VHDL Implementation of Two-State Multiple Turbo Codes

Vikas Bhatia
Defense Electronics Applications Laboratory,
Dehradun-248001, India
Email: vikas.bhatia@deal.drdo.in

Adrish Banerjee
Department of Electrical Engineering,
Indian Institute of Technology Kanpur,
Kanpur-208016, India
Email: adrish@iitk.ac.in

***Abstract*—With increasing demand for different data rates and services for communication systems, reconfigurability is of utmost importance. Field Programmable Gate Arrays (FPGAs) provide the flexibility in operation and function by a simple change in the configuration bit stream. Low complexity turbo-like codes based on simple two-state trellis or simple graph structure results in decoder with low complexity. Two-state multiple turbo code is one such example. In this paper, we present the VHDL implementation of a 2-state multiple turbo code architecture targeted towards the Xilinx Vertex-5 FPGAs and compared its implementation with 8-state 3GPP turbo code in terms of hardware complexity and speed.**

I. Introduction

Turbo codes [1] are a class of capacity approaching parallel concatenated codes that have found applications in many communication standards . Several implementations for standard 8-state 3GPP turbo codes [2] have come up with different strategies covering issues related to decoding algorithm [3], [4], fixed point arithmetic [5], [6], low power techniques, memory management, hardware architectures, and different hardware platforms [7].

In literature [8]–[14] most of the implementations use the MAP algorithm [15] in logarithmic domain or some variant of log-MAP algorithm. These simpler variants are the MAX-Log-MAP, Constant-Log-MAP, Linear-Log-MAP and Scaled-Log-MAP algorithms. The designs support the fixed point arithmetic. With upto two fractional bit representations it has also been shown that the optimized quantization levels are 5-bits for LLR systematic data, 7-bits for LLR estimate and 9-bits for internal metrics to achieve optimum performance [6]. Metric normalization is generally done by subtracting the maximum or minimum value at each stage from all the values. The decoder stopping criteria is either fixed number of iterations or dynamic depending upon conditions which also leads to low power consumption. The interleaver/deinterleaver implementations are kept simpler using Look-Up Table (LUT) based approach to store the permuted addresses. The decoding structure is generally serial concatenation of component SISO decoders. However, higher throughputs can be achieved by performing decoding in parallel fashion. Throughput improvements can also be achieved by designing internal pipelined parallel computing structures and improved contention free memory access.

The paper is organized as follows. In section II, we describe low complexity multiple turbo codes. In section III, we give the VHDL implementation details of 2-state multiple turbo codes. Finally in section IV, we present post synthesis results, and conclude the paper.

II. Multiple Turbo Codes

Multiple turbo codes (MTC) are a class of parallel concatenated codes with three or more constituent encoders separated by multiple interleavers [16]. Multiple turbo codes provide us with more parameters to design an efficient error control coding scheme. Several approaches to low complexity turbo-like code designs based on very simple graph structures or 2-state trellises have been designed that results in low decoder complexity. It has been shown that 2-state multiple turbo codes outperform 8-state 3GPP turbo codes both in the waterfall and errorfloor regions [17], [18]. For multiple turbo codes using 2-state constituent encoders, there are only two possibilities, namely, an accumulator (ACC) $\left[1 \ \frac{1}{1+D}\right]$ encoder and a feed-forward (FF) $[1 \ 1+D]$ encoder. Figure 1 shows the encoder structure of a nonsystematic multiple turbo code that uses an asymmetric combination of four 2-state constituent encoders with three interleavers [18]. It employs a parallel concatenation of three ACC encoders and a FF encoder. The overall rate of this 4-parallel un-punctured nonsystematic code is R= 1/4. The resulting code is then punctured to rate 1/2. The puncturing pattern is shown in Figure 1. Using an EXIT chart analysis [19], [20], the authors in [21] have shown that the ACC encoder helps to achieve good initial extrinsic estimates, while the FF encoder aids in faster convergence. Constituent decoders for multiple turbo codes can operate in *parallel* at any given time. It was shown in [22] that parallel decoding will result in fastest convergence and one of the best performance among different decoding configurations.

