Parallel VLSI Architecture for MAP Turbo Decoder

Reuven Dobkin, Michael Peleg and Ran Ginosar
VLSI Systems Research Center, Electrical Engineering Department
Technion—Israel Institute of Technology
Haifa 32000, Israel
[ran@ee.technion.ac.il]

Extended Abstract

Summary - Turbo codes achieve performance near the Shannon limit. Standard sequential VLSI implementation of turbo decoding requires many iterations and incurs a long latency, which cannot be tolerated in some applications. A novel parallel VLSI architecture for turbo decoding is described, comprising multiple SISO elements, operating jointly on one turbo coded block, and a new parallel interleaver. Latency is reduced ten-fold and more and throughput is increased up to eight-fold relative to sequential decoders, using the same area of silicon, and achieving the same coding gain. The parallel architecture scales favorably—latency and throughput improve with growing block size and chip area.

Index Terms: maximum a posteriori (MAP) algorithm, turbo codes, parallel architectures, decoders, interleaver.

1. Introduction

Turbo-codes [1,2] have received considerable attention since their introduction in 1993. This is thanks to achieving performance near the Shannon capacity limit along with reasonable complexity and flexibility in terms of providing different block sizes and code rates.

A turbo encoder consists of convolutional encoders connected either in parallel or in series. In the parallel scheme (Figure 1) one encoder receives the original information bits while the other receives them interleaved. The output comprises the original bits concatenated with the output of the two encoders. A turbo decoder is composed of decoding stages, performed by SISO (Soft-In Soft-Out) decoding units, with interleavers between them.

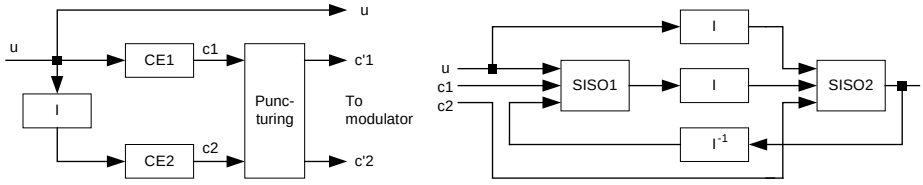

Figure 1 Turbo encoder and decoder, I denotes interleaver

The decoding is performed in an iterative way: information from one SISO is processed by the other SISO until the desired degree of convergence is achieved.

VLSI implementations of turbo decoders [4] consist either of a number of SISOs connected in a pipeline or of a number of SISOs independently processing their own encoded blocks. Both architectures are equivalent in terms of coding gain, throughput, latency and complexity. These architectures are referred to as *sequential architectures* in this paper. Due to iterative decoding the latency is very high, which may be inappropriate for many applications such as mobile communications, interactive video and telemedicine. One way to reduce the latency is to reduce the number of required decoding iterations, but this may degrade coding gain. Another approach, described in [3], is to optimize the SISO algorithm. However the gain in total decoding latency is rather small.

This paper presents a novel parallel turbo decoding architecture, which significantly reduces decoding latency and improves decoder throughput, without any degradation in coding gain relative to sequential architectures. Performance of the sequential and parallel architectures are compared. This paper also presents a method of parallel interleaving required in the proposed parallel turbo decoder architecture.

After a brief review of the decoding APP algorithm in Section 2, the sequential decoder architecture is discussed in Section3. The novel parallel decoding architecture is detailed and compared to the sequential architecture in terms of coding gain, throughput, latency and required silicon area in Section 4.

2. Turbo Coding – Theory of Operation

2.1. Encoder

The turbo encoder consists of two parallel concatenated convolutional encoders (CE) and an interleaver as depicted in Figure 1. The interleaver permutes the bit sequences (u). The interleaving is a crucial component of the turbo encoding influencing the performance [2][5].

For every new input block, the encoder starts in a known trellis state. In order to finish in a known trellis state, traditionally some extra input bits, called tail bits, are generated. There are several configurations of termination [5] that are traditionally applied in turbo code: Termination after CE1 only, terminations after CE1 and CE2, and others. The problem is that such terminations result in changes in the block size and in some cases at least one of the added terminations is not interleaved. The *tailbiting* termination technique [6], which is mostly used for recursive convolutional codes, keeps the block size unchanged. The technique finds a data dependent initial state such that the initial and the final states of the encoder are identical. The parallel decoder architecture analyzed in this paper employs the tailbiting termination technique. However, it is applicable to the other techniques as well.

