POWER- AND AREA-EFFICIENT SINGLE SISO ARCHITECTURE OF TURBO DECODER

Dongwon Lee and Wayne Wolf
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332
dwlee@gatech.edu and wolf@ece.gatech.edu

ABSTRACT

In this paper, we propose a power- and area-efficient architecture of Turbo decoder. In order to improve the non-functional performance metrics such as power consumption and area, we use the trade-off method between bit error rate (BER) performance and the two non-functional performance metrics. Our proposed architecture shows about 16.7% reduction in power consumption and about 22.5% reduction in area compared to the general architecture.

Index Terms— trade-off method, data dependency, parallelism, sub-optimal initialization, single SISO architecture

1. INTRODUCTION

In digital wireless communication systems, channel coding is used to restore the damaged bits to the original bits. Turbo coding is one of the best channel coding schemes in terms of bit error rate (BER) performance [1]. When it was invented, Turbo coding was not practically used because of its computational complexity and large decoding latency. But as some applications such as wireless communication and satellite communication require better BER performance, Turbo coding starts being considered again. Many papers have been published to improve the non-functional performance: area, power, decoding latency, and throughput [2, 3, 4]. This paper introduces a new architecture that reduces the power consumption and area at the cost of only modest losses in BER.

In order to improve the non-functional performance metrics such as power, area, and throughput, we trade off BER degradation. First, the power consumption can be reduced by removing the necessary but non-critical computations. Second, if the computations are done in separate functional units, those units can be eliminated, which results in the reduced area as well. The BER degradation due to the removal of the computation (and the corresponding units) should be tolerable. Finally, the throughput can be improved by exploiting some parallelism which exists in Turbo decoding sequence. In some cases, the exploitation of parallelism requires the removal of data dependency. In those cases, first of all, data dependency between computations in Turbo decoding sequence should be analyzed. Then the BER degradation is measured when the dependency is removed. If the degradation can be acceptable, the parallelism is exploited. In this paper, we focus on the power consumption and area, and use the trade-off approach to improve them.

We use the Log-Maximum *a posteriori* (Log-MAP) [5] and sliding window algorithms [6] in this paper, which make the practical implementation of the MAP algorithm [7] possible. The Log-MAP algorithm reduces the computational complexity of the MAP algorithm. One of the most critical issues in implementing the MAP (including the Log-MAP) algorithm is the memory size required to store one of the forward and backward state metrics until Log-Likelihood Ratios (LLRs) are calculated. Depending on implementation, only one of the two state metrics is stored. Our proposed architecture stores the forward state metrics. In order to resolve the memory size problem, the sliding window algorithm is used in this paper [6].

This paper is organized as follows. Section 2 shortly describes the Log-MAP and sliding window algorithms. In Section 3 and 4, the trade-off method to improve the power, area, and throughput is described. In section 5, we propose a power- and area-efficient Turbo decoder architecture. We compare the power, area, and BER of our proposed architecture with those of the general architecture in Section 6.

2. LOG-MAP AND SLIDING-WINDOW ALGORITHMS

The Log-MAP algorithm is a derivative of the MAP algorithm, which reduces the pure MAP's computation complexity by converting multiplications and divisions to additions and subtractions respectively. Implementation of MAP-based algorithms including Log-MAP requires a large size of memory to store the forward (or backward) state metrics until the LLRs are calculated. This memory size problem is one of the main issues in practically implementing MAP-based algorithms.

The sliding window algorithm helps us resolve the memory size problem. As shown in Fig. 1, one frame is

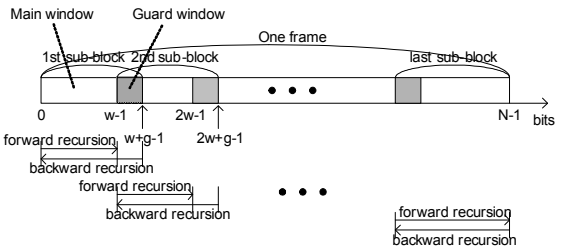

Figure 1. General sliding window algorithm

divided into several sub-blocks and each sub-block is decoded separately, where w represents the size of main window, g is the size of guard window, and N is the total frame size. Therefore, we need to store the forward (or backward) state metrics only for one small sub-block, instead of the whole frame.

