A 50 Mbit/s Iterative Turbo-Decoder

F. Viglione, G. Masera, G. Piccinini, M. Ruo Roch, M. Zamboni
Dipartimento di Elettronica - Politecnico di Torino - Torino, ITALY

Abstract

Very low bit error rate has become an important constraint in high performance communication systems that operate at very low signal to noise ratios: due to their impressive coding gains, turbo codes have been proposed for several applications, although they suffer a large decoding delay. This paper presents the design of a turbo decoder with high performances in terms of throughput implemented using TSPC (True Single Phase Clocking) logic family. In order to achieve the best compromise between cost (in terms of area) and throughput, several architectural solutions have been analyzed. The whole system and in particular its core, the SISO module, has been verified through VHDL simulations. HSPICE simulations show that the system can operate with a 1 GHz clock and thus it can reach a throughput of 50 Mbit/s.

1. Introduction

Several applications need high coding gains and performances closed to theoretical Shannon's limit in terms of bit error rate. High gains and performances can be achieved using block codes or convolutional codes with long constraint length: the use of these codes often results in very time consuming decoding algorithms. Concatenated codes have the advantage of dividing the encoding/decoding process in several, simpler steps: inside this family of codes, the most important ones are the convolutional concatenated codes, also called "Turbo codes" [1], which have been proved as the most powerful solution for high coding gain applications.

Recently, turbo codes have been proposed for satellite and deep-space communications [2], such as in the ESA's mission Rosetta. Other applications of turbo codes are in wireless communications, such as the third generation of mobile communications (UMTS) and in standard protocols for disk drivers [3].

A concatenated encoder is composed of two or more recursive and systematic convolutional encoders connected in an *encoding network* [4]. Interleaving blocks are placed among single encoders: an interleaver is a memory in which data are read and written in different orders. There are two principal schemes of connection: parallel concatenated convolutional codes (PCCC, the "original" turbo codes) and serially concatenated convolutional codes (SCCC).

PCCC has been used in several cases, while SCCC has been used fewer. However, SCCC have been shown to yield performance comparable, and in some cases superior, to turbo codes [5].

The decoder consists in a network of single decoders correspondent to the encoder network. Each single decoder is a *soft decoder* providing an index of reliability (soft information) of the decoded bits (hard information). The whole soft decoder operates in an iterative way and the decoding process is stopped when the wished level of reliability is reached. Better correction performance are obtained as the iteration number is increased; however a large number of iterations has a negative effect on both decoding speed and latency (it is a fact that the latency of turbo codes is considered unacceptable in some telephony applications).

In this paper we present the VLSI design of a fast SCCC decoder with short decoding delay and high throughput. Several architectures have been analyzed in order to choose the best solution for the decoder; the selected architecture has been implemented making use of TSPC [6] logic library operating at 1 GHz.

2. The SISO algorithm

In the case of concatenated codes, the decoding is split into a number of subproblems equal to the number of the constituent codes. In order to exploit entirely the properties of turbo codes, single decoders must exchange soft information. This soft quantity is the sequence of the distribution of probability, conditioned by the received signal and by the knowledge of the code [7]. A solution to the decoding problem is a modification of the traditional Viterbi's algorithm into the soft output Viterbi algorithm (SOVA) , which generates the needed soft information: this algorithm is a suboptimal solution. A different approach to the problem is represented by the *a posteriori probability* (APP) algorithms: the SISO (Soft Input Soft Output) algorithm [8] and

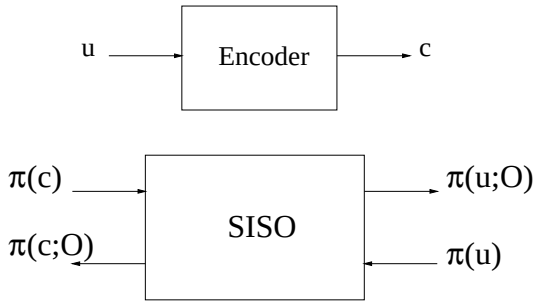

Figure 1. The encoder and the SISO module

its modification implemented in this work belong to this category.

The SISO algorithm is the core of the decoding algorithm, which receives the soft informations related to the decoded bits and refines them iteratively. In the following the SISO algorithm described in [8] (and derived from [9]) will be shortly presented with reference to Figure 1

In the description of the algorithm we will use the following notations. We will indicate the encoder input and output symbols with u and c respectively; y will be the signal received from the demapper, while s_k will indicate the state of the decoder at time k; e will represent the transition between the start state s^S and the end state s^E.