2.2. Decoder

The decoding is performed using the iterative decoding scheme shown in Figure 1. Each SISO produces an increasingly better correction term, referred to as *extrinsic information,* which is appropriately (de)interleaved and used as *a* priory information

by the next SISO [2]. According to the original BCJR algorithm and using notations from [4], the probability distributions obtained as SISO output is:

$$P_k(u;O) = H_u \cdot \sum_{e:\, u(e)=u} A_{k-1}[s^S(e)] \cdot P_k[c(e);I] \cdot B_k[s^E(e)]$$

where

1) $P(u;I)$ and $P(c;I)$ are estimation of probability distribution of the encoder input (u) and output (c) symbols.
2) $P(u;O)$ is new refined value of distribution $P(u;I)$.
3) The index *k* indicates the time step and runs over the entire transmission length.
4) The symbol *s* represents a code state (state in trellis).
5) *e* is a generic trellis edge, while c(e) and u(e) are the output and input coder symbols associated with the edge *e*.
6) $s^S(e)$ and $s^E(e)$ indicate the starting and ending states for the generic trellis edge *e*.
7) H_c is a normalization constant.
8) A_{k-1} and B_k are probability distributions accumulated in the forward and backward directions along the trellis, according to the following updating relations:

$$A_k(s) = \sum_{e:\, s^E(e)=s} A_{k-1}[s^S(e)] \cdot P_k[u(e);I] \cdot P_k[c(e);I]$$

$$B_k(s) = \sum_{e:\, s^S(e)=s} B_{k+1}[s^E(e)] \cdot P_{k+1}[u(e);I] \cdot P_{k+1}[c(e);I]$$

In practice, expensive multiplications are avoided by working in the log domain and using the E-function operator [7]. Introducing the following definitions:

$$\pi_k(c;\mathrm{I}) \equiv \log[\mathrm{P_k(c;I)}]$$
$$\pi_k(u;\mathrm{I}) \equiv \log[\mathrm{P_k(u;I)}]$$
$$\pi_k(u;\mathrm{O}) \equiv \log[\mathrm{P_k(u;O)}]$$
$$\alpha_k(s) \equiv \log[A_k(s)]$$
$$\beta_k(s) \equiv \log[B_k(s)]$$

the previously presented equations of forward and backward metric calculations take the form of:

$$\alpha_k(s) = \underset{e:\, s^E(e)=s}{E} \left\{ \alpha_{k-1}[s^S(e)] + \pi_k[u(e);\,\mathrm{I}] + \pi_k[c(e);\,\mathrm{I}] \right\}$$

$$\beta_k(s) = \underset{e:\, s^S(e)=s}{E} \left\{ \beta_{k+1}[s^E(e)] + \pi_{k+1}[u(e);\,\mathrm{I}] + \pi_{k+1}[c(e);\,\mathrm{I}] \right\}$$

Then, the output metric of the SISO is calculated as follows:

$$\pi_k(u;\,\mathrm{O}) = \underset{e:\, \mathrm{u(e)=u}}{E} \left\{ \alpha_{k-1}[s^S(e)] + \pi_k[c(e);\,I] + \beta_k[s^E(e)] \right\}$$

The traditional decoding is performed by computing first the β metrics for the entire block (going backwards) and storing them. When all the values of β are available, the α values and output metrics $\pi_k(u;\ \mathrm{O})$ are computed (going forward). Adopting graphical representation introduced in [10], the algorithm is shown in Figure 2. Note the graphical notations from Table 1.

Graphical Notations	
	Input of the intrinsic (channel) information and the extrinsic information from the previous decoding stage to SISO.
	Dummy α state metrics calculation (no store).
	Dummy β state metrics calculation (no store).
	Valid β state metrics calculation and store.
	Valid α metrics and SISO output metrics, $\pi_k(u;\ \mathrm{O})$, calculation.

