High throughput implementation of an adaptive serial concatenation turbo decoder

Maurizio Martina, Andrea Molino, Fabrizio Vacca, Guido Masera, and Guido Montorsi

Abstract— **The complete design of a new high throughput adaptive turbo decoder is described. The developed system is programmable in terms of block length, code rate and modulation scheme, which can be dinamically changed from frame to frame, according to varied channel conditions or user requirements. A parallel architecture with 16 concurrent SISOs has been adopted to achieve a decoding throughput as high as 35 Mbit/s with 10 iterations, while error correcting performance are within 1dB from the capacity limit. The whole system, including the iterative decoder itself, de-mapping and de-puncturing units, as well as the input double buffer, has been mapped to a single FPGA device, running at 80 MHz, with a percentage occupation of 54%.**

Index Terms— **turbo codes, channel decoders, parallel architectures, VLSI implementation**

I. INTRODUCTION

AFTER the pioneering work published in 1993 [1] on turbo codes, a large amount of efforts has been spent on the design, evaluation and implementation of concatenated convolutional codes with iterative decoding. Nevertheless turbo codes are still a major subject of interest in the scientific literature and a growing effort is currently being dedicated to the implementation issue.

The availability of deeply scaled CMOS technologies today enables the low cost implementation of high complexity decoding algorithms, such as the BCJR [2] commonly adopted for turbo codes; however two main important requirements of modern and future channel decoders still make the hardware implementation of turbo codes a difficult challenge:

- the quest for extremely high data rates
- the need of incorporating elements of flexibility in the decoder.

The first requirement directely results in the adoption of parallel decoding architectures that significantly impact on the implementation cost and pose additional constraints in the design of the code and especially of the interleaver, as reviewed in section II. The characteristic of flexibility is referred at least to the capability of supporting different block lengths and code rates; this is often coupled to the need of supporting multiple modulation schemes, so demanding flexibility also in the demapping section of the receiver. Versatility always affects hardware cost and energy consumption and as a consequence it must be incorporated with carefulness, in order not to depress the energy efficiency. So new efficient architectures are needed to achieve in turbo decoders both characteristics of high throughput and flexibility.

Particularly, the implementation of high throughput turbo decoders has been addressed in several applicative contexts, including wireless communications [3] [4], satellite applications [5] [6], optical communications [7] and data storage [8]; however it still represents a major challenge to electronic system designers and today the domain of efficient and flexible implementation solutions is deeply investigated by researchers. On the other hand, a few works have been published on the implementation of versatile turbo decoders (see for example [9] [10]).

In this work, the complete design of a new high throughput versatile turbo decoder is described. The addressed code makes use of a serial concatenation scheme; the system flexibility is referred to block length, code rate and modulation scheme. In order to achieve high decoding throughput, a parallel architecture has been designed with 16 SISO units; the implemented system also includes inner and outer de–puncturing sections, a flexible de-mapper and an input double buffer to decouple decoder frequency from the rate of incoming symbols. The whole system has been mapped and tested on a single advanced FPGA device, achieving a 35 Mbit/s throughput with a clock frequency of 80 MHz.

The rest of the paper is organized as followed: after a review of parallel architectures for turbo decoders in section II, the addressed adaptive communication scheme is described in section III, while sections IV and V report implementation details and synthesis results respectively; finally some conclusions are drawn in section VI.

II. PARALLEL ARCHITECTURES FOR TURBO DECODERS

Different forms of parallelism can be introduced in a turbo decoder. First of all, given a decoding unit capable of iteratively processing one block at the time, one could think to simply organize multiple decoding units in a parallel architecture that simultaneously works on a number of data blocks. Despite the fact that this straightforward approach provides a throughput speed-up proportional to the number of allocated processors with no increase in the decoding latency, it has not been adopted for practical implementations, mainly due to the unacceptable growth of complexity and particularly

Manuscript received March 30, 2006; revised August 25, 2006. This research was supported in part by Italian Ministry of Science and Technology (MIUR) under FIRB PRIMO project, in 2004-2006.

Authors are with Department of Electronics, Politecnico di Torino, Italy (e-mail: {maurizio.martina, andrea.molino, fabrizio.vacca, guido.masera, guido.montorsi}@polito.it).

1845-6421/06/5193

in the required memory for buffering of data and interleaving. Alternatively, multiple decoders can be allocated to operate in a pipelined fashion on consecutive data blocks: this form of iteration level parallelism achieves high throughput at the cost of additional storage for the multiple blocks that are simultaneously processed.

A more efficient strategy is to exploit potential parallelism at different levels in the decoding algorithm, rather than allocating multiple complete decoders. Of course, parallelism is routinely introduced in the implementation of the *Add-Compare-Select* modules that implement the basic equations for the forward and backward metric recursion [1].

Block level parallelism is also very often exploited in high performance decoders by independently processing consecutive sub-blocks. The *sliding window* approximation [11] [12] [13] of the original decoding algorithm is routinely adopted for dividing the data block into a number of windows (length N_W) to be sequentially processed; in order to achieve high processing throughputs, identified windows can alternatively be decoded in parallel, provided that proper initialization metrics are available at the window borders. Two possibilities have been proposed to obtain these initialization metrics in a form that makes negligible the approximation effects on the code performance. In the first method (see as an example [14]), windows are extended in the forward and backward directions, so obtaining overlaps by a trellis length N_S with the two adjacent windows; these two windows extensions imply a processing overhead but N_S can be sized long enough (typically a few times the constraint length of the convolutional code [15]) to provide reliable initialization metrics for the non-overlapped part of the window, which is the only one actually decoded at each step.

The second method to initialize border metrics has been proposed in [16] and makes use of the trellis metrics evaluated at the bounds of the window in the previous decoding iteration. Also this approach has been proved to generate negligible performance losses and additionally it has no throughput and latency penalty because there is no overlap in the window processing.

A different, reduced complexity approach to exploit window level parallelism in turbo decoders has been proposed in [17] (*multiple slice turbo encoding*), where authors avoid sub-block overlapping by independently encoding each sub-block at the transmitter side.

Block level parallelism is more efficient than the allocation of multiple decoders, however it demands architectures where a number of separated SISO units operate concurrently on different windows and this implies that also interleaving memories are accessed concurrently. While the implementation of interleavers and de-interleavers in serial architectures typically requires the use of a unique memory with proper addressing, in the parallel case multiple memories must be organized to serve in parallel allocated SISO units, with the constraint that all simultaneous accesses from or to different SISOs are referred to different memories.

Of course, one can think to simply stall the decoder when a *collision* in the memory access occurs: this means that when two SISOs need to read a data from the same memory, two subsequent access operations are scheduled and the decoding process is slowed down. However it has been proved that random interleavers frequently generate collisions on parallel architectures and as a consequence more efficient solutions to this problem are required. Known approaches from the literature are usually classified in two categories:

1) either the code itself, and particularly the interleaving law, can be designed with the collision constraint in mind, so making it able to support a given degree of parallelism,
2) or decoding architectures may be conceived with a collision-free structure that enables them to completely avoid, or at least attenuate, decoding stalls

Several efficient methods to design collision-free interleavers have been proposed by many authors (see for example [18] [14] [19] [20] [21] [22] [17]): in most cases, suitable combinations of spatial and temporal permutations are adopted together with specific optimization constraints aimed at obtaining high spreads on single error events.