The a posteriori probability can be achieved by

$$P(u|y) = \sum_{S_i} \sigma_k(S_i, u) \tag{1}$$

where σ_k is the a posteriori transition probability

$$\sigma_k \stackrel{\Delta}{=} P(u_k = u, s_{k-1} = S_i|y) \tag{2}$$

as reported in [9]. The SISO routine elaborates the quantities $\pi_k(c;O)$ and $\pi_k(u;O)$, correlated to σ_k, with the following equations:

$$\pi_k(c;O) = \log \sum_{e:c(e)=c} \exp\left\{\alpha_k[s^S] + \pi_k[u] + \right. \\ \left. + \pi_k[c] + \beta_k[s^E]\right\} \tag{3}$$

$$\pi_k(u;O) = \log \sum_{e:u(e)=u} \exp\left\{\alpha_k[s^S] + \pi_k[c] + \right. \\ \left. + \pi_k[u] + \beta_k[s^E]\right\} \tag{4}$$

$$\alpha_k(s) = \log \sum_{e:s^E(e)=s} \exp\left\{\alpha_{k-1}[s^S] + \pi_k[u] + \pi_k[c]\right\} \tag{5}$$

$$\beta_k(s) = \log \sum_{e:s^S(e)=s} \exp\left\{\beta_{k+1}[s^E] + \pi_{k+1}[u] + \right. \\ \left. + \pi_{k+1}[c]\right\} \tag{6}$$

where s^S, s^E, u and c depend on e. The decoder must evaluate expressions like:

$$a = \log\left[\sum_i^L \exp\{a_i\}\right]. \tag{7}$$

This type of relation could be approximated with the maximum of a_i; better performances are obtained with the following routine:

$$\begin{aligned} a^{(1)} &= a_1 \\ a^{(l)} &= \max(a^{(l-1)}, a_l) + \\ &\quad + \log[1 + \exp(-|\, a^{(l-1)} - a_l\,|)] \;\; l = 2 \div L \\ a &\equiv a^{(L)} \end{aligned}$$

The exponential function is mapped to a look-up table.

Equation 6 describes a backward recursion and it implies that all symbols are received before starting the decoding procedure. This is not acceptable in continuous transmissions and then the modification described in [8] has been adopted: the decoder operates on a finite length (NDP) of received samples. The journey metrics β are initialized at step k as

$$\beta_k(s) = constant \quad \forall s \tag{8}$$

and the equation 6 is executed from step k to step $k - NDP$. Simulations have shown that the needed value of NDP is equal to $6 \div 7$ times the logarithm of the number of states N.

3. High speed SISO architectures

The direct implementation in a VLSI architecture of equations 3 to 6 implies the allocation of RAM memories to store the branch metrics (π) and the journey metrics (α and β) and ACS (Add Compare Select) modules to calculate α, β, $\pi_k(c;O)$ and $\pi_k(u;O)$. It is worth noting that equation 5 describes a forward recursion, where new α metrics are evaluated from the previous ones; on the contrary equation 6 works in backward direction so requiring the storage of the received branch metrics for NDP steps. So, while equation 5 implies that each new evaluated α depends on the complete history of the previously received samples, the β metrics has a finite memory of NDP samples.

In order to achieve high throughput values two main implementation problems must be solved.

1. the use of RAM memories for metrics storage must be avoided as the RAM access time tends to be the bottleneck of the decoder;

2. the implementation of equation 5 must be modified as it introduces a feedback loop in the decoding process: if this loop is not broken, any speed-up advantage cannot be obtained by pipelining.