Table 1: Graphical Notations

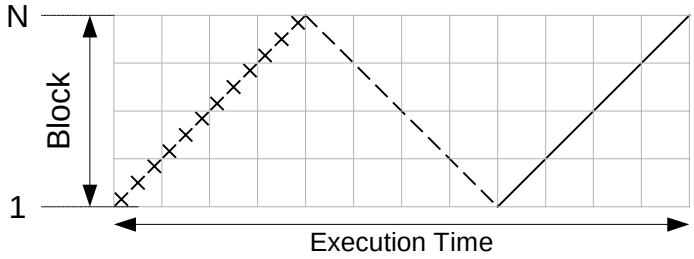

Figure 2: Traditional SISO Algorithm

Storing all β metrics requires an unacceptable amount of memory. Therefore the sliding window approach is used [4][8][9]. The backward (and/or forward) metrics are initialized at an intermediate point instead of at the end (or at the beginning, for forward metrics) of the block. The degradation due this optimization is negligible when an appropriate intermediate point (sufficient window size) is used [4][8]. It is implemented using *sliding windows*.

Decoding with the sliding window is performed as follows. The block is divided into windows of size WL. For each window, initial values of α and β are calculated. The initial α values are the last values of α of the previous window, and the initial β values are calculated by dummy β metrics calculation over the next window (initial values of β for the dummy β calculation is arbitrary). Note that the initial values of α may be also calculated by dummy α metrics calculation over the previous window. Decoding with sliding window is shown in Figure 3.

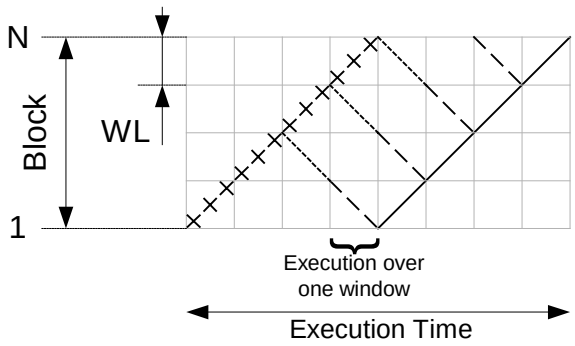

Figure 3: Sliding Window SISO Algorithm

When tailbiting termination is employed, the last WL bits of the block (*tail window*) are sent to the SISO prior to the whole block send. The SISO performs dummy α calculation over this window in order to get initial α values for the first window of the block. The initial β values for the last window are calculated and stored during the valid β state metrics calculation over the first window. Note that the cost of using tailbiting is additional latency of WL cycles per decoding iteration. The decoding with sliding window and tailbiting is shown in Figure 4.

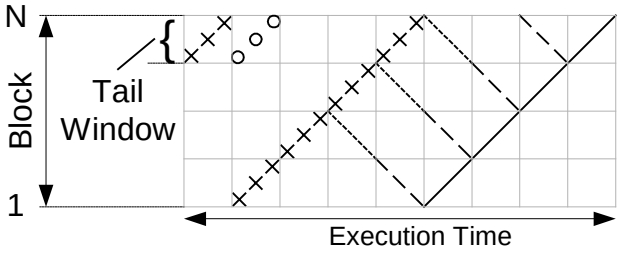

Figure 4: Sliding Window with Tailbiting SISO Algorithm

3. Sequential Decoder Architecture

The detailed scheme of an iterative decoder is depicted in Figure 5.

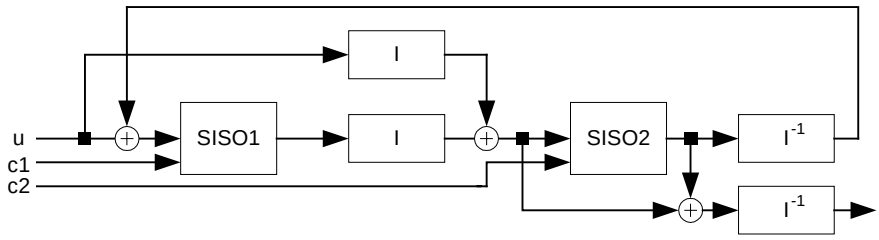

Figure 5: Iterative Decoder Detailed Scheme, I denotes interleaver

One iteration of the decoder can be divided into two stages:

i. "*Interleaving Stage*". The result of the previous iteration plus u, and c1 bits are processed by SISO1. The result is passed through interleaver I.

ii. "*DeInterleaving Stage*". The extrinsic data from Interleaving Stage plus an interleaved version of u, and c2 are processed by SISO2. The result is passed through deinterleaver I^{-1}.