Simulations studies conducted in [18] showed that the Bit Error Rate (BER) performance of the two-state codes is about 0.05-0.1 dB better than the 8-state 3GPP code in the waterfall region. The Frame Error Rate (FER) of the two-state 4-parallel multiple turbo code is one order of magnitude better than the 3GPP code. However, the 2-state multiple turbo codes typically require 4-6 more iterations to converge compare to the 8-state 3GPP standard at low SNRs.

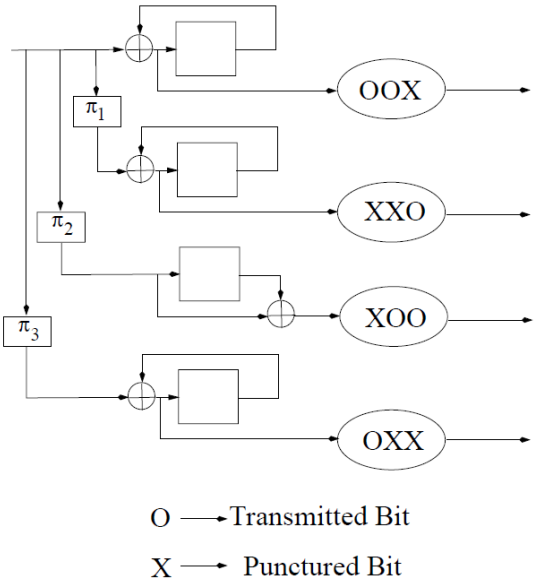


Fig. 1. R=1/2, 2-State 4-Parallel MTC Encoder

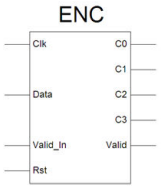
III. Implementation

The encoder and decoder are designed completely using synthesizable VHDL following structural hierarchy . The design to a large extent is parameterized with definitions in a separate VHDL design package. Moreover, the VHDL constructs do not use any macros that are device specific (Xilinx or Altera or any other vendor) and hence can be ported to any FPGA device.

The decoder design has the following features:

1) 5-bit soft representation for input log likelihood ratio (LLR) parity data, 6-bit for internal metrics and 6-bit for extrinsic information. No significant improvement in the performance was observed for further increase in number of bits used for representation. These numbers are less than what was suggested in [23] for turbo codes.
2) Scaled MAX-Log-MAP algorithm with scaling factor = 0.75. Since MAX-Log-MAP algorithm has simpler implementation, it was chosen for implementation. The performance was evaluated for different scaling factors for MAX-Log-MAP algorithm. The best performance was observed for scaling factor of 0.7. Since, scaling factor of 0.75 has simple implementation, it was chosen for implementation.
3) Fixed point arithmetic with 2 LSB representing fractional values.
4) Symmetric structure of trellis utilized for gamma (path metrics) computation reduces memory requirements to half.
5) In-built RAM storage for gamma and alpha (forward recursion) metrics and LLR estimates .
6) In-built LUT (look up table) based interleaving and de-interleaving operations
7) Dynamic normalization of alpha and beta (backward recursion) metrics reduces arithmetic complexity, storage requirements and power consumption.
8) Punctured systematic bits for rate 1/2 implementation reduce the arithmetic complexity significantly.
9) Latency (clock cycles) = $2 \times FrameLength + 5$

The encoder block interface in Figure 2 shows the input and output ports of the block and their descriptions.



