A Pipelined Architecture for High Throughput Efficient Turbo Decoder

S. M. Karim
Indian Institute of Technology
Kharagpur

Girish Mahale
IBM India Pvt Ltd.
Bangalore, India

Indrajit Chakrabarti
Indian Institute of Technology
Kharagpur

ABSTRACT

This paper presents a new pipelined architecture of Turbo decoder which runs at nearly four times the speed of a recently reported architecture with a reasonable increase in hardware. The proposed architecture is based on block-interleaved pipelining technique which enables the pipelining of the add-compare-select-offset (ACSO) kernels. Moreover next iteration initialization (NII) method has been adapted in the proposed work to initialize sliding window border values. The decoder chip consumes 219.8 mW of power at a maximum operating frequency of 192.3 MHz when implemented using 0.18 µm CMOS technology. Synthesis results indicate that the designed turbo decoder can achieve a decoding throughput of 38.46 Mb/s with an energy efficiency of 1.14 nJ/ bit/ iteration at the maximum operating frequency. The proposed architecture is therefore considered suitable for a real time wireless application such as video-telephony in mobile networks.

Keywords:

Iterative turbo decoder, high speed architecture, sliding window, block interleaved pipelining, pipelined ACSO.

1. INTRODUCTION

Turbo codes [1] have been adopted in various standards for wireless communication systems, such as 3GPP [2], W-CDMA [3], CCSDS [4], IEEE 802.16, and DVB-RCS [5] due to the outstanding performance in terms of bit error rate (BER) at very low signal-to-noise ratio (SNR).

Turbo decoder as shown in Figure 1 consists of two constituent decoders, known as soft-input soft-output (SISO) decoders, which communicate iteratively through an interleaver/de-interleaver. The SISO decoders calculate the log-likelihood ratios (LLR) of each of the two component codes using *maximum a posteriori probability* (MAP) algorithm, often simplified to MAX-Log-MAP or Log-MAP algorithm for efficient implementation in VLSI circuit. The throughput and complexity of turbo decoders and receivers are mostly determined by the SISO decoders. This is why active research in design of high-throughput MAP decoders has been undertaken.

The decoding of turbo code is based on the utilization of the *a priori* information of the component code iteratively. The *a priori* information for one component code is obtained by permuting the extrinsic information derived from the LLR values of the other component code. In an alternate cycle, the original codeword followed by the interleaved codeword is getting decoded by the component decoder. The intermediate soft values obtained from one iteration are used for the next iteration. Finally, the decision is made after running the iterations completely. Due to this iterative process, it is difficult to achieve a high throughput with the conventional turbo decoders used in the recent applications. Applications of a heuristic approach to reduce the total number of iterations and modification of architectural design have been recently reported [6] to alleviate the throughput bottleneck of the turbo decoder.

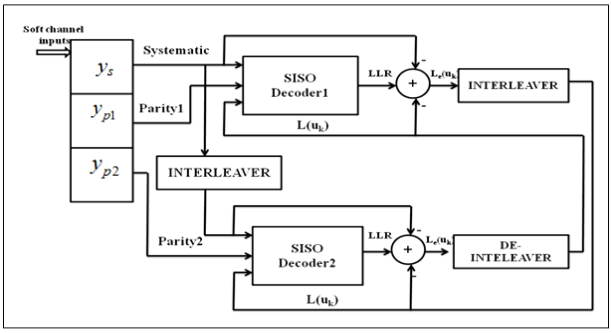

Figure 1. Decoding structure for turbo code

Reduction of the critical path delay of the computational unit paves the way for attaining high throughput. Add-Compare-Select-Offset (ACSO), the computational kernel in SISO decoder, calculates the state metric values in recursive manner which poses restrictions on the reduction of critical path delays. ACSO can be pipelined with minimal silicon area overhead to reduce the critical path delays and hence the system operating frequency can be increased to yield high throughput. However, pipelining the ACSO structure requires the data blocks to be processed independently. The warm-up property allows data dependency to be circumvented by dividing data frame into sub blocks with appropriate border values.

The MAP algorithm [1] was used in the original turbo decoding. However, as it is too complicated to be implemented in VLSI, it gave way to the computationally less intensive Max-Log-MAP [7] and Log-MAP algorithms [7] that operate in the logarithmic domain. In these two algorithms, the multiplications are replaced with additions, and the exponentiations disappear. To improve the decoding speed, sliding window MAP algorithm [8] was introduced to the turbo decoding.