There are two kinds of window in the sliding window algorithm: main window and guard window. As shown in Fig. 1, the forward recursion is accomplished only through the main window, but the backward recursion is carried out through both the main and guard windows. The backward recursion through the guard window (called dummy computation) is not for computing the LLRs, but for obtaining the reliable initial backward metric at the boundary of the main window and the guard window, e.g., at w-1, 2w-1, etc. The size of guard window is a key factor affecting BER performance; the larger the size, the better the BER performance. The size of guard window should be determined by trade-off between the memory size, the decoding latency, and the BER performance [8]. For simple implementation, the size of guard window can be set to be equal to that of main window [2, 3]; the beta computation unit for the main window can be also used for the guard window without any modification. In this paper, we use the equal-size window method.

3. HOW TO IMPROVE THROUGHPUT

Even though this paper does not focus on throughput, we describe the method to improve the throughput in this section. The throughput is one of the most important non-functional performance metrics. In Turbo decoding, the throughput is defined as the rate at which LLR values are produced. We can improve it by pipelining or parallelizing computations in Turbo decoding sequence. (pipelining can be viewed as one of the parallelizing)

Fig. 2 shows the decoding sequence of a non-pipelined architecture. The horizontal axis represents the trellis time and the vertical axis represents the processing time. The forward arrows indicate alpha (forward state metric) and gamma (branch metric) computations and the backward solid arrows indicate beta (backward state metric), gamma, and LLR computations. The dashed backward arrows indicate the beta and gamma computations, which are for

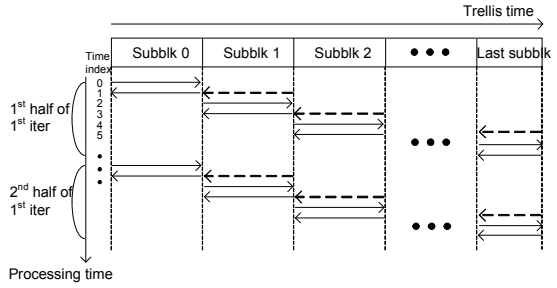

Figure 2. Non-pipelined decoding sequence

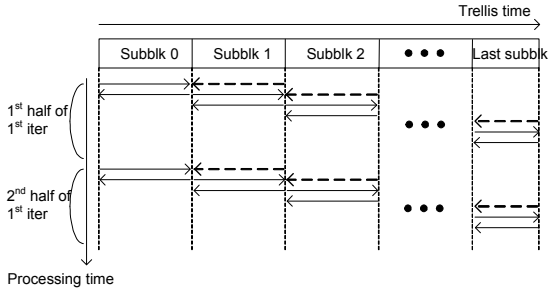

Figure 3. Pipelined decoding sequence

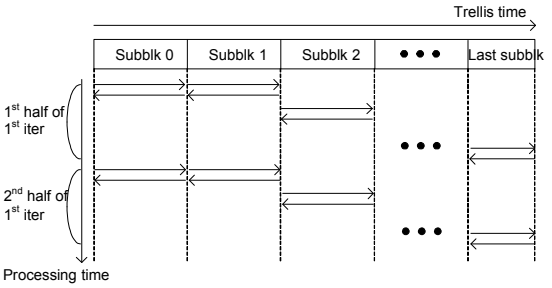

Figure 4. Parallel decoding sequence

getting the reliable initial beta metric at the boundary as described in section 2. (The equal-size window method is used here) As shown in Fig. 2, the forward and backward computations are done sequentially in the non-pipelined architecture. As a result, the LLR values are produced with interruption; for example, they are generated at processing time index 1, 3, 5, etc., but not generated at time index 2, 4, etc.

In the non-pipelined architecture, two kinds of data dependency limit the throughput: one is within each sub-block and the other is across adjacent sub-blocks. First, within each sub-block, computations across the iterations have true data dependency. For example, the 2nd half iteration of the 1st iteration can be started only after the 1st half of the 1st iteration is completed. And the 1st half of the 2nd iteration can be started only after the 2nd half of the 1st iteration is completed, and so on. This data dependency is due to the inherent nature of the iterative Turbo decoding scheme using the MAP algorithm; the extrinsic values of one component decoder are used as *a priori* information of

another component decoder. Since this dependency is very strong - the removal of the dependency results in a functional failure (huge BER degradation), there is no way to remove this data dependency. Therefore, the decoding sequence within each sub-block should be strictly observed. Because of this limitation, even if an infinite number of functional units are added, the maximum attainable throughput is still limited. Second, another type of data dependency exists across the adjacent sub-blocks. Since it is related to initial beta metric computation, we call it *initial value dependency* in this paper. Although the initial value dependency is also true data dependency, it is very weak – its removal only results in the modest BER degradation.