Port	Type	Description
Clk	IN	Clock Input to the Block
Data	IN	Information Data to be Encoded
Valid_In	IN	Valid Strobe for Data Input, Active Low
Rst	IN	Reset port for the Block, Active High
C0,C1,C2,C3	OUT	Coded Output Data of the Encoder
Valid	OUT	Valid Strobe for Output Data, Active Low

Fig. 2. Encoder Block Interface

The detailed schematic diagram of the encoder is shown in Figure 3. The encoder structure comprises of four parallel

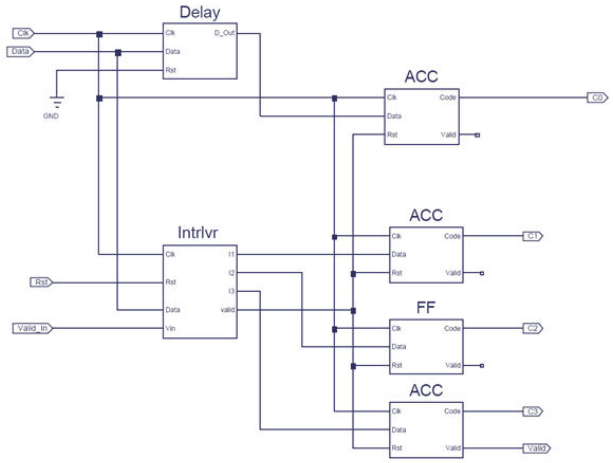
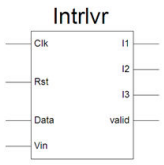


Fig. 3. Encoder Schematic

concatenated two-state accumulator (ACC) and the feedforward (FF) encoders in the ACC-ACC-FF-ACC configuration. The main sub-blocks of the encoder are ACC, FF, Interleaver and Delay. The delay block is basically used to delay the data by frame-length so that all four encoded streams are output in a synchronous fashion. The interleaver permutes the data bits before feeding to the other three encoders. The block interface for the interleaver is shown in Figure 4.

The design is based on random interleaving. A counter starts to increment as soon as data stream is input. As the counter progresses the input data is at first copied at the interleaved addresses stored in the look-up table (LUT). Once this process completes for the whole frame, the interleaved data starts to output as a stream. The main advantage of using LUT based interleaver is that we can change the interleaver algorithm any time and accordingly generate addresses and store them in an LUT, rather than having a particular logic



Port	Type	Description
Clk	IN	Clock Input to the Block
Rst	IN	Reset port for the Block; Active High
Data	IN	Data to be Interleaved
Vin	IN	Valid Strobe for Data Input; Active Low
I1,I2,I3	OUT	Interleaved Output Data for ENC1, ENC2 & ENC3
Valid	OUT	Valid Strobe for Output Data; Active Low

Fig. 4. Interleaver Block Interface

for interleaver. Moreover, there is no need for a separate design for deinterleaver. Only the addresses in the LUT have to be changed as per deinterleaving logic which can be pre-calculated using a software routine.

The block interface of the decoder is shown in Figure 5.

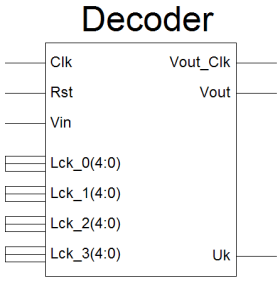


Fig. 5. Decoder Block Interface

The schematic diagram in Figure 6 shows the various component SISO decoders. Since the encoder configuration is ACC-ACC-FF-ACC, the configuration of the corresponding SISO decoders is also the same. The extrinsic information from each of the decoders is added in the ExInfo_Add module and fed to the component SISO decoders in the next iteration after suitable deinterleaving/interleaving process embedded within the block. All the interleaving operations on the received channel LLRs are assumed to be external to the decoder block and performed by a separate block that acts as a control unit to the whole decoding operation. The control unit also runs the decoder for a specified number of iteration. Each of the SISO decoder comprises of its own Gamma (path metrics), Alpha (forward recursion), Beta (backward recursion) and LLR calculation units along with their associated memories required for the metrics storage. Since for the current configuration of multiple turbo codes, no systematic bits are transmitted; hence at the decoder side they are treated as if all are punctured. In such a case the channel LLR for systematic bits are all assumed to be zero for calculations of various metrics. This in effect eases the requirement for various arithmetic operations and saves in the critical calculation paths. Each of the SISO unit produces interleaved versions of the extrinsic information to be fed to other three SISO component decoders.