Researchers have extensively delved into the feasibility of achieving high-throughput implementations of turbo decoder. These high-throughput architectures are based on parallel processing [9]-[11], look-ahead computation [12]-[14] and pipelined architecture [14]-[15].

Parallel processing and look-ahead computation achieve high throughput at the cost of complexity and silicon area. But the pipelined technique presented in [14] demonstrates the use of pipelining technique to enhance the throughput with minimal impact on silicon area. It employs two levels of pipelined ACSO in the form of block-interleaved pipelining (BIP) [14]-[15] at the architectural level to design a high-throughput MAP

decoder. Increasing the number of pipeline stages can further reduce the critical path whereas the complexity remains within a tolerable increase. The proposed work aims at increasing the level of pipelining at the ACSO and subsequent stage of design in order to decrease the critical path delays. However, the consequent increase in latency due to the high level of pipelining is not desirable. With a view to making the proposed design suitable for real time applications like wireless video communication the sliding window technique along with the next iteration initialization (NII) [16] method has been utilized in the present work to circumvent further delay in calculating the dummy backward metrics.

The remainder of this paper is organized as follows. Section II describes the sliding window Log-MAP algorithm. Section III describes the pipelined ASCO kernel and data flow for block interleaved sliding window Log-MAP procedure with sliding window next iteration initialization method. The implementation of the pipelined Log-MAP decoder architecture is described in Section IV, followed by an ASIC implementation of the Log-MAP Turbo Decoder chip in Section V. Finally, conclusions are drawn in section VI.

2. SLIDING WINDOW LOG-MAP ALGORITHM

A. LOG-MAP Algorithm

A Log-MAP decoder [17] computes the branch metrics $\gamma_k(s',s)$, forward state metrics $\alpha_{k+1}(s)$, backward state metrics $\beta_{k-1}(s')$ and the log-likelihood ratio (LLR) $L(u_k)$ using the following equations

$$\gamma_k(s',s) = \frac{1}{2}[u_k L(u_k) + L_c \sum_{l=1}^{n} y_{kl} x_{kl}] \quad (1)$$

$$\alpha_{k+1}(s) = \max^*_{s'}[\alpha_k(s') + \gamma_k(s',s)] \quad (2)$$

$$\beta_{k-1}(s') = \max^*_{s}[\beta_k(s) + \gamma_k(s',s)] \quad (3)$$

$$L(u_k) = \max^*_{s',\, s:\, u_k=1}[\alpha_k(s') + \beta_k(s) + \gamma_k(s',s)] - \max^*_{s',\, s:\, u_k=0}[\alpha_k(s') + \beta_k(s) + \gamma_k(s',s)] \quad (4)$$

where k is a trellis index, u_k is the data at the k^{th} trellis index, s and s' are trellis states. $L(u_k)$ is a priori information, x_k denotes the transmitted codeword, y_k denotes the received codeword, n is the number of parity bits, and L_c is the channel reliability value.

The sign bit of the parameter $L(u_k)$ (LLR) decides whether the transmitted bit u_k was *+1* or *-1*, whereas its magnitude represents the confidence of the decision taken. The $\max^*(x,y)$ operation is defined as

$$\max^*(x,y) = \max(x,y) + \ln\left(1 + e^{-|x-y|}\right) \quad (5)$$

The $\max^*(x,y)$ can be implemented by an add-compare-select-offset (ACSO) unit as shown in Figure 2. A small look-up table (LUT) is used to provide the correction term to be added with the state metric values.

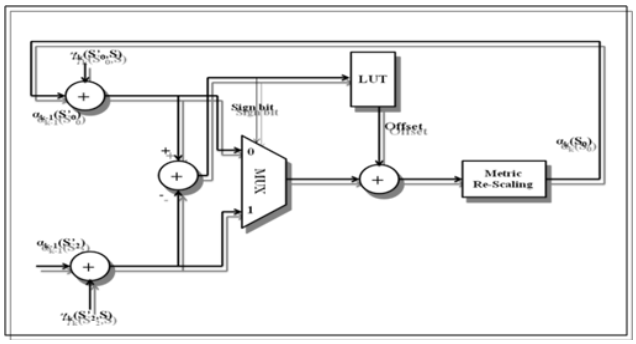

Figure 2. Conventional ACSO Architecture

B. Sliding Window Data Flow Graph

The present work is based on the sliding-window log-MAP algorithm [8] as it minimizes the metric storage requirements. The sliding window Log-Map algorithm (SW Log-MAP) can be derived via the warm-up property which states that the forward and the backward metrics α_k and β_k converge after a few constraint lengths have been traversed in the trellis, independent of the initial conditions. The warm-up period (L) [8] is normally taken as four times the constraint length. In sliding window technique, the warm-up property can be employed for computing the backward metrics as shown in the data flow graph in Figure 3, where the warm-up and the computation processes are depicted using dashed and solid lines, respectively.