The order of operation of the two decoding stages is the same: Add, compute (SISO) and de/interleave. When CE1 and CE2 are identical, SISO1 is identical to SISO2. In addition, the same memory unit can perform the interleaving and deinterleaving processes while suitable interleaver/de-interleaver addresses are provided. Therefore, the above two stages can be implemented by the same hardware block, used twice for each iteration.

In order to decode a block the following *decode resource* hardware modules are needed: A SISO, an interleaver memory, an adder, memories for channel data (u, c1, c2), an interleaving address memory and control logic. When parallel processing of, say, *n* blocks is required to achieve higher data rate, the entire decoding resource should be duplicated *n* time. Interleaving addresses can be generated by logic, instead of storing them in memory, but a memory provides flexible interleaver design. All resources can share the same interleaving address memory, using appropriate FIFOs.

The maximal input rate, $F_{in}^{Uncoded}$, for the sequential architecture (with input double buffer) is:

$$F_{in}^{Uncoded} = \frac{NR}{2 \cdot NI} \cdot F_{\text{int}}^{eff} \qquad (1)$$

$$\text{Where } F_{\text{int}}^{eff} = \frac{F_{\text{int}} \cdot N}{N + 5 \cdot WL}$$

where

NR: number of resources,

NI: number of iterations,

N: block size,

WL: window length,

F_{int}^{eff} : effective processing rate, and

F_{int} : internal clock rate.

For given silicon area the throughput of the decoder, $F_{in}^{Uncoded}$, depends on the number of decode resources that can be placed on that area. The area efficiency of sequential and parallel architectures is discussed below.

Latency of the sequential architecture is that of the decode resource:

$$D_{Seq} = \frac{2 \cdot NI \cdot (N + 5 \cdot WL)}{F_{\text{int}}} \qquad (2)$$

The latency consist of the SISO delay due to prior input of five windows (as evident in Figure 4) in addition to processing N metrics of the block, all multiplied by number of iterations.

4. Parallel Decoder Architecture

The parallel decoding architecture applies all available SISOs in parallel to one incoming block (Figure 6). The decomposition of the processing to sub-blocks is facilitated by the sliding windows technique which allows independent decoding of sub-blocks without degradation in error-correction performance [8]. Dummy α and dummy β metrics are calculated in order to determine the initial values of α and β for each sub-block, providing flexibility of choosing size and position of the sub-block.

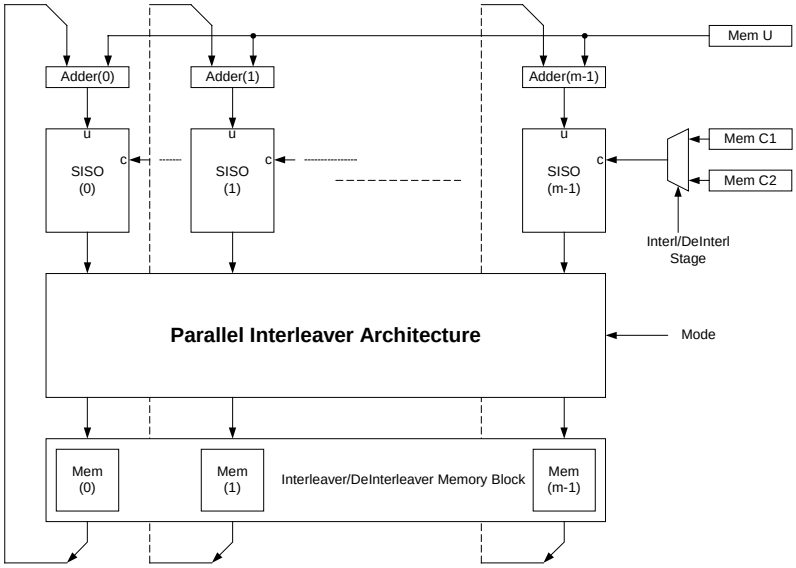

Figure 6: Parallel Decoder Architecture

The block is divided into *m* sub-blocks and each sub-block is sent to a separate SISO. The operations are the same as in the sequential architecture: Add, compute (SISO) and de/interleave. After SISO processing, the extrinsic metrics are sent (in sets of *m* metrics each) to the *Parallel Interleaver* where they are permuted and sent to the interleaver memory (Interleaver / Deinterleaver Memory Block in Figure 6). The interleaver memory, as well as the channel data memories (Mem U, Mem C1, Mem C2) consist of arrays of *m* memories, each of depth N/m.