The block interface of a SISO component decoder along with port definitions is shown in Figure 7.

The basic blocks in a SISO decoder are Gamma, Alpha,

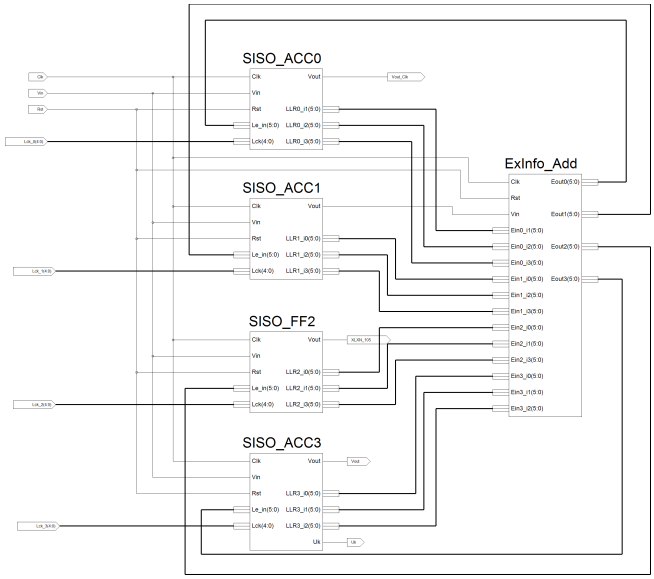


Fig. 6. Decoder Schematic

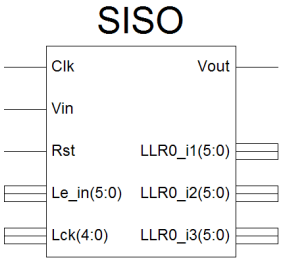


Fig. 7. SISO Block Interface

Beta and LLR calculation units (Figure 8). The Gamma Unit computes the branch metric γ in log-domain.

The Alpha unit computes the forward state metric α in log-domain.

Beta & LLR computation unit computes the backward state metric β and the final LLR of bit u_k in log domain.

The details of the computation that takes place in the Gamma, Alpha, Beta & LLR units along with their block interface is given in [24].

The block has in-built provision for storing the γ and α values in their respective RAMs. The LLR values are calcu-

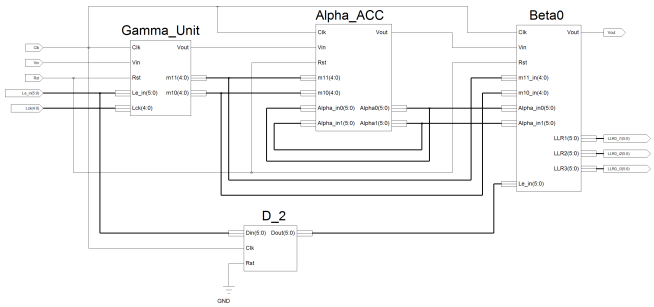


Fig. 8. SISO Schematic

lated simultaneously along with the β values during backward recursion and stored in a RAM. The final extrinsic information to be passed to other decoders is calculated by subtracting the previous iteration extrinsic information from the ones computed and stored in the RAM and further scaling by a factor equal to 0.75. The scaling factor is easily implemented by addition of 1-bit and 2-bit right shifted results of the value. Suitable deinterleaving/interleaving operation for those extrinsic values is also performed in this block to be passed to other three decoders.

During the VHDL development of the rate 1/2, 2-state 4-parallel multiple turbo codes, some modifications were done that led to an optimized implementation in terms of both speed and area. This was possible because of the simple trellis structure of 2-state ACC and FF encoders. For MAX-Log-MAP algorithm, while calculating forward and backward state metrics (α & β) only the relative difference between the different state metrics is significant. Hence, we did a *dynamic normalization* by subtracting the metric for *state 0* from both state metric values. This meant that the state metric for *state 0* was normalized to 0 and doesn't need any storage in the RAM and also did not appear in all the computations (addition/subtraction) in subsequent operations. This led to a significant reduction in memory storage requirement for metrics and lot of savings in arithmetic operations leading to faster logic implementation. These methods also enabled us to keep the metric values within the limits and the metrics were efficiently represented with minimum possible bits.