A smaller number of works have been published on the subject of collision-free architectures. The most relevant proposed solutions are related to specific rescheduling architectures that manage the memory conflicts by suitably buffering data [23] [24] [25] [26], and to an high complexity routing network made of crossbar switches coupled with interleaving memories that have been shown to support arbitrary permutation laws for any degree of parallelism [27] [28].

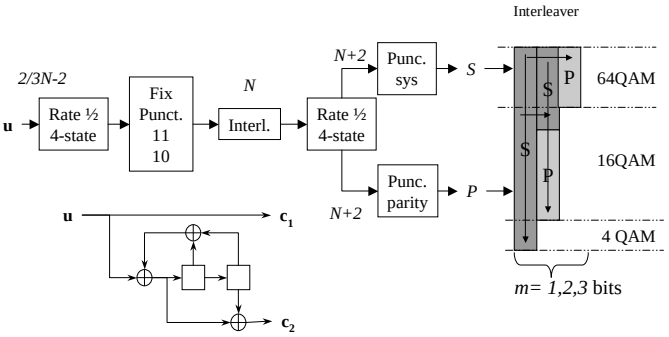

Fig. 1. Block diagram of the encoding scheme

III. A HIGH SPEED ADAPTIVE SCHEME

A versatile pragmatic encoder to be used in conjunction with high order modulation schemes has been proposed in [9]. A slight variation of it is reported in Figure 1: the addressed system exploits a versatile physical layer with OFDM modulation that adapts to both the mobile channel conditions and the changing user throughput; the embedded encoding scheme, given in Figure 1, is characterized by the following features:

1) Variable information block size K that depends on the throughput of the user, as the radio frame is assumed to be fixed at 10 ms
2) Variable code rate K/N and modulation scheme, to cope with variable channel conditions

3) Since water filling techniques [29] are used to fully exploit the channel capacity, multiple modulations may be associated to the same codeword
4) Simple and fast decoder
5) Performance close to the capacity and low error floor

The encoder is build around a serial concatenation of two simple rate 1/2 4-state systematic encoders. The outer encoder is always punctured to a rate 2/3 encoder using fix puncturing pattern $[1, 1, 1, 0]$. As a consequence, the interleaver size N is always equal to $3/2K + 3$, independently of the SCCC code rate.

The variable codeword size N is obtained by puncturing the inner encoder. Different puncturing is applied to the $N + 2$ systematic and parity check bits generated by the inner encoder. Systematic bits (S in the figure) are punctured according to an optimized and incremental pattern of length 512. Parity check bits are instead punctured according to a rate-matching similar to that used in the UMTS standard.

The amount of puncturing on the two branches depends on the desired rate. Proper puncturing allows to achieve rates very close to one without performance losses. The minimum rate achievable is $K/(3K+10)$. If lower rate are desired, *repetition* is applied to the parity check bits only. The generated bits are then mapped to the different modulation symbols associated to the codeword through an interleaver and Gray mapping as shown in Figure 1. Most reliable bits of modulations are associated to systematic bits.

The set of interleaver sizes has been chosen to be $N = 768n$ with $n = 1, \ldots, 21$, so that $K = 512n - 2$. The interleavers have been designed to allow simple parallel implementation of the decoder with 16 parallel SISO (see Figure 2). The simple

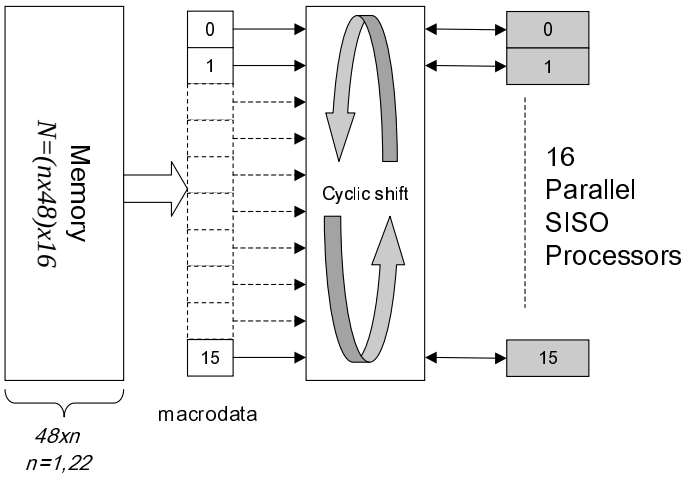

Fig. 2. The designed interleaver with simplified parallel access.

structure also allows to represent them with a more compact data structure, as detailed in section IV-B

Performance of the scheme are reported in Figure 3. In the figure we report the loss with respect to the *unconstrained capacity* of the scheme as a function of the spectral efficiency, measured in bits per dimension. The interleaver size is $10,000$ while modulations ranges from 4QAM up to 4096QAM. The FER where the signal-to-noise ratio was measured is 10^{-3}.

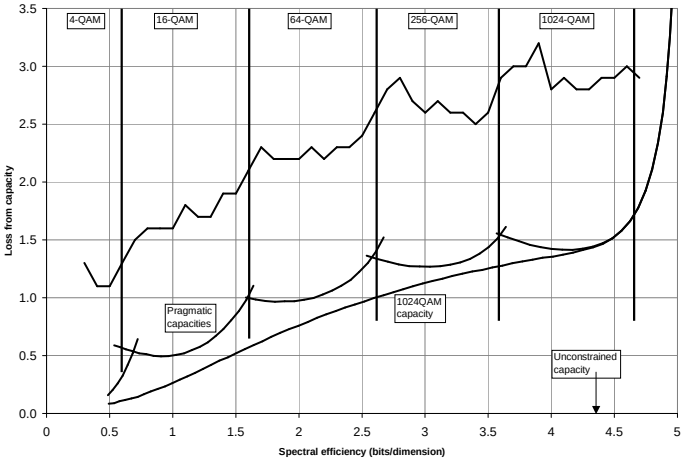

Fig. 3. Loss from capacity of proposed scheme and pragmatic capacities vs the spectral efficiency.

On the same plot we report for reference the capacity of the 1024QAM and the capacities of the various scheme when using the *pragmatic* (BICM) approach. Note that the pragmatic capacities cross each other and this justify to switch from one modulation scheme to another when increasing the spectral efficiency. The scheme shows, up to 64QAM, a constant loss from correspondent capacity of roughly 1dB, independently of the modulation and spectral efficiency. Above 64QAM the losses seems to slightly increase.

In Figure 4 we report the performances of the quantized decoder for three different cases, namely 4QAM with spectral efficiency η 1 bit/s/Hz, 16QAM with spectral efficiency 3 bit/s/Hz and 64QAM with spectral efficiency 5 bit/s/Hz. Solid lines are the performances of the ideal floating point decoder. Short dashed lines are the performance of a decoder that uses 5 bits for the LLR representation and 7 bits for the representation of extrinsic information (loop variables). In the representation of both LLR and extrinsic information, two bits are kept to the right of the point and this imply the use of a LUT of 9 elements for the $\max^*$ operation (see eq. 11 in the following Section). Finally, long dashed lines are the performance of a decoder that uses 6 bits for the LLR representation and 7 bits for the representation of extrinsic information (loop variables),

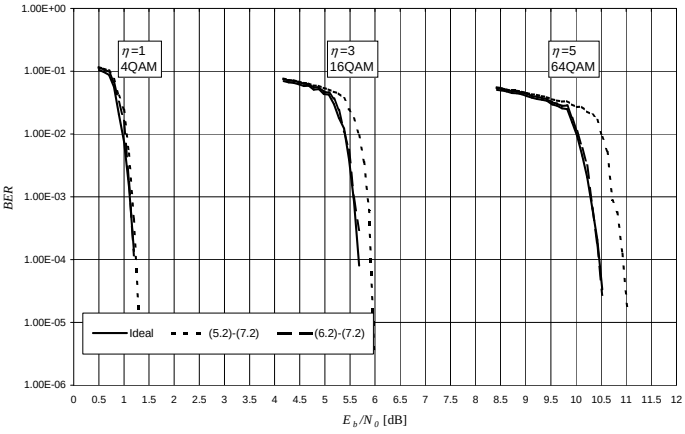

Fig. 4. Performance of quantized decoder for three different spectral efficiency.