Parallel decoding significantly reduces the processing latency. Figure 7 explains parallel decoding graphically, comparing it to sequential decoding.

Decoding is performed with tailbiting. For each sub-block the initial α metrics are calculated over the last window of the previous sub-block (for the first sub-block, 'last window' means last window of the last sub-block thanks to tailbiting). Initial β values for the last window of a sub-block are β values received for the first window in the next sub-block (these values are computed and stored during the first window of each sub-block calculation and thus are ready when the calculation is performed over the last window of a sub-block).

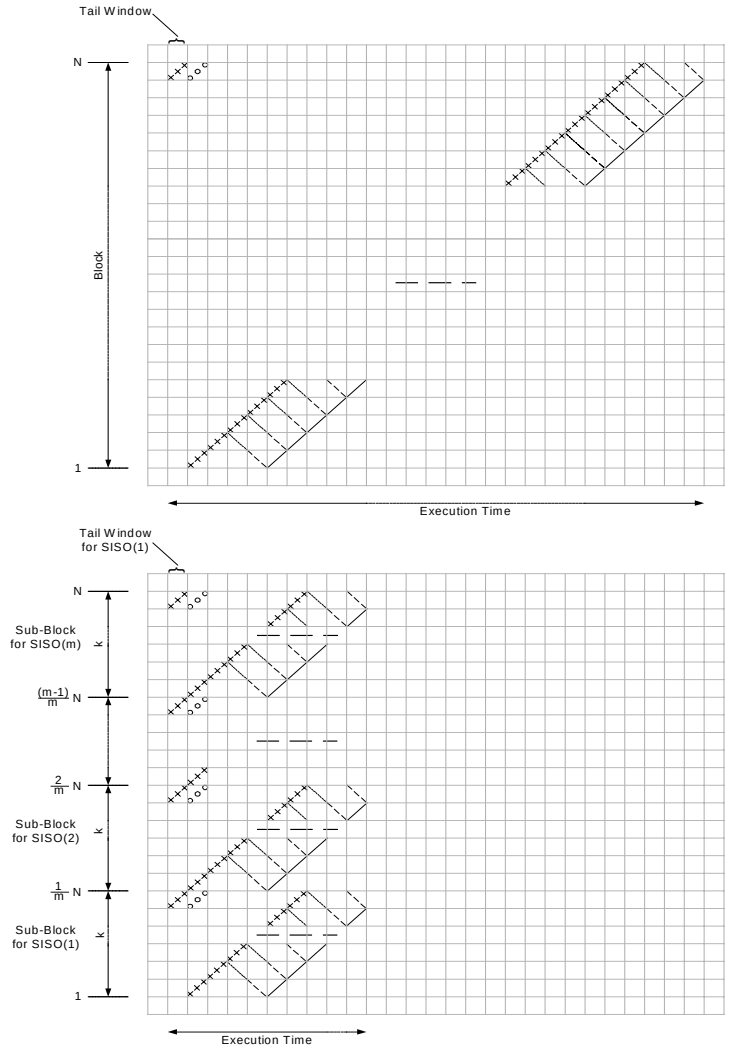

Figure 7: Sequential and Parallel Decoding

The maximal input rate formula for the parallel architecture is the same as (1) except for NS (Number of SISOs) substituted for NR:

$$F_{in}^{Uncoded} = \frac{NS}{2 \cdot NI} \cdot F_{\text{int}}^{eff} \tag{3}$$

The latency, however, is calculated differently and depends on NS:

$$D_{Par} = \frac{2 \cdot NI \cdot (N / NS + 5 \cdot WL)}{F_{\text{int}}} \tag{4}$$

Parallel and sequential architectures are compared in terms of latency and throughput for given silicon area. Performance, as can be seen from equations (1)-(4), is highly correlated to NR and NS. These values are higher for more efficient use of the area. Parallel architectures are more area efficient thanks to the fact that for each additional SISO, no additional memories and almost no additional logic are required, whereas for the sequential architecture, the entire decode resource is duplicated, including memories and SISO, as described above.