Fig. 3 shows one example of the decoding sequences with the improved throughput by using pipelining. Compared to Fig. 2, the decoding sequence of following sub-blocks is moved one processing time index ahead. In the middle of pipeline (neither prologue nor epilogue), the three computations corresponding to the backward solid arrow, forward solid arrow, and backward dashed arrow of different three sub-blocks respectively are done at the same time. As a result, the LLR values are generated without interruption and the throughput is about two times that of Fig. 2. In Fig. 3, the initial value dependency is still observed.

In some cases, the pipelining or parallelizing requires the removal of data dependency. In those cases, data dependency between computations should be analyzed and the BER degradation due to the removal of the dependency should be measured. If the BER degradation can be acceptable, the parallelism can be exploited. Fig. 4 shows the example to which the trade-off approach is applied in order to improve the throughput. The decoding sequence is the case with a parallelism level = 2, i.e. two sub-blocks are concurrently processed. The following decoding sequences are moved one more processing time index ahead compared to that of Fig. 3. Now the initial value dependency is removed. If the BER degradation due to the removal is tolerable, this parallelism can be exploited. The throughput is about four times that of the Fig. 2.

In this paper, we consider serial architecture, not parallel architecture of Turbo decoder. In other words, we consider the architecture corresponding to Fig. 3, not Fig. 4.

4. HOW TO REDUCE POWER CONSUMPTION AND AREA

In section 3, we described how throughput can be improved by the pipelining and parallelizing of computations. If needed, the trade-off approach between the BER performance and the throughput performance is used. In order to reduce the dynamic power consumption and area, we also use the trade-off approach in this section.

As described in section 3, the initial value dependency can be removed at the expense of modest BER losses; the dummy computations (corresponding to the backward

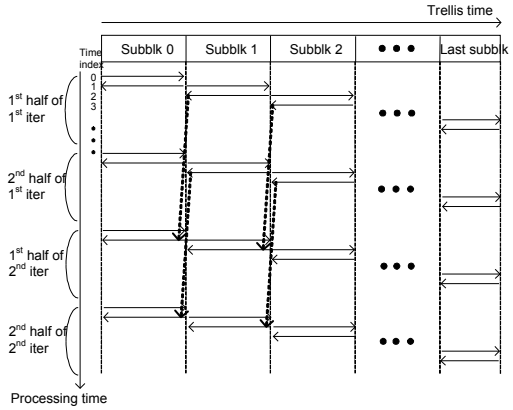

Figure 5. Decoding sequence with sub-optimal initialization

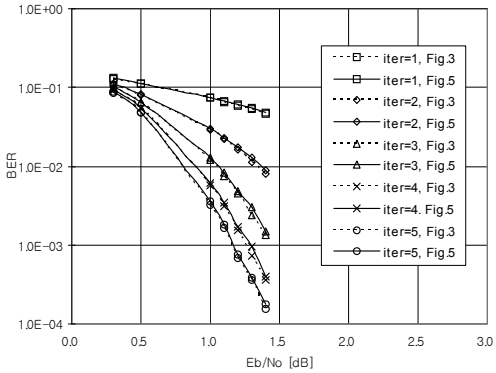

Figure 6. BER performance comparison

dashed arrows in Fig. 3) can be removed because their results are not directly used to get the LLR values, but to get the reliable initial beta metrics at the boundary (at the end point of each sub-block). All the dashed lines in Fig. 3 can be removed as in Fig. 5. Eliminating these computations definitely results in the reduced dynamic power consumption. Furthermore, the additional BCU and GCU related to the dummy computations are no longer required. The elimination of those units also causes the area to be lowered. This elimination, however, can lead to BER performance degradation, which might be acceptable. But if there are some ways to compensate the degradation, it is better to apply the methods than doing nothing. The larger sub-block size and more iteration number are examples of the compensation methods. But the both methods have a negative effect on the power consumption and area. Boutillon et al. showed another alternative solution, the sub-optimal initialization method in [9]. Although they explained the sub-optimal initialization method for parallel architecture, its concept can be also applied to this paper, serial architecture. In the sub-optimal initialization method, the initial beta metrics are referenced across the iterations. For example, as shown in Fig. 5, the final beta metric of sub-block 1 at the 1st half of the 1st iteration is used as the initial beta metric of sub-block 0 at the 1st half of the 2nd iteration,

and so on. The reference should be taken only within the same component decoder.