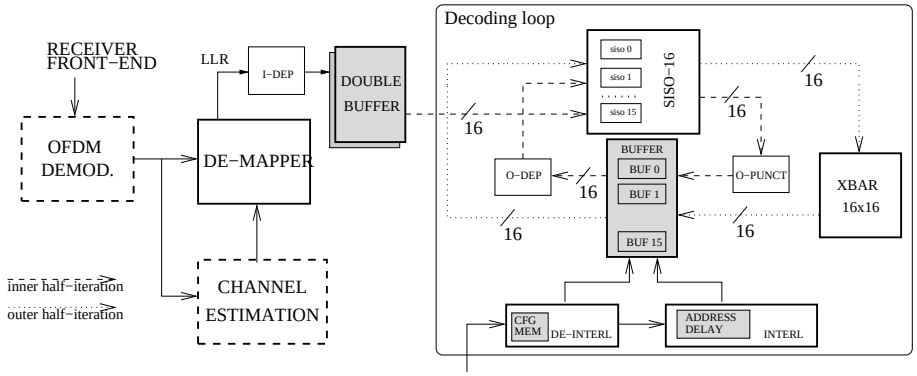

Fig. 5. Decoder block scheme.

with the same precision. From the curves it is evident that this last solution offers performances almost identical to the ideal case. A larger dynamic for the LLR should be chosen if larger constellation (e.g 256 or 1024 QAM) are used.

IV. Implementation

The overall high level scheme of the implemented system is shown in Figure 5. The OFDM demodulator gets symbols from the receiver front–end and provides samples to the de–mapper that generates logarithmic likelihood ratios (LLR) to the decoder, according to selected modulation and obtained channel estimation. De–mapper and channel decoding units have been implemented in hardware, while dashed blocks in Figure 5 are not described in this work. I–DEP block inverts the inner puncturing procedure applied at the encoding side, so adapting the global rate at the input of the decoding loop that basically implements the iterative MAP algorithm by means of a parallel structure. In order to support high data–rates, a double memory buffer has been instantiated between the inner depuncturer and the decoding loop. This double buffer guaranties that the non iterative part of the decoder (namely de–mapper and de–puncturer) and the decoding loop work simultaneously on consecutive data blocks.

The decoding loop in Figure 5 is organized to handle both inner and outer half–iterations and it is made of the following main units. SISO–16 includes sixteen SISO processors that concurrently operate on different windows along the trellis and synchronize exchanged data through FIFO (First In First Out) memories; DE–INTERL and INTERL blocks implement the required metric permutations by generating proper addresses for the data memory that is mapped to the BUFFER unit, consisting of sixteen parallel memories; XBAR is a 16×16 cross-bar switch network allowing for proper memory to processor mappings with no collisions, for all supported interleaver sizes; finally O–PUNCT simply applies the outer puncturing pattern, while O–DEP performs the opposite operation.

Two paths are distinguished in the decoder scheme, related to inner (dashed lines) and outer (dotted lines) half-iterations; both paths connect processing or storage units that are internally organized as parallel architectures, thus each connection carries 16 metrics at the same time.

A. Parallel SISO architecture

The proposed parallel SISO architecture is based on 16 processors working in parallel (see Fig. 6).

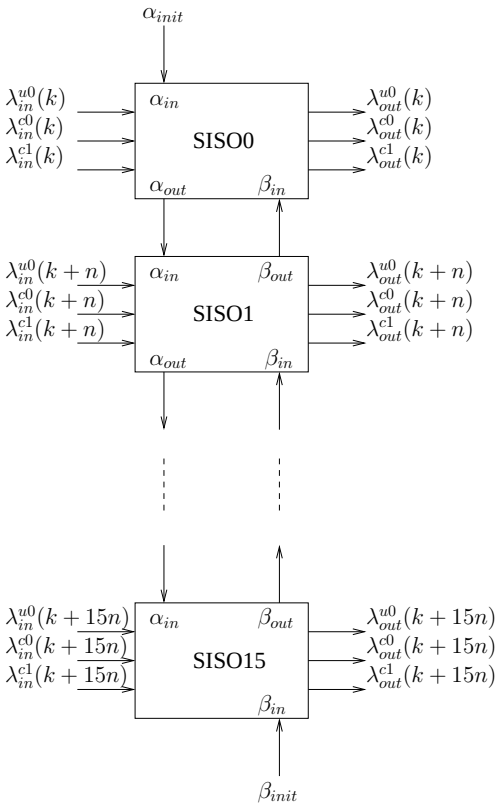

Fig. 6. Parallel SISO architecture

The single SISO, depicted in Fig. 7, works on consecutive windows of logarithmic likelihood ratios (LLRs) provided by the demapper and indicated as $\lambda_{in}(k)$. The considered code leads to a 4 states SISO that receives 3 LLRs at each clock cycle

$$\lambda_{in}^{u0}(k) \qquad \lambda_{in}^{c0}(k) \qquad \lambda_{in}^{c1}(k) \tag{1}$$

(related to encoder input and output symbols $u0$, $c0$ and $c1$ respectively) and combines them to produce the branch metrics

(BMs)

$$\gamma_i(k) = \lambda_{in}^{u0}(k)$$
$$\gamma_{o1}(k) = \lambda_{in}^{c0}(k) + \lambda_{in}^{u0}(k) \qquad \gamma_{o2}(k) = \lambda_{in}^{c1}(k) \quad (2)$$

This operation is combinatorially performed by the branch metric unit (BMU) that also generates the addresses and the write enable signals to store BMs into the BM memories (BM-mem0 and BM-mem1).

Since each SISO elaborates in pipeline two windows, the BMs, produced by the BMU, are immediately used by the α processor (α-proc) to compute the α recursion. At the same time these BMs and their α values ought to be stored into the BM memories and into the α memories (α-mem0 and α-mem1) in order to later perform the β recursion. This task is accomplished by the β processor (β-proc); in addition this processor generates proper addresses and read enables to read BMs and α values from respective memories. In fact the BMs ought to be delayed and combined with properly aligned α and β values to produce the output LLRs $(\lambda_{out}^{u0}(k), \lambda_{out}^{c0}(k), \lambda_{out}^{c1}(k))$.

