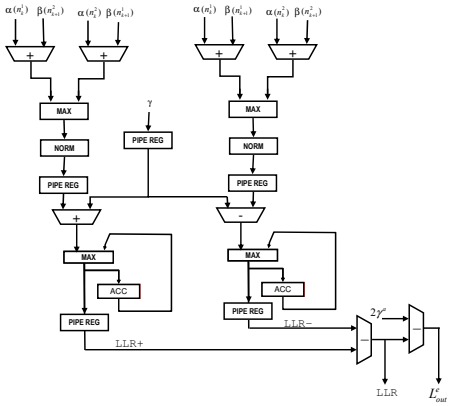

Figure 4: Pipelined implementation of the LLR-L^e_{out} Computation.

equation:

$$\overline{\gamma}_k(s',s) = x^s_k\gamma^a + x^p_k\gamma^b \quad (6)$$

Therefore we have just four possible values for $\overline{\gamma}_k(s',s)$ depending on the codeword. However, taking into account the property of the butterfly pairs mentioned before, in a butterfly pair there are only two values of gamma and they are opposite to each other:

$$\gamma^{even} = (+1)\gamma^a + (+1)\gamma^b = -\{(-1)\gamma^a + (-1)\gamma^b\} \quad (7)$$
$$\gamma^{odd} = (+1)\gamma^a + (-1)\gamma^b = -\{(-1)\gamma^a + (+1)\gamma^b\} \quad (8)$$

This leads to the conclusion that for a Butterfly pair, only a single $\overline{\gamma}_k(s',s)$ computation must be performed since the other one is computed as the opposite to the former. But we could go even further stating that for a whole trellis section there is no need to calculate more that two $\overline{\gamma}_k(s',s)$, namely γ^{even} and γ^{odd}.

Both simplifications of the MAP algorithm —Log-MAP and Max-Log-MAP— are implemented, hence the max^* module can be customized depending on the accuracy and frequency needs.

4 Datapath

All the computations of the system are carried out in the Datapath of the processor, therefore the main design effort was performed on it. This module is responsible for the frequency and latency of the whole processor; moreover, it takes up most of the area and power consumption of the system. Pipelining of each operation was carefully pondered to eventually get a well-balanced structure that achieves a high throughput. 100% hardware resources utilization for every operation represent the ideal goal in an ASIP datapath, in this work we have almost reached that goal (see Results section).

4.1 Alpha and Beta computations

As shown in equations 2 and 3, these two computations perform the same basic operations and thus they will be dealt with as one. In a trellis section there are 2^m equal maximization operations. Taking into account the butterfly property, two butterfly pairs can be computed making use of just five inputs as long as they share $\gamma_k(s',s)$. This is very convenient for low power consumption since it reduces memory accesses and connectivity requirements. A specialized register file, able to deal with the butterfly pairs, is also tailored to achieve low power consumption. Supposing an 8-state Turbo code, this computation attains a throughput of 2 cycles/u_k.

Since serialization is assumed, we have to slightly modify the normalization scheme proposed in [13]. In this work, $> 2^{q-2}$ is calculated throughout several cycles so it is impossible to decide whether -2^{q-2} should be computed or not until we have finished with all the state metrics. The modification consists on flagging whenever any state metric is greater than 2^{q-2}, this flag is registered and used in the next block of state metrics to decide if *normalization at the input* must be applied. Figure 3 displays the general Alpha-Beta computation structure.

4.2 LLR and L^e_{out} computation

These computations are described by equations 4 and 5. Considering that equally signed transitions in a butterfly pair share the same value of $\overline{\gamma}_k(s',s)$, we can extract this term out of the maximization operand, so it is added afterward. In order to minimize the connectivity, all the operations corresponding to one butterfly are computed in parallel, so for an 8-state Turbo code, it attains a throughput of 4 cycles/u_k. Only five inputs are required. Figure 4 shows our pipelined approach in which the first two stages enclose the same $add\text{-}max^*$ structure that was used in the alpha and beta computations; the third stage copes with the final LLR computation and the L^e_{out} computation. Two normalization modules are needed in this computation since both alpha and beta could be unnormalized.