Fig. 6 compares the BER performance of Fig. 3 and Fig. 5 decoding sequences. The simulation parameters are set as follows: code rate = 1/3, constraint length = 3, frame length = 512 bits, sub-block length = 128 bits, and the number of iterations = 5. As shown in Fig. 6, the BER performance degradation of Fig. 5 compared to Fig. 3 is negligible over all E_b/N_0 and iterations. In this paper, the serial architecture corresponding to Fig. 3 is called the general architecture [10], and one corresponding to Fig. 5 is called the proposed architecture. To sum up, we propose a Turbo decoder architecture with the low power consumption and small area for the decoding sequence in Fig. 5.

5. TURBO DECODER ARCHITECTURE

The proposed architecture operates according to the decoding sequence in Fig. 5. In traditional Turbo decoders, several component decoders – mostly two - are used. The two separate Soft-In Soft-Out (SISO) decoders are corresponding to two component encoders respectively. But considering the utilization efficiency of the SISO units, the dual SISO-based scheme is inefficient because one of the two is always in the idle state in turn. For example, the 2nd SISO is always in the idle state during every 1st half iteration and the 1st SISO is always in the idle state during every 2nd half iteration. Therefore, we consider a single SISO scheme in this paper.

The block diagram of our proposed architecture based on the single SISO scheme is shown in Fig. 8. It is composed of ALU, memory unit, and controller. The ALU is general, but the memory unit and the controller are specific to our architecture. For showing the difference between the proposed architecture and the general architecture, the general one is also shown in Fig. 7.

5.1. ALU

Since we use the single SISO scheme and remove the dummy computations in the proposed architecture, only one Alpha Computation Unit (ACU) and one Beta Computation Unit (BCU) are required as shown in Fig. 8. The ACU and BCU are always in operation for two consecutive sub-blocks at the same time excluding at the prologue and epilogue of the pipeline processing. For example, at the time index 1 in Fig. 5, the ACU is working for the sub-block 1 and the BCU is working for the sub-block 0. For this pipeline processing, the gamma branch metrics should be provided to the both ACU and BCU at the same time. Therefore, two Gamma Computation Unit (GCU)s are required.

In the general architecture, the dummy computation is necessary, so one additional BCU and GCU are required as shown in Fig. 7.