The correct metric computation into the α and β processors is granted by the metric wrapping arithmetic [30]. The two metric processors compute the state equations defined by the 4 states trellis:

$$\hat{\alpha}_0(k+1) = \overset{*}{\max}(\hat{\alpha}_0(k), \gamma_{o1}(k) + \gamma_{o2}(k) + \hat{\alpha}_2(k)) \quad (3)$$
$$\hat{\alpha}_1(k+1) = \overset{*}{\max}(\hat{\alpha}_2(k), \gamma_{o1}(k) + \gamma_{o2}(k) + \hat{\alpha}_0(k)) \quad (4)$$
$$\hat{\alpha}_2(k+1) = \overset{*}{\max}(\gamma_{o2}(k) + \hat{\alpha}_3(k), \gamma_{o1}(k) + \hat{\alpha}_1(k)) \quad (5)$$
$$\hat{\alpha}_3(k+1) = \overset{*}{\max}(\gamma_{o2}(k) + \hat{\alpha}_1(k), \gamma_{o1}(k) + \hat{\alpha}_3(k)) \quad (6)$$

and

$$\hat{\beta}_0(k-1) = \overset{*}{\max}(\hat{\beta}_0(k), \gamma_{o1}(k) + \gamma_{o2}(k) + \hat{\beta}_1(k)) \quad (7)$$
$$\hat{\beta}_1(k-1) = \overset{*}{\max}(\gamma_{o2}(k) + \hat{\beta}_3(k), \gamma_{o1}(k) + \hat{\beta}_2(k)) \quad (8)$$
$$\hat{\beta}_2(k-1) = \overset{*}{\max}(\hat{\beta}_1(k), \gamma_{o1}(k) + \gamma_{o2}(k) + \hat{\beta}_0(k)) \quad (9)$$
$$\hat{\beta}_3(k-1) = \overset{*}{\max}(\gamma_{o2}(k) + \hat{\beta}_2(k), \gamma_{o1}(k) + \hat{\beta}_3(k)) \quad (10)$$

where $\hat{\alpha}_i(k)$, $\hat{\beta}_i(k)$ with $i \in \{0,1,2,3\}$ represent the 4 forward and backward SMs ($\alpha(k) = [\hat{\alpha}_0(k) \dots \hat{\alpha}_3(k)]$ and $\beta(k) = [\hat{\beta}_0(k) \dots \hat{\beta}_3(k)]$); the operator $\max^*$ is the same defined in [31], namely

$$\overset{*}{\max}(a,b) \triangleq \max(a,b) + \ln\left(1 + e^{-|a-b|}\right) \quad (11)$$

The α and β processor architecture is based on 4 $\max^*$ blocks working in parallel, one for each SM. Every $\max^*$ has been implemented as an add–compare–select (ACS) block with a look–up–table (LUT) for the logarithmic correction [32]. The logarithmic correction is based on 3 fractional bits, leading to a 22×3 bits LUT [31]. Despite the wrapping, the SMs relative distance is preserved during the computation. However to combine the SMs and to compute the output LLRs, they need to be renormalized. The "norm" blocks in Fig. 7 are devoted to perform the renormalization: the minimum SM among $\hat{\alpha}_0(k)$, $\hat{\alpha}_1(k)$, $\hat{\alpha}_2(k)$ and $\hat{\alpha}_3(k)$ (or $\hat{\beta}_0(k)$, $\hat{\beta}_1(k)$, $\hat{\beta}_2(k)$ and $\hat{\beta}_3(k)$) is found and subtracted to obtain $\tilde{\alpha}_i(k) = \hat{\alpha}_i(k) - \min\{\hat{\alpha}_j(k) | j \in \{0,1,2,3\}\}$ (or $\tilde{\beta}_i(k) = \hat{\beta}_i(k) - \min\{\hat{\beta}_j(k) | j \in \{0,1,2,3\}\}$). Then the normalized SMs, combined with the BMs generate the output LLRs as:

$$\lambda_{out}^{u0}(k) = \gamma_t(k) - \gamma_i(k) \quad (12)$$
$$\lambda_{out}^{c0}(k) = \gamma_t(k) - \gamma_{o1}(k) \quad (13)$$
$$\lambda_{out}^{c1}(k) = \gamma_t(k) - \gamma_{o2}(k) \quad (14)$$

where

$$\gamma_t(k) = \overset{*}{\max}(x_3(k), x_1(k)) - \overset{*}{\max}(x_2(k), x_0(k)) \quad (15)$$
$$x_3(k) = \overset{*}{\max}(\gamma_{o1}(k) + \gamma_{o2}(k) + \tilde{\alpha}_2(k) + \tilde{\beta}_0(k), \gamma_{o1}(k) + \gamma_{o2}(k) + \tilde{\alpha}_0(k) + \tilde{\beta}_1(k)) \quad (16)$$
$$x_1(k) = \overset{*}{\max}(\gamma_{o1}(k) + \tilde{\alpha}_1(k) + \tilde{\beta}_2(k), \gamma_{o1}(k) + \tilde{\alpha}_3(k) + \tilde{\beta}_3(k)) \quad (17)$$
$$x_2(k) = \overset{*}{\max}(\gamma_{o2}(k) + \tilde{\alpha}_3(k) + \tilde{\beta}_2(k), \gamma_{o2}(k) + \tilde{\alpha}_1(k) + \tilde{\beta}_3(k)) \quad (18)$$
$$x_0(k) = \overset{*}{\max}(\tilde{\alpha}_0(k) + \tilde{\beta}_0(k), \tilde{\alpha}_2(k) + \tilde{\beta}_1(k)) \quad (19)$$

The output generation, implemented into the γ processor is based on a tree of adders and ACS as described in [32].

The correct scheduling of the operations is mainly performed by two control units (α-CU and β-CU) devoted to start the BMU and the β processors respectively. Moreover the control units manage the correct multiplexers selection and the FIFOs. It is worth pointing out that the α and γ processors are synchronized by the BM memories write enables and the α memories read enables respectively (see Fig. 7).

Instead of implementing a training window to grant the SMs reliability, each SISO uses the boundary SMs evaluated at the $i-1$th iteration as the initialization SMs for the ith iteration [16] as depicted in Fig. 8.

In the implemented architecture, the control unit complexity has been reduced by imposing that all SISOs process the same number of windows, N_w. Both window size w and N_w have to be tuned on the basis of the input block length, K, to be supported. To size the window we have to consider that each outer SISO follows a de-puncturer and is followed by a puncturer, devoted to make the rate $R = 1/2 \times 4/3 = 2/3$. In the presented implementation we employ $w = 32$ or $w = 64$ for the outer SISO, and $w = 48$ or $w = 96$ for the inner SISO. As a consequence of this choice, the interleaver size (I) is quantized to satisfy the condition $I = 48 \times 16 \times l = 768l$ with $l \in \mathbb{N}_+$. This restriction allows the 16 SISO to process the same number of windows and grants a great simplification in the parallel SISO control unit. Considering that every convolutional encoder adds 2 termination bits, $I = 768l$