Sliding window is based on three recursion units, two of which are used for backward recursion (RU_{B1} and RU_{B2}) and the third is used for forward recursion (RU_A) unit. In the data flow graph, the vertical axis represents the processing time expressed in units of a symbol period. The horizontal axis represents the received symbol or trellis time. Figure 3 describes how the L symbols $\{Y_k\}_{L<k\leq 2L}$ are decoded. Over the duration from time $t=L$ to $2L-1$, RU_{B1} performs L recursions, starting from Y_{3L-1} to Y_{2L}. The state vector β_k is initialized with the all-zero value and after processing L symbols, convergence is reached and β_{2L} is obtained. Next, between $t=2L$ and $3L-1$, RU_{B2} starts from state β_{2L} to compute β_{2L-1} down to β_L. The vectors $\{\beta_k\}_{L<k\leq 2L}$ are stored in the state vector memory unit for LLR computation. Finally, between $t=3L$ to $4L$, the recursion unit RU_A generates the vectors $\{\alpha_k\}_{L<k\leq 2L}$ and the vector β_k corresponding to the computed α_k is extracted from the memory in order to compute LLR $L(u_k)$. This process is repeated after every L cycles.

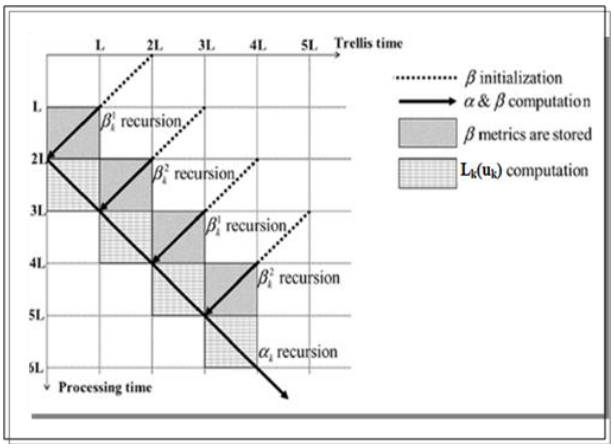

Figure 3. Data flow graph of the sliding-window log-MAP decoder with the assumption that the computation and warm up periods are equal to L

3. PIPELINED ACSO UNIT

A pipelined architecture for the ACSO unit is shown in Figure 4. In ACSO, the look-up table (LUT) is implemented by combinational logic. To prevent arithmetic overflow and to reduce hardware complexity, metric normalization schemes are employed by the metric re-scaling block in the ACSO. At each time instant, one checks if any of the state metrics (α or β) is larger than 2^{q-2}, where q represents the word length of the state metrics. If one of the state metrics is larger than 2^{q-2}, then 2^{q-2} is subtracted from all the state metrics (α or β). The state metrics remain same if their values are well below the maximum value that can be represented by the word length [14]. The ACSO is divided into four parts by inserting the q bit registers, and the critical path is therefore broken into shorter ones. This technique improves the clock frequency up to four times compared to the conventional ACSO. Registers should be inserted at proper location such that the critical path can be divided into almost equal parts.

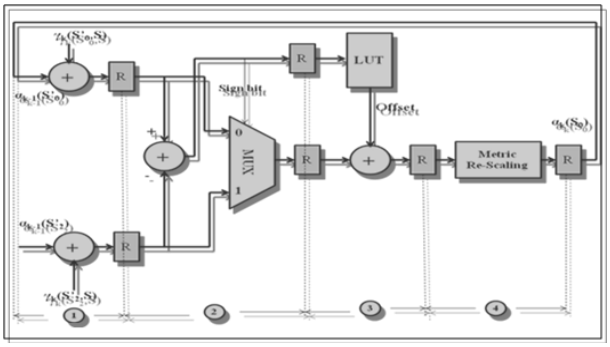

Figure 4. The pipelined ACSO Architecture; R denotes register

ACSO units with two and four stages of pipelining have been synthesized in *Synopsys DesignVision* using *Faraday 0.18μm* library. Table I depicts the critical path length and the maximum frequency obtained in two cases. It may be observed that by the four level pipeline of the ACSO block, reduction of the critical path can be achieved by a factor nearly equal to four compared to the conventional ACSO block.