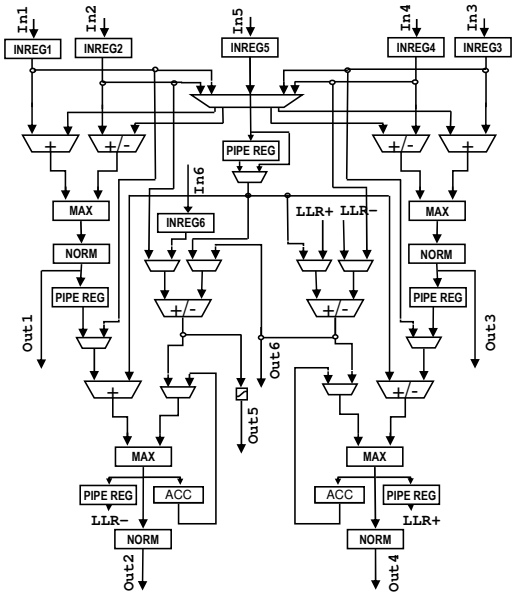

Figure 5: α-β-LLR-L^e_{out} module of the datapath.

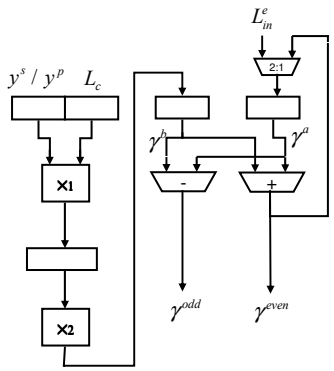

Figure 6: Implementation of the Gamma Computation.

Figure 5 shows the α-β-LLR-L^e_{out} module of the datapath, it is able to perform all the previously described operations. Pipeline registers separate add-max^* structures, which yields a design that is both well-balanced and independent of the max^* module implementation.

4.3 Gamma computation

The gamma operation is described by equation 1. In order to reduce the latency and the storage requirements, this computation is performed in parallel with the rest. Hence a unit of the datapath as well as a part of the register file are exclusively dedicated to this computation. Moreover it has to fulfill the timing of the most restrictive operation in terms of throughput, i.e. 2 cycles/u_k for the alpha and beta computations. Figure 6 illustrates the implementation of this unit, a pipelined multiplier is introduced to reduce the combinational delay between registers. In fact, the critical

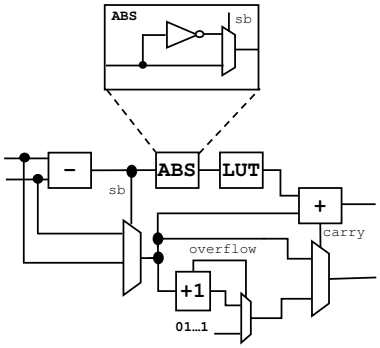

Figure 7: Improved implementation of the max^* module.

path of the processor is found at the first and second stages of the α-β-LLR-L^e_{out} module.

4.4 max^* module optimization

The max^* module corresponds to a maximization operation for the Max-Log-MAP algorithm whereas in the case of the Log-MAP algorithm the correction term was implemented by a LUT. In this latter case, straightforward implementation of the max^* module entails three carry chains which greatly bounds the frequency of the system. In order to reduce the input-to-output delay three improvements were introduced: (a) feed as few bits as possible to the LUT; (b) neglect the +1 chain in the absolute value module; and (c) perform a pseudo *carry-look-ahead* on remaining bits so as to avoid having to wait for the carry chain coming from the LUT. The optimization is shown in figure 7.

5 Control

The control requirements of our system are depicted in figure 8. The global throughput is 8 cycles/u_k. The processor is microcoded and an assembly language has been developed to ease the programming. Low power consumption is achieved by minimizing the internal storage and the connectivity between functional units and registers as well as by specializing the register file to the needs of the MAP

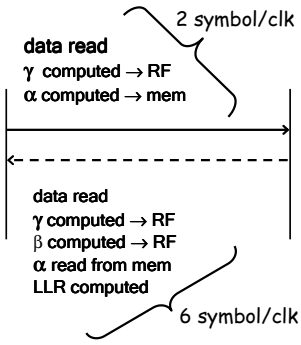

Figure 8: Control requirements.

Table 1: Resource utilization results.

Computation	Resources Util.
Alpha and Beta Computation	89.0%
Alternating Beta-LLR Computation	92.5%
Direct Procedure	91.6%
Sliding Windows Mechanism	91.1%

Table 2: FPGA Implementation Frequency Results.