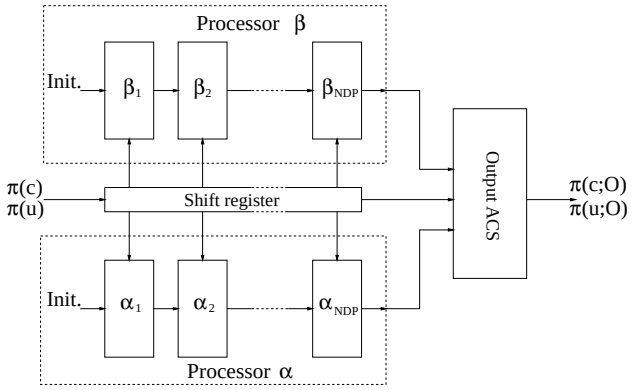

Figure 2. Architecture of the SISO module

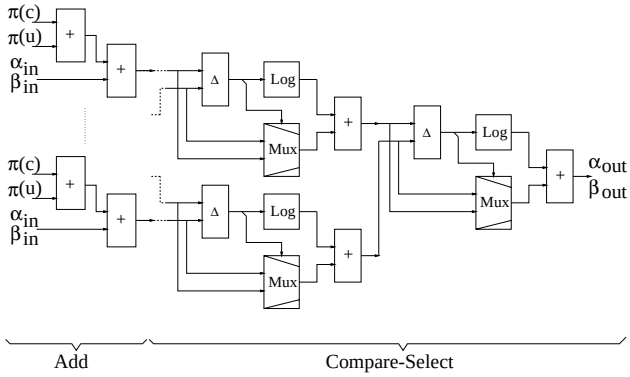

Figure 3. Block diagram of an ACS module

The first problem is solved by replacing memories with shift-register modules, which guarantee higher speed. Concerning the second problem, the algorithm has been slightly modified so that both α and β recursions work on a finite window of NDP samples, starting from the initialization given in equation 8. As consequence of this modification equations 5 and 6 result in a continuous decoding process without feedback loop; the speed of this process can be strongly increased by using pipelining.

The entire SISO architecture is described in Figure 2: it includes a shift-register module and three processors. The first processor elaborates the journey metrics α (*processor* α); the second one calculates the journey metrics β (*processor* β); these two blocks receive the input probabilities $\pi(c)$ and $\pi(u)$ and provide the journey metrics to the last module, that computes the output probabilities $\pi(c;O)$ and $\pi(u;O)$. The shift-register module (SRM) has the function of supplying the branch metrics $\pi(c)$ and $\pi(u)$ to the processing units at the proper times. The length of the shift register is a linear function of the latency of the α and β sections.

The processors α and β are composed of *NDP* identical sections connected in cascade.

Each section of modules α and β is composed of *N* Add-Compare-Select (ACS) blocks, where *N* is the number of states. A block diagram for the ACS units is reported in Figure 3. The number of compare operations is equal to the number of transitions that lead to or originate from a state. The *Add* part of each ACS block includes two adders: the first one sums $\pi(u)$ and $\pi(c)$ while the second one completes the operation adding the journey metric α or β to the first partial result.

The first adder operates on the branch metrics, while the second one adds the journey metric contribution. It is possible to prove that, using a two complement representation for all quantities, overflow problem can be neglected [10], provided that the final result of the Add operation is represented with three additional bits with respect to the branch metrics.

The *Compare* and *Select* operations are achieved from a binary-tree structure: each node of this structure compares only two journey metrics. The compare operation is simply implemented by a subtracter, while the selection operation must evaluate expressions like:

$$a = \max\{a_1, a_2\} + \log[exp(-|a_1 - a_2|)] \qquad (9)$$

The choice of the maximum is executed by a multiplexer driven by the sign of the difference calculated during the compare operation. The logarithmic correction is evaluated by a combinatorial net.

It is evident that the most important constituent element of the ACS blocks, and thus of the entire SISO module, is the adder. Several types have been analyzed and in particular *carry propagate, carry look-ahead, Brent-and-Kung'* and *Sklanski'* adders [11].

The structure of the ACS is strongly dependent on the choice of this component. The carry-propagate adders are very simple, but they require pre-skew and de-skew blocks because they provide the bits of the result with different latencies. These blocks increase the cost of the ACS modules and, because of their high latency, they increase the length (and thus the cost) of the shift-registers. Brent-and-Kung' and Sklanski' adders present an higher latency than carry look-ahead adder; this backdraw is balanced by an higher regularity in their structure and by a lower fan-out.

The equations 5 and 6 imply the sum of the input probabilities $\pi(c)$ and $\pi(u)$ related to the same instant *k* and the same transition *e*: in order to reduce the number of the adders, branch metrics have been added only one time at the input of the SISO module. Furthermore, the cost of SRMs is reduced as the result of this modification, because only one probability is stored instead of two.