TABLE I. Comparison of Pipelined ACSO Units

ACSO type	critical path (ns)	max. frequency (MHz)
Conventional ACSO	6.8	147.06
Two level pipelined	3.6	277.78
Four level pipelined	1.8	555.55

However, incorporating the warm up property in the sliding window increases the latency for the pipelined ACSO structure. This unwanted delay can be circumvented by using the warm-up free β calculation (initialized by the previous iteration β values) in the sliding window Log-Map algorithm. It is the sliding window next iteration initialization method [13], in which the pointer generated by backward recursion at the iteration k is used to initialize the backward recursion at $k+1th$ iteration. The β values corresponding to the previous iteration are stored in memory and in the next iteration, these are used to start calculation of the backward metric

Figure 5 shows the corresponding data flow graph for the block of size $N=16L$ with pipeline level $M=4$ such that the block is divided into few sub blocks each of size $4L$. The forward state metrics are calculated within a sub block, while the backward state metrics are recursively computed within a sliding window. The backward state metric β_L obtained from the trellis section between $2L$ and L in the previous iteration is used to start β calculation for the trellis section between L and 0. The initial values of the backward state metrics for all sub blocks are obtained in a similar fashion. The forward state metric values at the end of each sub block are used to start the forward recursion of the next sub block. For example, α values at $4L$ trellis stage in the 1st sub block will be used to start the forward recursion process of the 2nd sub block in the next iteration. The forward and backward recursions were initialized by using zero vectors for the starting iteration.

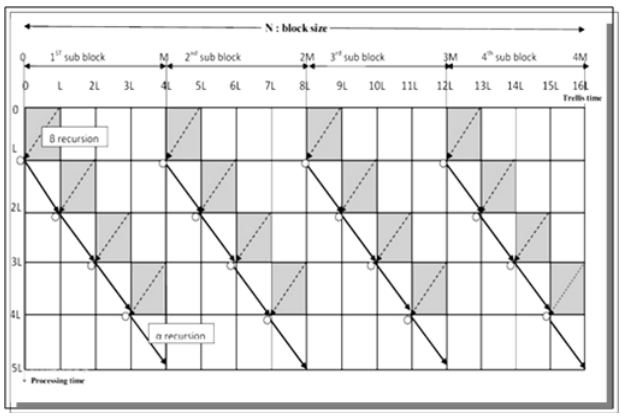

Figure 5. Data-flow graph of sub block interleaved MAP computation for sliding window with warm-up free β calculation

4. ARCHITECTURE OF THE PIPELINED LOG-MAP DECODER

The architecture of SISO (soft input soft output) decoder for Log-Map turbo decoder chip is shown in Figure 6.

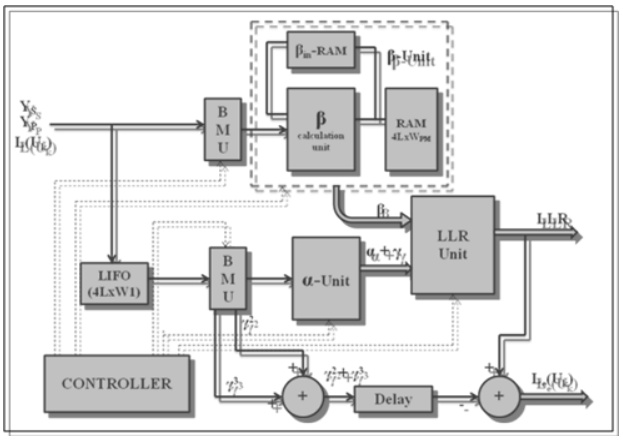

Figure 6. Architecture of the SISO decoder chip where W_I and W_{pm} denote the input data precision and the path-metric precision respectively

The SISO architecture is implemented following the block interleaved pipelining technique and using the sliding window Log-MAP algorithm with warm-up free *backward metric* calculation, as shown in data flow graph (vide Figure 5). The proposed SISO decoder is composed of α *unit,* β *unit,* γ *unit,* and *LLR unit* all of which are pipelined. The β *unit* has one $4L \times W_{pm}$ *RAM* to store the β values and one β_{in} RAM to store the initial values of β to be used to initialize the β calculation for the next iteration. Another storage unit of size $4L \times W_I$ is required in the form of LIFO (where W_I is the input data precision). The LIFO buffer is required to reorder the input sequence into the α *unit*, since the β *unit* operates in a time reversed order

C. Branch Metric Calculation unit

The branch metrics are computed based on the knowledge of the input and the output associated with the branch transition from one state to another. To reduce the memory, the branch metric unit (BMU) only stores the γ_k^{B-j} and the other branch metrics can be generated from the stored γ_k^{B-j} where $j=0, 1, ..., 2^m-1$, $B=2^{m+1}-1$ and m is the order of the associated encoder memory. The word length of the BMU can be reduced to $2^m \times n_{bm}$ bits [15] where n_{bm} stands for number of bits to represent a single branch metric.