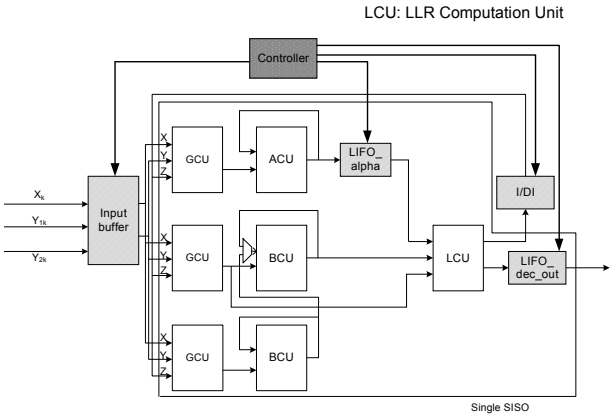

Figure 7. General single SISO architecture

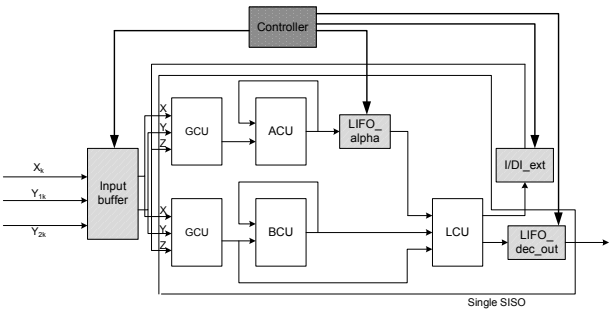

Figure 8. Proposed single SISO architecture

5.2. Memory

Three kinds of memory are used in terms of its function: input buffer, interleaver/deinterleaver (I/DI), and LIFO buffer. The input buffer stores the raw received information and parity bits. For continuous decoding, the buffer size should be twice the frame size; during the current frame decoding, the next frame is stored for the future processing. And since the code rate is 1/3 in this example, the input buffer is composed of six component buffers: two for the information bits, two for the first parity bits, and two for the second parity bits. The two for the information bits are shared between the in-order information bits and the interleaved information bits. Secondly, the I/DI is used to permutate the extrinsic values. One I/DI is shared between two component decoding (although we use the single SISO scheme, there are two component decoding procedures) by time-division multiplexing; when the single SISO is operating as the first component decoder, I/DI is used as deinterleaver. And when the single SISO is used as the second component decoder, I/DI is used as interleaver. Finally, there are two kinds of LIFO buffer: one for the alpha metrics and the other for the decoded bits. In this paper, since the alpha computation is first done and the LLR computation is done with the beta computation, a LIFO buffer for the beta metrics is not needed. The size of the two LIFOs is twice the sub-block length respectively, regardless

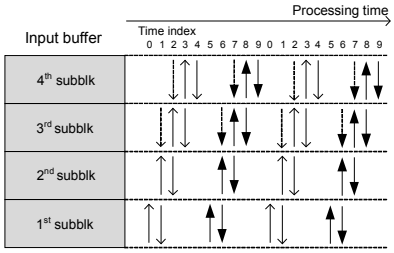

Figure 9. Read operations to input buffer in normal architecture

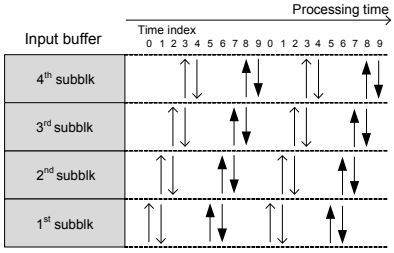

Figure 10. Read operations to input buffer in proposed architecture

of the total frame size because the sliding window algorithm is used in this paper.

The LIFO buffers are totally the same in the both general and proposed architecture because they are not affected by the dummy computation. But there is difference in the input buffer and I/DI. In order to show the difference, read operations to the input buffer are shown in Fig. 9 and 10 with the example of four sub-blocks. For simplicity, only one buffer (one of the six component buffers) is shown. The first type arrow (⟶) represents the in-order read operations and the other (⟶) represents the permuted-order read operations. As shown in Fig. 10, in the proposed architecture, the two memory accesses at most are needed at the same time: one for the forward data reading and the other for the backward data reading. Therefore, the input buffer can be implemented by using a dual-port memory with bidirectional ports. Since write and read operations are not occurred in overlapping segments of time, they can share the bidirectional ports (although write operation is not shown here). But in the general architecture, the three memory accesses at most are needed at the same time as shown in Fig. 9. Therefore, a triple-port memory with bidirectional ports is used to implement the input buffer. The bidirectional port can be shared between write and read operations as in the proposed architecture. This analysis is also applied to the I/DI.

5.3. Controller

The controller generates all control signals for all memories such as address, chip select, write enable, etc. As shown in Fig. 10, there are only ten states which correspond to the ten time indices, and the control signals to be generated at each state are unique. Therefore, we can design the controller as finite state machine (FSM) [11]. In addition, except for the prologue (time index 0 or 5) and the epilogue (time index 4 or 9), we can represent the three states in the middle (time index 1, 2, 3 or 6, 7, 8) as just one state because the required signal patterns are the same at the all three states. Therefore, even if the number of sub-blocks increases, the FSM-based controller is scalable because the required number of states is always six regardless of the frame length: one for the in-order prologue, one for the in-order middle, one for the in-order epilogue, one for the permuted-order prologue, one for the permuted-order middle, and final one for the permuted-order epilogue. The same design concept is applied to the control part design of the LIFO and I/DI.

6. EXPERIMENTAL RESULTS

We implement the proposed architecture in Verilog language and functionally verify it by comparing its output vector with the golden reference vector which is generated from our high-level C model. Then we synthesize it and measure the area in terms of equivalent gate count by using Xilinx XST tool [12], and measure the dynamic power consumption by using Xpower tool [13]. In order to show the advantages of the proposed architecture in terms of the power consumption and area, we also implement the general architecture.