IV. Results

We present here the post synthesis complexity comparison chart for the encoders in Figure 9, a single SISO unit in Figure 10 and the full decoder in Figure 11 for a frame length of 256. An 8-state 3GPP decoder was designed and synthesized for comparison and to bring out the benefits of multiple turbo codes over the standard 8-sate 3GPP turbo code in terms of hardware complexity. The designs were targeted to the Xilinx Virtex-5 device family. The design tools used were Xilinx ISE 10.1 for design entry, ModelSim SE 6.1 for simulation and XST for synthesis.

S.No.	8-State 3GPP Turbo Code	2-State 4-Parallel Multiple Turbo Code
1.	9-bit adder : 1 Flip-Flops : 1046 9-bit comparator less : 1 1-bit 256-to-1 multiplexer : 1	9-bit adder : 1 Flip-Flops : 1051 9-bit comparator less : 1 1-bit 256-to-1 multiplexer : 3
2.	No. of Slice Registers : 794/32640 2% No. of Slice LUTs: 261/ 32640 No. of IOs: 9 No. of BUFG/BUFGCTRLs: 1/ 32 3%	No. of Slice Registers: 1369/32640 4% No. of Slice LUTs: 991/32640 3% No. of IOs: 9 No. of BUFG/BUFGCTRLs: 1/32 3%
3.	f_{MAX}: 211.332MHz	f_{MAX}: 314.733MHz

Fig. 9. Encoder Complexity Comparison

Shown in Figure 11 is the complexity comparison for the full decoder obtained from the synthesis reports. The simpler and less complex 2-state structure with optimizations led to a lower complexity implementation over the standard 8-state 3GPP Turbo code. We can see significant reduction in implementation complexity in terms of both the number and

S.No.	r=1/3, 8-State 3GPP Turbo Code	r=1/2, 2-State Multiple Turbo Code
1.	#Adders/Subtractors 12-bit subtractor : 1 11-bit subtractor : 16 11-bit addsub : 8 11-bit adder : 32 10-bit subtractor : 16 9-bit subtractor : 8 9-bit adder : 8 8-bit subtractor : 1 8-bit adder : 6 **Total: 96**	#Adders/Subtractors 8-bit subtractor : 1 7-bit subtractor : 5 7-bit adder : 6 6-bit subtractor : 2 6-bit adder : 17 **Total: 31**
2.	#Counters 9-bit up counter : 2 8-bit down counter : 1 **Total: 3**	#Counters 9-bit up counter : 2 **Total: 2**
3.	#Comparators 11-bit Comparator(>) : 22 9-bit Comparator(>) : 9 8-bit Comparator(>) : 1 8-bit Comparator(<=) : 1 **Total: 33**	#Comparators 9-bit comparator(>) : 1 9-bit comparator(<=) : 1 8-bit comparator(>) : 1 7-bit comparator(>) : 6 **Total: 9**
4.	#Multiplexers 11-bit 256-to-1 : 1 10-bit 257-to-1 : 8 **Total: 9**	#Multiplexers 1-bit 256-to-1 : 6 1-bit 257-to-1 : 7 **Total: 13**
5.	Device Utilization No. of Slice Reg : 70% No. of Slice LUTs : 34%	Device Utilization No. of Slice Reg : 11% No. of Slice LUTs : 24%
6.	f_{MAX} : 51.880 MHz	f_{MAX} : 116.271 MHz