A cost analysis of the whole SISO module has been done. It has been performed for different values of the code and architectural parameters: number of states, number of symbols of encoder inputs and outputs, number of

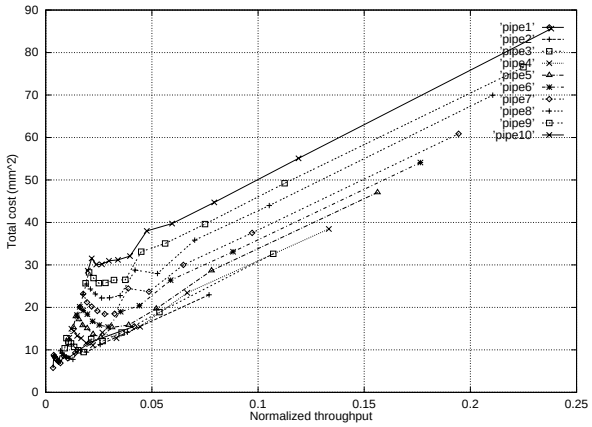

Figure 4. Cost vs Throughput for a code with rate 1/2 and 4 states.

bits for the representation of the metrics, number of levels of pipelining, type of the adders and throughput.

As an example of the performed study, in figure 4 the cost versus throughput is reported for a code with rate 1/2, four states, using 9 and 7 bits to represent respectively journey and branch metrics and using Sklanski' adders. Costs figures are given for a $0.7\mu m$ technology library. The different curves are for different values of the number of pipelining stages; the points along the same curve are related to architectures where reduced numbers of α and β sections are shared among the iterations. The highest throughput is reached with 10 pipelining levels per section and with no sharing of resources.

4. The architecture of the whole decoder

The encoding network is composed of two convolutional encoders with four states and rate 1/2. They are connected in a serial concatenation separated by an interleaver that operates on single bits and not on the entire encoded words. The first encoder (outer encoder) provides two bits per each bit generated by the source: thus the second encoder (inner encoder) must operate at double throughput. A block diagram of the decoder is shown in figure 5.

The decoder includes two SISO modules separated by the interleaver I (identical to that used in the encoding network) and the de-interleaver I^{-1}, which performs the inverse interleaving function. These last two blocks are implemented as external memories. It is worth noting that an interleaver implemented as a single RAM would became the system bottleneck, due to the access time limitation; however a properly designed interleaving permutation can be mapped to more RAM memories to be addressed in parallel, so increasing the overall interleaver speed.

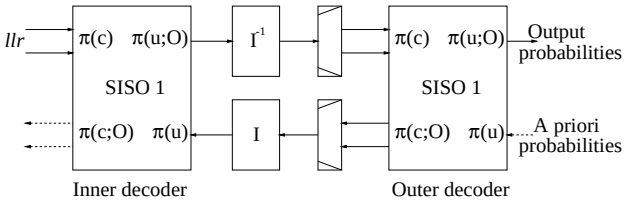

Figure 5. Block diagram of the decoding network

The a priori probabilities indicated in the picture are not usually provided, so that the outer decoder can be simplified; the $\pi(c;O)$ of the inner decoder are not used and thus the correspondent ACS module can be dropped.

Inteleaver and de-interleaver operate on probabilities related to single bits (logarithmic likelihood ratio, llr) while SISO modules process quantities related to the encoded words: interfacing blocks are then needed. To obtain $\pi(c)$ and $\pi(u)$ the following equations have been implemented:

$$\pi(u=0)=0; \qquad \pi(u=1)=llr(u_0)$$
$$\pi(c=00)=0; \qquad \pi(c=01)=llr(c_0)$$
$$\pi(c=10)=llr(c_1); \;\; \pi(c=11)=llr(c_0)+llr(c_1)$$

while the new llr come from:

$$\begin{aligned} llr_{out}(u) &= \pi(u=0)-\pi(u=1)-llr_{in}(u) \\ llr_{out}(c_0) &= max^*\{\pi(c=00),\pi(c=10)\}+ \\ &\quad -max^*\{\pi(c=01),\pi(c=11)\}+ \\ &\quad -llr_{in}(c_0) \\ llr_{out}(c_1) &= max^*\{\pi(c=00),\pi(c=01)\}+ \\ &\quad -max^*\{\pi(c=10),\pi(c=11)\}+ \\ &\quad -llr_{in}(c_1) \end{aligned}$$

Likewise to the encoder, the rate of the inner decoder must be double with respect to the outer decoder: thus the two SISO modules must be implemented with different schemes. In order to achieve the best performance, the inner decoder has been implemented with the full speed architecture, without any processor sharing. Differently, the outer decoder uses a folding technique where the number of sections in the α and β processors has been halved and each processor is shared among two iterations.