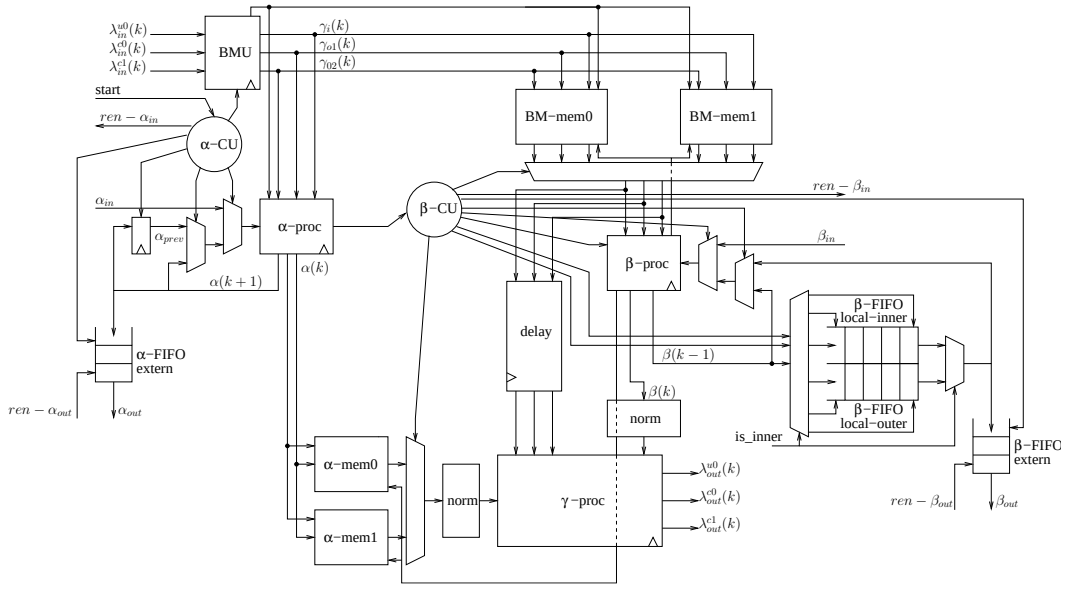

Fig. 7. Single SISO architecture

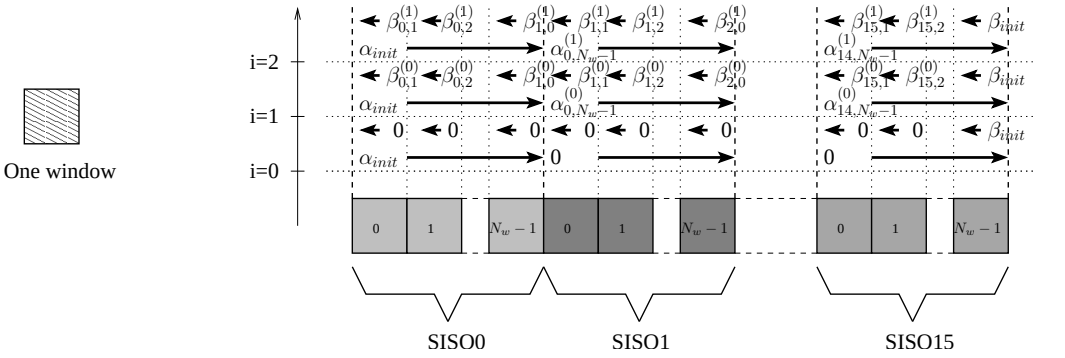

Fig. 8. Parallel elaboration with state metric passing scheme

implies $K = 512l - 2$.

One of the key point in the organization of parallel SISO units is related to the exchange of initialization state metrics (SMs) for both α and β processing (see Figure 6). The synchronized propagation of metrics among SISOs is achieved through 4 FIFO memories, as detailed in the following.

In the α recursion, the first window of each SISO requires to load its α_{in} value from a neighbor (the nth SISO from the $n-1$th). In other words, the α_{out} of the $n-1$th SISO, becomes the α_{in} of the nth one. Similarly in the β recursion, the last window of each SISO requires to load its β_{in} from a neighbor: the β_{out} of the $n+1$th SISO becomes the β_{in} of the nth one (see Fig. 6). Since each SISO acts both as inner and outer, the inter-SISO SMs propagation is managed through 2 FIFOs (α-FIFO extern and β-FIFO extern in Fig. 7); each FIFO has 2 positions (inner and outer).

In addition to inter-SISO communication, intra-SISO SMs propagation also needs to be correctly managed. In the α recursion, for every window, each SISO stores the last α values into the α_{prev} register. These values will be used as the initialization α for the next window. As described in Fig. 8, the last α values of the ith SISO at the kth iteration are $\alpha^{(k)}_{i,N_w-1}$.

On the other hand the β inheritance is more complex. The β recursion is performed in direct order on the windows $(0 \dots N_w - 1)$ and in reverse order on each window ($k = w-1 \dots 0$). As a consequence each SISO, for every window, needs to load a proper starting β. The starting β (β_{in}) for the last window in the nth SISO is loaded from the $n+1$th one through the β-FIFO extern. However to correctly manage intra-SISO β initialization another FIFO is required (β-FIFO local in Fig. 7). For the ith SISO at the kth iteration the last β of the jth window ($\beta^{(k)}_{i,j}$) is pushed into the FIFO. Since the windows are processed in direct order the first β pushed in the β-FIFO local ($\beta^{(k)}_{i,0}$) has to be extracted and pushed into the β-FIFO extern. In fact at the next iteration it will become the β_{in} for the $i-1$th SISO (see Fig. 8). Furthermore, since the SISO acts both as inner and outer, two β-FIFO local are

required, inner and outer respectively (see Fig. 7).

B. Interleaver-Deinterleaver

The interleaver implementation is based on a 16–ports non pipelined crossbar switch. The lowest $N \times N$ network free of internal blocking and able to implement a generic permutation of inputs is the Benes network; however for low to modest numbers of inputs (up to 64) the crossbar is known to be the preferred choice, mainly because of reduced cost and simplicity. The obtained combinatorial delay for the implemented 16×16 crossbar is shorter than the worst path delay of the whole decoder architecture, allowing a working frequency up to 115 MHz on the prototyping FPGA.

The implemented interleaving scheme is the one in Figure 2, where the data stored in the memory buffer have to be dispatched to the 16 SISOs. The permutation law is composed by two steps. The former is a traditional index translation, based on a set of pre–stored permutations (called *macrodata* in the figure). This first set of permutations are common to the 16 buffers. In the latter phase (called *Cyclic shift*) a displacement is added to the obtained temporary address. This displacement is a number in the space 0–15, meaning an additional translation of a given number of window length. This part is periodical and cyclic, i.e. the displacement IDX_{adx} for a given address adx can be obtained as $(IDX_{(adx-n\times WSIZE)} + n \times WSIZE)_{MOD16}$, where $WSIZE$ is the current window size.

The memory containing the interleaving law is included in the de–interleaver unit, and it can be directly accessed in write mode from outside at every new block size configuration. The interleaving law memory (CFG–MEM) contains $\frac{B_{max}}{16} \times \log_2\left(\frac{B_{max}}{16}\right)$ bits (e.g. ≈ 10 kbit for a maximum block size of 16384 bits).

Two buffers (ADDRESS DELAY) have been allocated as address queues, which hold and properly synchronize the generated addresses required by the SISO processors in order to make access to the de–interleaving memories. As SISO processors output extrinsic information in the reverse order, a LIFO (Last In First Out) policy is required for these queues to share the same interleaving law among INTERL and DE–INTERL.

The obtained memory usage is a number of address words equal to the latency of SISO processors, that is twice the maximum supported window size (i.e. 96). Therefore, for the implemented scheme the memory requirement is $2 \times \log_2\left(\frac{B_{max}}{16}\right) \times 96 \approx 2$ kbit.

C. Demapper

In figure 9 the block scheme of the whole demapper is depicted.

The following signals are received by the demapper block:

- $r_i(t_0)$ Complex signal received on the i–th sub–carrier at time time t_0.
- α_i Channel state information relative to the i–th sub–carrier. This data comes directly from the Channel Estimation block.