D. Forward and Backward State Metric (SM) Calculation module

.The structure of a forward state metric calculator (α *unit*) bears resemblance to that of the backward state metric calculator (β *unit*). However, the α *unit* provides the output of sum of the forward state metric and the branch metric to the LLR-unit for LLR calculation. Whereas, the β *unit* gives the backward state metric values, which have been stored in memory. Based on the precision of the calculation and the hardware complexity, the state metric is represented by a 9 bit unsigned number. The α *unit* and β *unit* updates the state metrics in parallel employing 2^{K-1} ACSO kernels, where K is the constraint length. Along with the *β-calculation unit*, the β *-unit* also has 2^{K-1} LIFO (each LIFO has the size of $4L \times W_{pm}$) so that each LIFO stores the intermediate $4L$ backward path metrics (L from each sub block) for each state. The β_{in} *RAM* (whose size is equal to the number of windows in each sub block multiplied by $2W_{pm}$) is used to store the initial values of β to start the backward state metric calculation in the next iteration.

5. ASIC IMPLEMENTATION OF MAP DECODER

The MAP decoder has been implemented using a Standard cell based design methodology. It uses *Faraday 0.18 µm* library, targeting the *UMC 0.18 µm six-metal Mixmode/RFCMOS* Process. The MAP decoder has been implemented in Verilog and its functionality has been verified using a Synopsys register transfer level (RTL) simulator. Table II shows the area and power requirement for the designed component and the whole of the turbo decoder. Each sub module in the RTL behavioral description of the MAP architecture has been translated into a gate-level netlist using a Synopsys DesignVision tool.

TABLE II. Area and power for the proposed pipelined Turbo Decoder

Design Module	Area (mm^2)	Power (mW)
SISO Decoder	0.45	38.4
Interleaver–Deinterleaver memory	0.25	19.1
Input buffer	0.52	62.6
MAP Decoder without input buffer	1.33	157.2
Turbo Decoder	1.85	219.8

The corresponding layout as depicted in Fig. 7 has been generated using Cadence SOC Encounter tool. Layout-versus-schematic (LVS) and design-rule-check (DRC) have been followed by static timing verification using Synopsys PrimeTime analyzers at the sub block and full chip levels. Power has been calculated using Synopsys PrimePower.

TABLE III. Pipelined turbo Decoder IC Characteristics

Turbo decoder	[18]	[14]	This work
Technology	0.25 µm	0.18 µm	0.18 µm
Supply Voltage (Volts)	2.5	1.8	1.8
Core Size (mm^2)	8.9	8.7	9.65
Max. System Clock (MHz)	135	285	192.3
Max. throughput (Mbps)	5.48	27.6	38.46
Power Consumption (mW)	n.a	330	219.8
Energy efficiency (nJ/b/iteration)	6.98	2.36	1.14

Table III summarizes the key characteristics of the designed pipelined turbo decoder architecture and provides comparison with two important existing turbo decoder implementations.