5. The VLSI implementation

In order to achieve the maximum throughput, the dynamic logic family TSPC (True Single Phase Clocking) [6]

has been used to implement the decoder. A library composed of required basic cells has been designed and each cell has been described at two levels: electrical level and behavioral level.

Using the $0.5\mu m$ MIETEC technology, the library has been designed and then simulated with HSPICE. After transistor size optimization, all cells have reached the frequency of 1 GHz; the maximum width of the transistors was $1.6\mu m$.

The description at the behavioral level has been done using VHDL; timing constraints have been included using the VITAL standard library, which allows to simulate wire delays and cell intrinsic delays; furthermore it allows to verify that the timing constraints (set-up time, hold time, maximum frequency) have been met.

Concerning the number of quantization bits for the metrics, the standard solution of 4 bits for the soft inputs ($llrs$) has been chosen; with this choice, branch metrics $\pi(c)$ and $\pi(u)$ need 5 bits, while the journey metrics α and β need 8 bits.

From the cost analysis of ACS sections and shift-register modules in the case of TSPC implementation, the Sklanski' adder results the best possible choice: the advantage in terms of transistors is equal to 10% with respect to Brent-and-Kung' adders and 20% with respect to carry look-ahead adders.

Since the synthesis tools do not support dynamic logic libraries like TSPC, this operation has been executed manually. Thus the entire outer decoder architecture has been implemented and a VHDL functional simulation has been run using *Leapfrog*.

The cost of whole decoder, in terms of number of transistors, has been evaluated. The outer and inner decoders need about 235 and 340 thousands transistors respectively, for a global cost of 575,000 transistors. The critical path of the synthesized decoder shows a delay lower than 1 ns.

Assuming a clock frequency f_0 equal to 1 GHz, ten decoding iterations allow a data rate of 50 Mbit/s.

6. Conclusions

In this paper we present the VLSI implementation of a turbo decoder with high performance in terms of throughput. The whole system has been verified by VHDL simulations. A library of TSPC cells has been created and HSPICE simulations show that the decoder is able to operate at the frequency of 1 GHz: this implies that, with 10 iterations, a throughput of 50 Mbit/s can be reached.

References

[1] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes", *Proc. 1993 Inter. Conf. Commun.*, pp. 1064-1070, May 1993.

[2] R. Garello, R. Maggiora, G. Montorsi, P. Coccia, S. Benedetto, A. Serra, "DSP implementation of turbo decoders for satellite communications", *Proceedings of Sixth International Workshop on Digital Signal Processing Techniques for Space Applications, DSP98*, Noordwijk, The Netherlands, September 1998, P.1.

[3] T. Sovigner, A. Friedmann, M. Öberg, P. Siegel, R. E. Swanson, J. K. Wolf, "Turbo codes for PR4: Parallel Versus Serial Concatenation", *accepted to Proc. 1999 Inter. Conf. Commun.*, Vancouver, Canada, June 1999

[4] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, "Soft-input Soft-output modules for the construction and distributed iterative decoding of code networks", *European Transactions on Telecommunications* , vol. 9, No. 2, March-April 1998.

[5] S. Benedetto, G. Montorsi, "Iterative decoding of serially concatenated convolutional codes", *Electronics Letters*, July 1996.

[6] Y. Jiren, I. Karlsson, C. Svensson, "A true single-phase-clock dynamic CMOS circuit technique", *IEEE journal of Solid-State Circuit*, vol. SC-22, pp. 261-266, 1983.

[7] G.D. Forney Jr., "Concatenated Codes", *Massachussetts Institute of Technology*, Cambridge, Massachussetts, 1966.

[8] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, "Soft-output decoding algorithms for continuous decoding of parallel concatenated convolutional codes", *The telecommunication and data acquisition progress report 42-124, October-December 1995*, Jet Propulsion Laboratory, Pasadena, California, pp. 63-87, February 1996.

[9] L.R. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Transaction on Information Theory*, pp. 284-287, March 1974.

[10] G. Masera, G. Piccinini, M. Ruo Roch, M. Zamboni, "VLSI Architecture for Turbo Codes", *IEEE Transactions on VLSI*, September 1999.

[11] M. Shoji, *CMOS digital circuit technology*, Englewood Cliffs, New Jersey, Prence-Hall, 1988.