The simulation parameters in this section are: code rate

Table 1. Equivalent gate count (frame length = 512 bits, sub-block length = 128 bits)

		General architecture	Proposed architecture
ALU	ACU	3,128	3,128
	BCU	3,128 x 2	3,128 x 1
	GCU	345 x 3	345 x 2
	LCU	2,992	2,992
Mem	Input buffer	80,081 x 6	61,361 x 6
	I/DI	80,081 x 2	61,361 x 2
	LIFO_ Alpha	26,155	26,155
	LIFO_ dec_out	2,444	2,444
Controller		7,143	5,389
Total		689,801 (100%)	534,814 (77.5%)

Table 2. Dynamic power consumption (mW) (freq = 188 MHz, Vccint = 1.2V, ambient temp = 25°C)

	General architecture	Proposed architecture
ALU	72.54	67.82
Memory	123.26	98.20
Controller	35.53	26.61
Total	231.33 (100%)	192.63 (83.3%)

= 1/3, constraint length = 3, frame length = 512 bits, and sub-block length = 128 bits. Table 1 shows the equivalent gate count with the fixed frame length and sub-block length. The proposed architecture shows 22.5% reduction in area compared to that of the general architecture, which is owing to three factors. First, the area of the ALU is reduced owing to the reduction in the required number of the BCU and GCU. Second, the area of the memory decreases because the number of access ports in the input buffer and I/DI decreases from three (triple-port memory) to two (dual-port memory). The area of the two LIFOs does not change. Finally, the area of the controller is reduced owing to the reduced number of control signals. All the three effects are fundamentally due to the fact that the dummy computations are removed in the proposed architecture.

Table 2 compares the dynamic power consumption. According to the synthesis result, the operating frequency is set to the maximum frequency = 188MHz, voltage = 1.2V, and ambient temperature = 25^{o}C. Table 2 shows that the total power consumption of the proposed architecture decreases by 16.7% compared to that of the general architecture. In order to see the effect of the increasing number of sub-blocks and iterations on the performance, we convert the power consumption into the energy consumption which is related to the battery life in mobile devices. As the number of sub-blocks increases, the required number of dummy computations also increases in the general architecture. Therefore, the energy consumption difference between the proposed architecture and the general one gets larger as the number of sub-blocks increases. Similarly, the difference also gets larger as the number of iterations increases.

7. CONCLUSION

In this paper, we proposed the power- and area-efficient architecture of Turbo decoder. In order to improve the non-functional performance metrics such as power consumption and area, we applied the trade-off approach between the BER performance and the non-functional performance. The proposed architecture is compared to the general architecture in terms of power consumption and area by using the experimental results. The proposed architecture shows about 22.5% reduction in area and about 16.7% reduction in power consumption compared to those of the general architecture. And as the number of sub-blocks or iterations increases, the percentage reduction in energy consumption of the proposed architecture compared to that of the general architecture gets larger.

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] S. Lee, N. R. Shanbhag, and A. C. Singer, "Area-efficient high-throughput MAP decoder architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 8, pp. 921-933, Aug. 2005.

[3] Z. Wang, Z. Chi, and K. K. Parhi, "Area-efficient high-speed decoding schemes for turbo decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 6, pp. 902-912, Dec. 2002.

[4] J. Kaza and C. Chakrabarti, "Design and implementation of low-energy turbo decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*., vol. 12, no. 9, pp. 968-977, Sep. 2004.

[5] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *in Proc. IEEE Int. Conf. Communications*, pp. 1009–1013, 1995.

[6] A. J. Viterbi, "An intuitive justification of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun*., vol. 16, pp. 260–264, Feb. 1998.

[7] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[8] M. Marandian, J. Fridman, Z. Zvonar, and M. Salehi, "Performance analysis of turbo decoder for 3GPP standard using the sliding window algorithm," *IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications*, vol. 2, pp. e127-e131, Sep. 2001.

[9] E. Boutillon, C. Douillard, and G. Montorsi, "Iterative Decoding of Concatenated Convolutional Codes: Implementation Issues," *Proc. of the IEEE*, vol. 95, no. 6, pp. 1201–1227, Jun. 2007.

[10] C. Lin, Y. Shih, H. Chang, and C. Lee, "A low power Turbo/Viterbi decoder for 3GPP2 applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 426-430, Apr. 2006.

[11] M. Hasan, T. Arslan, "A triple port RAM based low power commutator architecture for a pipelined FFT processor," *Proc. Int. Symp. Circuits and Systems*, vol. 5, pp. v353-v356, May 2003.

[12] Xlinx XST tool user guide 9.2i.

[13] Xlinx Xpower tutorial, ver. 1.3, Jul. 2002.