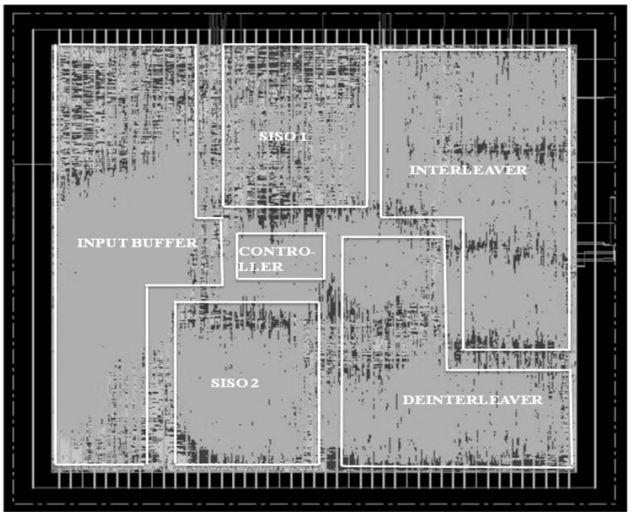

Figure 7. Chip layout diagram of pipelined turbo decoder

6. CONCLUSIONS

This paper presents high throughput architecture for Log-MAP Turbo decoder by combining block interleaved pipelining (BIP) and sliding window techniques with next iteration initialization method. Moreover, the design has been synthesized and in a 1.8V, 0.18 μm CMOS process. It has been demonstrated that the proposed architecture achieves the goal of high throughput turbo decoders. High-throughput operation has been achieved via four level block-interleaved pipelining of the ACSO kernel. The latency is also reduced by warm up free backward metric calculation. The new architecture speeds up the decoding process with a tolerable increase in hardware resource. The designed MAP decoder core consumes an area of 9.65 mm^2 and can achieve a decoding throughput of 38.46 Mb/s with a latency of 11.3μs with five iterations. The interleaver size is 512 bit. The chip consumes 219.8 mW of power at 1.8-V supply when operating at maximum frequency of 192.3 MHz. The proposed design is deemed appropriate for real time wireless video applications.

7. REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," *in Proc. IEEE Int. Conf. Communications, Geneva, Switzerland*, pp. 1064–1070, May 1993.

[2] Third Generation Partnership Project, "3GPP home page," www.3gpp.org.

[3] Japan's Proposal for Candidate Radio Transmission Technology on IMT-2000: W-CDMA [Online]. Available: http://www.arib.or.jp/IMT-2000/proponent.

[4] *Telemetry Channel Coding*, Consultative Committee for Space Data Systems (CCSDS), Blue book 101.0-B-4, May 1999.

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

[6] J. Kaza and C. Chakrabarti, "Design and implementation of low-energy turbo decoders," *IEEE Trans. VLSI Systems*, vol. 12, no. 9, pp. 968–977, Sep. 2004.

[7] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. Telecomm.*, vol. 8, no. 2, Mar.-Apr. 1997.

[8] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the map algorithm," *IEEE Transactions on Communications*, vol. 51, no. 2, pp. 175-185, Feb 2003.

[9] G. Prescher, T. Gemmeke, and T. Noll, "A Parameterizable Low-Power High-Throughput Turbo-Decoder," in *Proc. 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*, Philadelphia, Pennsylvania, USA, pp. V–25–28, Mar. 2005.

[10] D. Gnaedig, E. Boutillon, J. Tousch, and M. Jezequel, "Towards an optimal parallel decoding of turbo codes," in *Proc. 4nd International Symposium on Turbo Codes & Related Topics*, Apr. 2006.

[11] O. Muller, A. Baghdadi, and M. Jezequel, "From Parallelism Levels to a Multi-ASIP Architecture for Turbo Decoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 92–102, Jan. 2009.

[12] T. Miyauchi, K. Yamamoto, and T. Yokokawa, "High-performance programmable SISO decoder VLSI implementation for decoding turbo codes," in *Proc. IEEE Global Telecommunications Conf.*, vol. 1, pp. 305–309, 2001.

[13] M. Bickerstaff, L. Davis, C. Thomas, D. Garret, and C. Nicol, "A 24 Mb/s radix-4 LogMAP turbo decoder for 3 GPP-HSDPA mobile wireless," in *IEEE ISSCC Dig. Tech. Papers*, pp. 150–151, 2003.

[14] S. Lee, N. Shanbhag, and A. C. Singer, "A 285 MHz pipelined MAP decoder in 0.18 um CMOS", *IEEE J. Solid-State Circuits*, vol. 40, no. 8, pp. 1718–1725, Aug. 2005.

[15] Seok-Jun Lee, Naresh R. Shanbhag, and Andrew C. Singer, "Area-efficient high-throughput MAP decoder architectures," *ICASSP*, pages 25–28, 2005.

[16] J. Dielissen and J. Huiskens, "State Vector Reduction for Initialization of Sliding Windows MAP," in *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, Brest, France, pp. 387–390, Sep. 2000.

[17] J. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: An overview", *IEEE Trans. Vehicular Technology*, vol 49 no. 6, pp. 2208–2233, Jun 2000.

[18] M. C. Shin and I. C. Park, (2007). "SIMD processor-based turbo decoder supporting multiple third-generation wireless standards", *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 15, no. 7, pp. 801–810, Jul. 2007.