The analysis was performed using Synopsys 2000.05, Passport Libraries, 0.35µ (slow) tools. The ratio of throughput $F_{in}^{Uncoded}$ of the parallel architecture to that of sequential one for different block sizes vs. area is shown in Figure 8. It can be seen that for larger chip area, the parallel architecture can handle larger blocks more efficiently, and higher input data rates are accommodated.

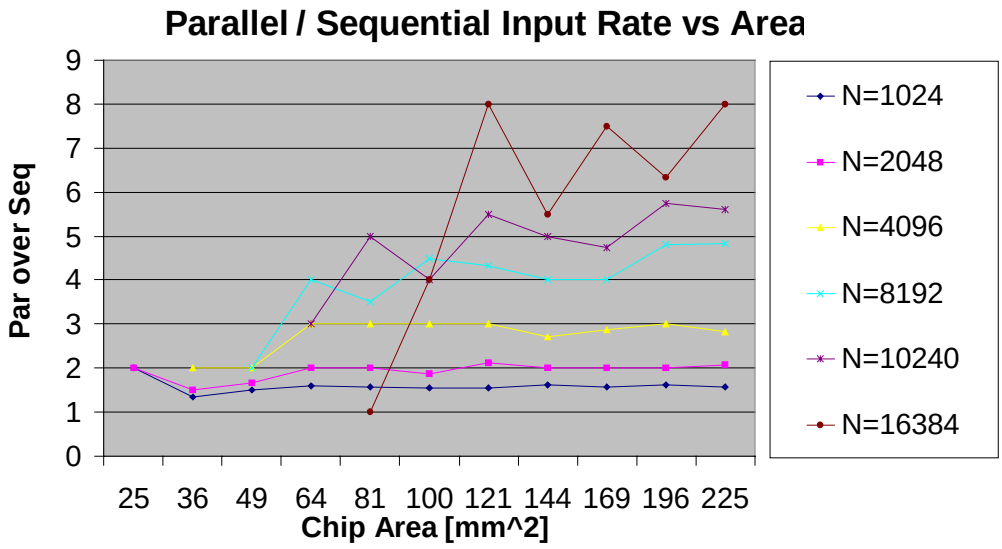

Figure 8: Throughput Ratio Results

The latency decrease results are summarized in Figure 9. The latency of the parallel architecture is significantly decreased to 6-62% of the sequential latency. As in the case of throughput, the larger the block size and the larger the chip area, the better the parallel architecture.

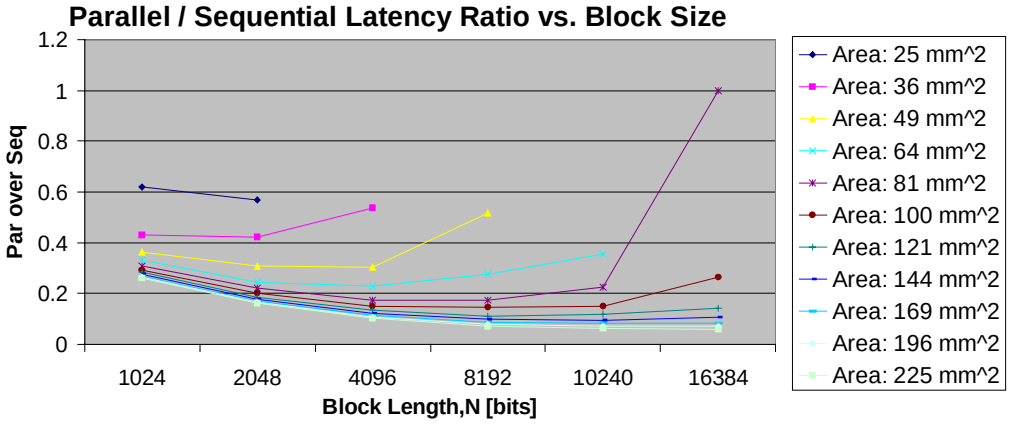

Figure 9: Latency Ratio Results

The parallel interleaver plays key role in the performance of the entire parallel decoder. The task is to permute in parallel at least *m* metrics coming simultaneously from *m* SISOs. The parallel interleaver is parameterized by *m*, the number of inputs/outputs of the interleaver, and *d*, the interleaver delay (Figure 10).