Algorithm	Max. Frequency	Data Throughput
Log-MAP	55.695 MHz	6.961 MSim/sec
MAX-Log-MAP	70.857 MHz	8.857 MSim/sec

algorithm, reducing memory accesses as much as possible [5]. Only the necessary connections are hardwared in the regfile, nevertheless it is highly flexible as far as Turbo codes are concerned.

The alpha and gamma metrics are calculated during the forward recursion. The alpha metrics are stored in memory; in contrast gamma metrics are consumed *on the fly* after being stored in the register file. In the backward recursion gamma metrics are recalculated, the alpha metrics are fetched from the memory, the beta metrics are calculated and eventually the LLR and the L^e_{out} calculated.

The program memory can contain various programs and thus the processor can execute different procedures —direct procedure and sliding windows mechanism— and standards —UMTS, CMDA2000 and any other 8-state Turbo code— depending on the external requirements.

6 Synthesis results

The Datapath achieves a throughput of 8cycles/symbol and a ratio of utilization of approximately 90% for every computation as shown in table 1, this significantly outperforms previous implementations.

The design was prototyped in a Xilinx VirtexII 4000 FPGA and the synthesis results are displayed by tables 2 and 3. As shown, the SISO decoder takes an extremely reduced percentage of the total FPGA, making this architecture suitable for embedded aplications.

ASIC portability requires few changes mostly referent to just the Regfile module.

Table 3: FPGA Implementation Area Results.

Module	Slices	%	Mems	%
Datapaht	246	1.07%	—	
Regfile	154	0.67%	—	
RAM	21	0.09%	3	2.5%
ROM	21	0.09%	—	
Control	152	0.66%	—	
Total	601	2.61%	3	2.5%

7 Conclusions

We have presented an ASIP architecture that implements a SISO decoder employing the (Max-)Log-MAX algorithm. Butterfly pair properties are extensively used to optimize the datapath. A high concurrency level is achieved, this reduces significantly the total execution time. The system is characterized by a great flexibility that allows to execute several standards and decoding procedures depending on the external requirements. Moreover the architecture —memory size and max^* module implementation— can be adapted to the hardware constraints considering the available area and technology.

References

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. on Information Theory*, pages 284–287, March 1974.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. *IEEE ICC*, May 1993.

[3] Nur Engin. A Turbo Decoder Architecture with Scaleble Parallelism. In *IEEE Workshop on Signal Processing Systems*, pages 298–303, 2004.

[4] G.Prescher, T. Gemmeke, and T.G. Noll. A Parametrizable Low-Power High-Throughput Turbo-Decoder. In *IEEE ICASSP '05*, pages 25–28, March 2005.

[5] Margarida F. Jacome and Gustavo de Veciana. Design challenges for new application-specific processors. *Design&Test of computers*, 17(2):40–50, Apr-Jun 2000.

[6] J. Kaza and C. Chakrabarti. Design And Implementation Of Low-Energy Turbo Decoders. *IEEE Trans. on VLSI Systems*, 12:968 – 977, Sept. 2004.

[7] J. Liang, R. Tessier, and D. Goeckel. A Dynamically-Reconfigurable, Power-Efficient Turbo Decoder. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 91–100, April 2004.

[8] M. López-Vallejo, A. Mujtaba, and I. Lee. A low-power architecture for maximum a posteriori decoding. In *Proc. 36th Asilomar Conf.*, pages 47–51, 2002.

[9] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni. Architectural Strategies For Low-Power Vlsi Turbo Decoders. *IEEE Trans. on VLSI Systems*, 10:279 – 285, June 2002.

[10] P. Robertson and P. Hoeher. Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding. *European Trans. on Telecommunication*, (8):119–125, Mar-Apr 1997.

[11] M.J. Thul and N. Wehn. Fpga implementation of parallel turbo-decoders. In *IEEE 17th Symposium on Integrated Circuits and Systems Design*, pages 198–203, Sept. 2004.

[12] A. J. Viterbi. An Intuitive Justification and a Simplified Implementation of the Map Decoder for Convolutional Codes. *IEEE Jnl. Selected Areas in Comms*, (2):260–264, 1998.

[13] Z. Wang, H. Suzuki, and K. K. Parhi. VLSI Implementation Issues Of Turbo Decoder Design For Wireless Applications. In *IEEE Workshop on Signal Processing Systems*, pages 503–512, Oct. 1999.

[14] Y. Wu, W. J.Ebel, and B. D. Woerner. Forward computation of backward path metrics for map decoder. *IEEE VTC2000*, 2000.