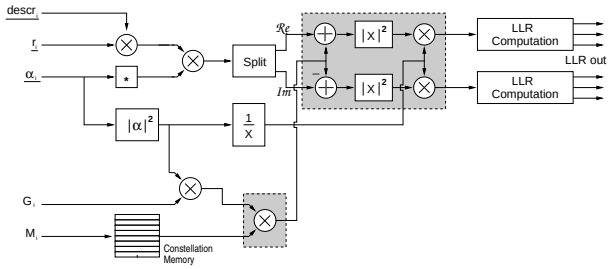

Fig. 9. Demapper block scheme.

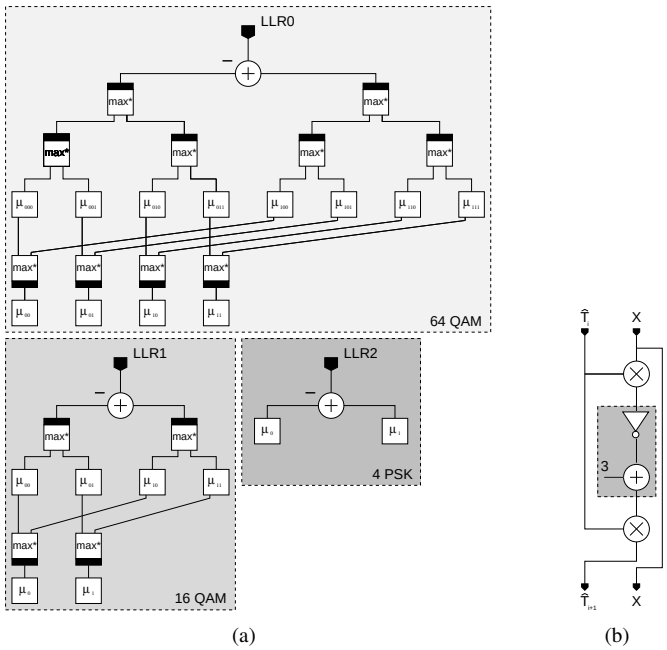

Fig. 10. Detail on the data–path implementation for two key blocks in the demapper: (a) represents the tree–structure used to produce LLR in the LLR Computation Unit, while in (b) the basic Newton–Ramphson step is sketched.

- descr_i Descrambling sequence relative to the received signal. This is useful to support carrier reusing among neighboring cells.
- G_i Gain associated to the i–th sub–carrier. These gains can be changed by the transmitter following particular strategies in order to maximize the Signal–to–Noise–Ratio (SNR) for a given sub–carrier. For these reasons the designed core needs to receive the proper gain value for each of the symbols to be demapped.
- M_i Modulation for the i–th sub–carrier. Valid values can be:
  - "00" this means no modulation for this carrier.
  - "01" this stands for 4-PSK.
  - "10" for 16-QAM.
  - "11" for the 64 QAM.

On the left hand side of Figure 9, the aforementioned inputs are shown, while the computed Log Likelihood Ratios (LLR) provided to the channel decoder, are depicted on the rightmost part. Two distinct regions can be seen in the demapper structure. The lighter one contains those blocks instanced only once; the darker region, on the other hand, contains blocks that have to be instanced as many times as the maximum

modulation cardinality (i.e. eight times for the 64 QAM).

In figure 10(a) the data path of the LLR Computation Unit (LLR–CU) is proposed. The eight received input values are the normalized Euclidean distances calculated by the leftmost part of the demapper, indicated as $\mu_{000} \dots \mu_{111}$. The LLR–CU structure is based on $\max^*$ operations and additions, while the black blocks on $\max^*$s' output are pipeline registers.

From a functional standpoint the LLR–CU can be viewed as composed by three successive stages. Each of these stage is devoted to the computation of a single LLR: as the input μ traverse the structure up to three LLR can be computed. An important feature of LLR–CU structure is that all the computed outputs have the same latency with respect to the input μ. For this reason we decide to insert new computed μ always as the input of the 64 QAM stage. In case of smaller modulations, data are simply fed toward the successive stage (16 QAM or even 4 PSK), but the latencies are always respected. In this way no stalls nor congestions can occur, leading to an improved throughput.

A remarkable number of real and complex multiplications are required in the leftmost part and they are mapped to FPGA embedded multipliers. Unfortunately the demapping process requires also a division, which poses more serious implementation problems. In this particular case the hardware division has been expressed through the use of multiplicative inverse (reciprocation) and the popular Newton–Ramphson function approximation method has been adopted to perform the reciprocation. Let X be the value for which we want to determine the inverse. If we we define a "reciprocal function" as:

$$f(T) = \frac{1}{T} - X \tag{20}$$

then this function will have its zero in $T = 1/X$. Then we can use the Newton–Ramphson method to find equation's (20) zero. In this way we obtain the following relationship, which is actually the one we used in the implemented architecture.

$$T_{i+1} = T_i(2 - XT_i) \tag{21}$$

In figure 10(b) the data path of a single Newton–Ramphson (NR) step is depicted: the unit can be easily pipelined to achieve a high throughput; moreover, as fixed–point simulations showed that very good approximation results can be obtained performing only two NR steps, the entire reciprocal unit architecture has been designed cascading two NR steps. An additional small ROM (256×8) is then required to find the proper initial approximation (T_0).

D. Latency and throughput

The allocated double buffer separating the non iterative part of the system from the decoding loop requires that a whole data block is completed by the parallel scheme before a new block is stored into the buffer. Being D_L the latency of the inner–depuncturer, one single buffer is ready in a time estimated as $T_D = D_L \times B \times T_{ck}$, where B is the incoming data block length and T_{ck} is the internal clock period.
The execution time for the decoding loop that includes SISO processors and interleaving memories can be written as:

$$T_L = \delta \times \frac{B}{16}\left(1 + \frac{2}{3}\right) \times Iter \times T_{ck} \tag{22}$$

where $\frac{B}{16} \times 1$ is the time required for the inner–SISOs to read the input buffer once, additional $\frac{B}{16} \times (2/3)$ cycles are required for the outer–SISOs reading, and $Iter$ is the iteration number. The parameter δ is the overhead due to the internal processing pipeline, which has been estimated on actual simulation waforms as 1.18 in the worst case conditions.

Using expressions provided above with $D_L = 2$ and $\delta = 1.18$, we obtain that $T_D = 2B \times T_{ck}$ while $T_L \approx 0.12B \times Iter \times T_{ck}$. Therefore, assuming that the iteration number is lower than 16, we have $T_D < T_L$, meaning that a whole data block is decoded before the following block is loaded into the input buffer.

Thanks to its regular and highly parallel structure, the demapper block exhibits a constant latency regardless the modulation used or the specific channel state information. More in detail, the demapper latency can be expressed as $D_{dmp} = 2 + D_\mu + D_{LLR-CU}$, where D_μ is the latency of the entire Euclidean distance computation part (see figure 9) and D_{LLR-CU} is the latency of the LLR–CU part.

For the implemented architecture we have $D_\mu = 7T_{ck}$ and $D_{LLR-CU} = 2T_{ck}$, leading to a total latency of 11 clock cycles. However such a deeply pipelined structure ensures to be able to produce six new LLRs for each clock cycle, leading to throughput of $THR_{dmp} = \frac{6}{T_{ck}}$ LLRs per second.