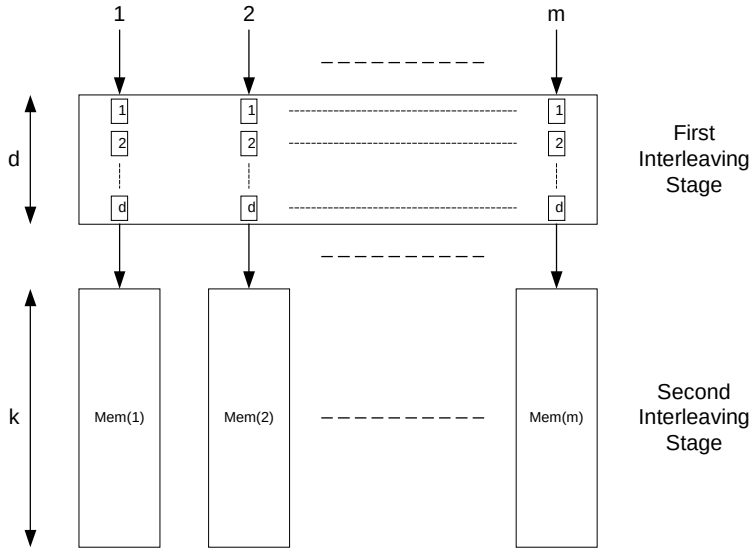

Figure 10: The Parallel Interleaver

The interleaving consists of two stages: Each of $d \cdot m$ metrics from the various SISOs are permuted by the First Interleaving Stage, and are subsequently permuted again in the target memories (Mem(1)-Mem(m)) of the Second Interleaving Stage.

The Parallel Interleaver spread [11], dispersion, and area/interconnect consumption characteristics, and their impact on coding gain are under study at the present.

5. Conclusions

A parallel turbo decoder architecture has been presented. A significant latency reduction was achieved (up to a factor of 16) in comparison with a sequential turbo decoder. In addition, it was found that the parallel architecture is more area efficient, improving throughput up to a factor of 8 for the same chip area.

Current research on this subject focuses on detailed design and analysis of the Parallel Interleaver and additional optimizations of the total latency of the decoder by decreasing SISO latency.

References

[1] C.Berrou, A.Glavieux, and P.Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes”, Proceedings of ICC’93, Geneva, Switzerland, pp.1064-1070, May, 1993.

[2] Berrou, and A.Glavieux, “Near Optimum Error Correcting Coding And Decoding: Turbo-Codes”, IEEE Transactions on communications, vol. 44, no. 10, October 1996.

[3] Peter A.Beerel, and Keith M.Chugg, “A Low Latency SISO Application to Broadband Turbo Decoding”, IEEE Journal on selected areas in communications, Vol.19, No.5, May 2001.

[4] G.Masera, G. Piccinini, M.R.Roch, and M.Zamboni, “VLSI Architectures for Turbo-Codes”, IEEE Transactions on VLSI Systems, Vol. 7, No.3, Sep.1999.

[5] Stewart Crozier, “Turbo-Code design Issues: Trellis Termination Methods, Interleaving Strategies, and Implementation Complexity”, Communications research Center (CRC), Ottawa, Canada (www.crc.ca/fec), ICC, June, 1999.

[6] Claude Berrou, “Additional information on the EUTELSAT/ENST-Bretagne proposed channel turbo coding for DVD_RCS”, DVD-TM, Ad Hoc Group on Return Channel over Satellite, 6th meeting, Geneva, 28-29 July, 1999.

[7] C.Schurgers, F.Catthoor, M.Engles, “Optimizing MAP Turbo Decoder”, IEEE, 2000.

[8] Andrew Hunt, Stewart Crozier, Mark Richards, and Ken Gracie, “Performance Degradation as a Function of Overlap Depth when using Sub-Block Processing in the Decoding of Turbo Codes”, Communications research Center (CRC), Ottawa, Canada (www.crc.ca/fec).

[9] Sorin Adrian Barbulescu,"On Sliding Window and Interleaver Design", Institute for Telecommunication Research, Univercity of South Australia, Mawson Lakes SA 5095, Australia.

[10] C.Schurgers, F. Catthoor, and M.Engels, "Memory Optimization of MAP Turbo Decoder Algorithms". IEEE Transactions on VLSI Systems, Vol. 9, No.2, April 2001.

[11] Stewart N.Crozier, "New High-Spread High-Distance Interleavers for Turbo Codes", Communication Research Center, Canada, stewart.crozier@crc.ca.