Fig. 10. Single SISO Unit Complexity Comparison

S.No.	r=1/3, 8-State 3GPP Turbo Codes	r=1/2, 2-State Multiple Turbo Codes
1.	Macro Statistics: #Adders/Subtractors 12-bit subtractor : 2 11-bit subtractor : 32 11-bit addsub : 16 11-bit adder : 64 10-bit subtractor : 32 9-bit subtractor : 16 9-bit adder : 19 8-bit subtractor : 6 8-bit adder : 12 #Counters 9-bit up counter : 7 8-bit down counter : 2 #Comparators 11-bit Comparator(>) : 44 9-bit Comparator(>) : 18 9-bit Comparator(>=) : 3 9-bit Comparator(<) : 3 8-bit Comparator(>) : 2 #Multiplexers 11-bit 256-to-1 : 2 10-bit 257-to-1 : 16 8-bit 256-to-1 : 4 7-bit 256-to-1 : 2 5-bit 256-to-1 : 1 1-bit 256-to-1 : 24	Macro Statistics: #Adders/Subtractors 8-bit subtractor : 4 7-bit adder : 24 7-bit subtractor : 20 6-bit adder : 78 6-bit subtractor : 8 # Counters 9-bit up counter : 8 # Comparators 9-bit comparator(>) : 4 9-bit comparator(<=) : 4 8-bit comparator(>) : 4 7-bit comparator(>) : 24 # Multiplexers 1-bit 256-to-1 : 18 1-bit 257-to-1 : 28
2.	Device Utilization: No. of Slice Reg : 59587 (>100%) No. of Slice LUTs : 29268 (90%)	Device Utilization: No. of Slice Reg : 14686 (44%) No. of Slice LUTs : 31767 (97%)
3.	f_{MAX} : 51.880 MHz	f_{MAX} : 116.089 MHz

Fig. 11. Decoder Unit Complexity Comparison

the type of arithmetic units required for the implementation. Moreover, the f_{MAX} obtained after synthesis is considerably higher.

The overall latency of our implementation of the decoder for the 2-state multiple turbo codes design is given by Equation1:

$$L = FL \times 2 + 5 \tag{1}$$

where, L is the latency for the decoder in number of clock cycles/iteration and FL is the frame length.

Hence, for frame length = 256 the latency is 517 clock cycles/iteration. This means operating at f_{MAX} =116.089 MHz, the decoding time per iteration is equal to 517 * f_{MAX}^{-1} = 4.4534 $\mu s/iteration$. It may be noted the decoding time is dependent upon the frame length. As the frame length increases, the decoding time also increases per iteration. The calculation of the throughput in terms of bits/s for a total of *I* iterations of the decoder proceeds as per the Equation 2 :

$$T = \left[\frac{f}{L \times I}\right] \times FL \tag{2}$$

The total metrics memory requirement for a single component SISO decoder is given by Equation 3:

$$\begin{aligned} M \quad = \quad & [\gamma \times FL \times States + \alpha \times FL \times States \\ & + LLR \times FL] \times No\_Of\_SISO\_Units \end{aligned} \tag{3}$$

For our current implementation, frame length *FL*=256 and total decoder iterations *I*=14. Hence, the throughput is:

$$T = \left[\frac{116.089}{517 \times 14}\right] \times 256 = 4.11 \ \ Mbps$$

Since, for the 2-state decoder, only one state metric storage is required due to dynamic normalization and 5-bits are used for Gamma, and 6-bits each for Alpha and Extrinsic values, the total metrics memory requirement comes to be: M =[5*256*2 + 6*256*1 + 6*256] * 4 = 22.528 kbits. In contrast, the 8-state 3GPP decoder requires M =[(8+9)*256*8 + 11*256] * 2 = 75.264 kbits.

It can be seen that the implementation costs are significantly reduced in case of 2-state multiple turbo codes as compared to the standard 8-state 3GPP Turbo code. This is mainly because of the savings in the memory due to dynamic metric normalization. The normalization reduced the metric storage requirements for 2-state 4-parallel turbo codes significantly since values for only one of the two states need to be stored. The logic burden was thus reduced and this resulted in smaller and faster logic implementation. As a result, the whole decoder design could be easily fitted into one device as compared to the 8-state 3GPP Turbo code.