V. System performance and synthesis results

The demapping and decoding architectures described in the previous sections have been completely described and validated in VHDL. Then, to prove functional correctness, the core has been successfully validated in an actual physical layer implementation on FPGA and tested on a prototyping board.

Logical and physical synthesis have been targeted for a Xilinx XC2VP100 device that offers the required resources in terms of internal block RAMs, embedded multipliers and equivalent gates. The placed design requires:

- 20265 SLICEs out of 44096, i.e. 46% of device resources
- 367 18kb Block Select RAMs out of 444 available.

for the decoder, including the input double buffer and the de–puncturing unit, and additional

- 442 SLICEs out of 44096, i.e. around 1% of device resources
- 8 embedded 18×18 multipliers, out of the 444 available
- 540 16x1 ROMs.

for the flexible de–mapper section; these figures refer to a simplified version of the de–mapper described in section IV-C, which has been scaled in order to adapt its throughput to the one admitted by the iterative decoder.
The detailed cost of main modules in the designed structure are given in Table I, while the global percentages of device usage are 54% in terms of combinational and sequential distributed elements, 83% in terms of Block Select RAMs and 13% in

TABLE I
DETAILS OF ARCHITECTURE COMPLEXITY

unit	registers	comb. logic	multipl.	Block Select RAM
demapper	310	577	8	-
turbo decoder	21255	37972	0	230
de–puncturing	999	1826	0	5
double buffer	202	732	0	132

terms of multipliers.

According to Xilinx ISE estimations, the total equivalent ASIC gate count for this IP is of 703,000; a total amount of memory of 4.8 Mbit is required for interleavers and buffers. For the implemented parallel decoder, the maximum supported throughput depends on block length and iteration number: at the clock frequency of 80 MHz and with a fixed number of ten iterations, the throughput ranges from 20 Mbit/s for the minimum block size (510 bits), up to 35 Mbit/s in the case of the longest block size (10,750 bits).

A. Fully parallel de–mapper

The fully parallel high-speed de–mapper has also been synthetized as a standalone unit, with the purpose of scaling up the offered throughput. The demapper block scheme (Figure 9) does not include any loop nor control–dependent path. For this reason, resorting to parallel techniques to achieve very high throughputs is probable the most attracting design choice. The purpose of the fully parallel de–mapper is to support one new complex value per clock cycle, with a different modulation choice and channel state information. To achieve such a remarkable data rate without pushing the clock frequency at higher values parallel techniques have to be exploited. More in detail two different types of parallelism can be employed, namely pipelining as well as parallel computation. As far as the pipelining is concerned, the proposed architecture is composed by 16 pipeline stages, so showing a latency of 16 clock cycles. However, once the pipeline is fully loaded, it is possible to extract up to 6 LLRs per clock cycle.

On the other hand parallel computation requires the duplication of some data path resources in order to reduce the number of clocks needed to compute the results. In this case, the constellation scaling operations as well as the distances evaluation have been implemented in parallel (see darker regions in figure 9). This choice enables us to "feed" the two rightmost LLR Computation networks with sixteen new distances for each clock cycle.

The synthetized fully parallel high-speed de–mapper requires 327,000 equivalent gates, corresponding to 8% of a Xilinx XC2VP100, and achieves a maximum decoding throughput of 480 Mbps with a clock frequency of 80 MHz.

VI. CONCLUSIONS

The paper presents implementation details of an adaptive high performance modulation and coding scheme mapped to a unique advanced FPGA device. The designed system supports a wide range of spectral efficiencies, achieving performance within 1 dB from the capacity. On the hardware side, careful architecture design of de–mapping, de–puncturing and decoding units have been performed with the aim of incorporating the required flexibility and achieving very high throughput at the same time. In particular, 16 parallel SISO units have been allocated in the decoder that concurrently process multiple windows of length adaptively selected on the basis of the required spectral efficiency. Synthesis results, derived for a clock frequency of 80 MHz, report a maximum throughput of 35 Mbit/s for 10 iterations, with a percentage device occupation close to 54% for logic cells and 83% for embedded RAMs.

REFERENCES

[1] Berrou, C.; Glavieux, A.; Thitimajshima, P.; "Near Shannon limit error-correcting coding and decoding: Turbo-codes", IEEE International Conference on Communications, 1993. ICC 93. Geneva. Volume 2, 23-26 May 1993 Page(s):1064 - 1070 vol.2

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

[3] Third-Generation Partnership Project. 3GPP specifications. [Online]. Available: http://www.3GPP.org

[4] Third-Generation Partnership Project 2. 3GPP2 specifications. [Online]. Available: http://www.3GPP2.org

[5] C. Douillard, M. Jezequel, C. Berrou, N. Brengarth, J. Tousch, and N. Pham, The turbo codec standard for DVB-RCS, 2nd Int. Symp. Turbo Codes and Related Topics, Brest, France, Sep. 2000

[6] "Telemetry Channel Coding", Consultative commitee for space data systems (CCSDS), Blue Book, 101.0-B-4, 1999

[7] Bosco, G.; Montorsi, G.; Benedetto, S.; "Soft decoding in optical systems" IEEE Transactions on Communications, Volume 51, Issue 8, Aug. 2003 pp. 1258 - 1265

[8] E. Yeo, P. Pakzad, B. Nikolic, V. Anantharam (2001) "VLSI architectures for iterative decoders in magnetic recording channels", IEEE Trans. Magn., vol. 37, pp. 748-755, Mar. 2001

[9] S. Benedetto, R. Garello, G. Montorsi, C. Berrou, C. Douillard, D. Giancristoforo, A. Ginesi, L. Giugno, M. Luise, "MHOMS: High-speed ACM Modem for satellite applications", IEEE Wireless Communications, April 2005

[10] O. Muller, A. Baghdadi, M. Jezequel, "ASIP-baser Multiprocessor oOC Design for Simple and Double Binary Turbo Decoding", Design Automation & Test in Europe Conference, Munich, Germany, March 6-10, 2006

[11] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara (1996) "Soft input soft output MAP module to decode parallel and serial concatenated codes", in TDA Progr. Rep. 42-127, Jet Propulsion Lab., Pasadena, CA, pp. 1-20, 1996.

[12] S. A. Barbulescu (1996) "Iterative decoding of turbo codes and other concatenated codes", Ph.D. dissertation, Univ. South Australia, pp. 23-24, 1996.

[13] S. S. Pietrobon (1996) "Efficient implementation of continuous MAP decoders and a synchronization technique for turbo decoders", in Proc. Int. Symp. Inform. Theory Appl., Victoria, B.C., Canada, 1996, pp. 586-589.

[14] B. Bougard, A. Giulietti, L. Van der Perre, F. Catthoor (2002) "A class of power efficient VLSI architectures for high speed turbo-decoding", Global Telecommunications Conference, 2002, GLOBECOM 2002, Vol. 1 , pp. 549-553

[15] A. Hunt, S. Crozier, M. Richards, K. Gracie (1999) "Performance degradation as a function of overlap depth when using sub-block processing in the decoding of turbo codes", Proc. of IMSC'99, 1999, Ottawa, Canada, pp. 276-280

[16] Aliazam Abbasfar and Kung Yao (2003) "An efficient and practical architecture for high speed turbo decoders" Proceedings of the IEEE Vehicular Technology Conference, 2003, Volume 1, 6-9 Oct. 2003, pp. 337 - 341

[17] D. Gnaedig, E. Boutillon, M. Jezequel, V.C. Gaudet, P.G. Gulak (2003) "Multiple Slice Turbo Codes" Proceedings of the 3rd International Symposium on Turbo Codes and Related Topics, pp 343-346, Brest, France, Sept. 2003

[18] A. Giulietti, L. Van der Perre, M. Strum (2002) "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements", Electronics Letters, Vol. 38, Iss. 5, Feb. 2002, pp. 232-234

[19] J. Kwak, K. Lee (2002) "Design of dividable interleaver for parallel decoding in turbo codes", Electronics Letters, Vol. 38, Iss. 22, Oct. 2002, pp.1362-1364
[20] J. Kwak, S. Min Park, S. Yoon, K. Lee (2003) "Implementation of a parallel turbo decoder with dividable interleaver", Int. Symp. on Circuits and Systems, 25-28 May 2003
[21] A. Nimbalker, T.K. Blankenship, B. Classon, T.E. Fuja, D.J. Costello Jr. (2003) "Inter-Window Shuffle Interleavers for High Throughput Turbo Decoding", Proceedings of the 3rd International Symposium on Turbo Codes and Related Topics, pp 355-358, Brest, France, Sept. 2003
[22] R. Dobkin, M. Peleg, R. Ginosar (2003) "Parallel VLSI architectures and Parallel Interleaving Design for Low- Latency MAP Turbo Decoders", Technical Report CCIT-TR436, Electrical Engineering, Technion-Israel Institute of Technology, July 2003
[23] M.J. Thul, F. Gilbert, N. Wehn (2002) "Optimized concurrent interleaving architecture for high-throughput turbodecoding", 9th Int. Conf. On Electronics, Circuits and Systems 2002, vol. 3, pp. 1099-1102
[24] F. Gilbert, M.J. Thul, N. Wehn (2002) "Communication centric architectures for turbo-decoding on embedded multiprocessors", Conference and Exhibition on Design, Automation and Test in Europe 2003, pp. 356-361
[25] M.J. Thul, F. Gilbert, N. Wehn (2003) "Concurrent Interleaving architectures for high-throughput channel coding", Proceedings of ICASSP 2003, Vol. 2, pp. 613-616
[26] F. Speziali, J. Zory (2004) "Scalable and area efficient concurrent interleaver for high throughput turbo-decoders", Euromicro Symposium on Digital System Design, Aug. 31 - Sept. 3, 2004 pp. 334- 341
[27] A. Tarable, G. Montorsi, S. Benedetto (2003) "Mapping interleaving laws to parallel Turbo decoder architectures", Proceedings of the 3rd International Symposium on Turbo Codes and Related Topics, pp. 153-156, Brest, France, Sept. 2003
[28] A. Tarable, S. Benedetto (2004) "Mapping interleaving laws to parallel Turbo decoder architectures", IEEE Comm. Letters, Vol. 8, No. 3, March 2004, pp. 162-164
[29] Wei Yu; Wonjong Rhee; Boyd, S.; Cioffi, J.M.; "Iterative water-filling for Gaussian vector multiple-access channels", IEEE Transactions on Information Theory, Volume 50, Issue 1, Jan. 2004 Page(s):145 - 152
[30] A. P. Hekstra (1989) "An Alternative to Metric Rescaling in Viterbi Decoders", IEEE Trans. on Communications, Vol. 31, No. 11, November 1989, pp. 1220-1222
[31] G. Montorsi, S. Benedetto (2001) "Design of Fixed-Point Iterative Decoders for Concatenated Codes with Interleavers", IEEE Journ. on Selected Areas in Communications, Vol. 19, No. 5, May 2001, pp. 871-882
[32] G. Masera, G. Piccinini, M. Ruo Roch, M. Zamboni (1999) "VLSI Architectures for Turbo Codes", IEEE Trans. on VLSI, Vol. 7, No. 3, September 1999, pp. 369-379
[33] G. Sapountzis, and M. Katevenis (2005) "Benes Switching Fabrics with O(N)–Complexity Internal Backpressure", IEEE Communications Magazine, Vol. 43, No. 1, January 2005, pp. 88-94

Maurizio Martina Maurizio Martina was born in Pinerolo, Italy, in 1975. He received the M.Sc. and Ph.D. in electrical engineering from Politecnico di Torino, Italy, in 2000 and 2004. He is currently a Postdoctoral Researcher at the VLSI Lab, Politecnico di Torino. His research activities include VLSI design and implementation of architectures for signal processing and comunications.

Andrea Molino Andrea Molino received the Dr.Ing and the Ph.D degree in Electronics Engineering from Politecnico di Torino , Italy, in 2001 and 2005 respectively. He is currently an Assistant Professor in the Department of Electronics at Politecnico di Torino. From Oct 2002 to June 2003, he was with the Video Processing Lab at University of California, San Diego (UCSD) where he worked in the development of low complexity algorithms for video coding. His current research interests are in the areas of high-performance and energy-efficient VLSI architectures for video coding and communications.

Fabrizio Vacca Fabrizio Vacca received the degree (summa cum laude) in Computer Engineering and the Ph.D. degree in Electrical Engineering from Politecnico di Torino, Italy in 2000 and 2004 respectively. He is currently a Postdoctoral Researcher at VLSI Laboratory, Politecnico di Torino. His research interests are in the field of flexible and scalable hardware architectures for telecommunications. From October 2002 to June 2003 he was a Visiting Researcher at the Videoprocessing Laboratory of University of California San Diego (UCSD). He has co-authored more than 30 scientific papers in the areas of application-specific integrated circuit (ASIC) development, optimized hardware architectures and FPGA-oriented cores.

Guido Masera Guido Masera received the Dr.Eng. degree (summa cum laude) in 1986, and the Ph.D. degree in electrical engineering from Politecnico di Torino, Italy, in 1992. Since 1986 to 1988 he was with CSELT (Centro Studi e Laboratori in Telecomunicazioni, Torino, Italy) as a researcher. Since 1992 he has been Assistant Professor and then Associate Professor at the Electronic Department, where he is a member of the VLSI-Lab group. His research interests include several aspects in the design of digital integrated circuits and systems, with special emphasis on high-performance architecture development (especially for wireless communications and multimedia applications) and on-chip interconnect modeling and optimization. He has coauthored more than 100 journal and conference papers in the areas of ASIC-SoC development, architectural synthesis, VLSI circuit modeling and optimization.

Guido Montorsi Guido Montorsi was born in Turin, Italy, on January 1, 1965. He received the Laurea in Ingegneria Elettronica in 1990 from Politecnico di Torino, Turin, Italy, with a master thesis, concerning the study and design of coding schemes for HDTV, developed at the RAI Research Center, Turin. In 1992 he spent the year as visiting scholar in the Department of Electrical Engineering at the Rensselaer Polytechnic Institute, Troy, NY. In 1994 he received the Ph.D. degree in telecommunications from the Dipartimento di Elettronica of Politecnico di Torino. In December 1997 he became assistant professor at the Politecnico di Torino. From July 2001 to July 2002 he spent one year at "Sequoia Communications" devoloping baseband algorithm for 3G wireless receivers. In 2003 he became senior member of IEEE and associate professor at Politecnico di Torino. His research interests are in the area of channel coding, particularly on the analysis and design of concatenated coding schemes and study of iterative decoding strategies. Guido Montorsi is author of more the one hundred papers on international journal and conferences.