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error correcting coding and decoding: Turbo codes. In *Proceedings of the IEEE International Conference on Communications*, pages 1064–1070, Geneva, Switzerland, May 1993.

[2] 3rd Generation Partnership Project. Technical specification group radio access network: Multiplexing and channel coding (FDD). 3GPP TS 25.212 V3.1.0, 1999.

[3] W. E. Ryan. *Wiley Encyclopedia of Telecommunications*, chapter Concatenated Codes and Iterative Decoding. John Wiley and Sons, 2003.

[4] J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. In *Proceedings of the IEEE GLOBECOM*, volume 3, pages 1680–1686, Dallax, Texas, nov. 1989.

[5] M. A. Castellon, I. J. Fair, and D. G. Elliott. Fixed-point turbo decoder implementation suitable for embedded applications. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, Saskatoon, CA, may 2005.

[6] G. Montorsi and S. Benedetto. Design of fixed-point iterative decoders for concatenated codes with interleavers. *IEEE Journal on Selected Areas in Communications*, 19(5):871–882, 2001.

[7] G. Masera. *Turbo Code Applications: A Journey from a Paper to Realization*, chapter VLSI for Turbo Codes, pages 347–382. Springer, 2005.

[8] M. C. Valenti and J. Sun. The UMTS turbo code and an efficient decoder implementation suitable for software-defined radios. *International Journal of Wireless Information Networks*, 8(4):203–215, oct. 2001.

[9] Y. Tong, T. Yeap, and J. Chouinard. VHDL implementation of a turbo decoder with log-map-based iterative decoding. *IEEE Transactions on Intrumentation and Measurement*, 53(4):1268–1278, aug. 2004.

[10] M. J. Thul and N. Wehn. FPGA implementation of parallel turbo-decoders. In *Proceedings of the Symposium on Integrated Circuits and Sytems Design*, pages 198–203, Brazil, sep. 2004.

[11] W. Tang. A low-power implementation of turbo decoders. Master's thesis, Dept. of Electrical Engineering at Linkoping University, apr. 2007.

[12] W. J. Gross and P. G. Gulak. Simplified MAP algorithm suitable for implementation of turbo decoders. *IEE Electronics Letters*, 34(16):1577–1578, 1998.

[13] B. Classon, K. Blankenship, and V. Desai. Turbo decoding with the constant-Log-MAP algorithm. In *Proceedings of the 2nd International Symposium on Turbo Codes & Related Topics*, pages 467–470, Brest, France, sep. 2000.

[14] S. Papaharalabos, P. Sweeney, and B. G. Evans. Constant log-MAP decoding algorithm for duo-binary turbo codes. *IEE Electronics Letters*, 42(12):709–710, jun. 2006.

[15] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. In *IEEE Transactions on Information Theory*, volume IT-20, pages 284–287, March 1974.

[16] D. Divsalar and F. Pollara. Multiple turbo codes. In *Proceedings of the 14th Military communications conference, MILCOM*, pages 279–285, 1995.

[17] P. C. Massey and D. J. Costello Jr. New low-complexity turbo-like codes. In *Proceedings of the IEEE Information Theory Workshop*, pages 70–72, Cairns, Australia, September 2001.

[18] D. J. Costello Jr., A. Banerjee, C. He, and P. C. Massey. A comparison of low complexity turbo-like codes. In *Proceedings of the 36th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 2002.

[19] S. ten Brink. Convergence of iterative decoding. In *Electronic Letters*, volume 35, pages 806–808, May 1999.

[20] S. ten Brink. Convergence behavior of iteratively decoded parallel concatenated codes. In *IEEE Transactions on Communications*, volume 49, pages 1727–1737, October 2001.

[21] C. He, A. Banerjee, D. J. Costello Jr., and P. C. Massey. On the performance of low complexity multiple turbo codes. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, October 2002.

[22] J. Han and O. Y. Takeshita. On the decoding structure of multiple turbo codes. In *Proceedings of the IEEE International Symposium on Information Theory*, page 98, Washington D.C., June 2001.

[23] P. C. Massey and D. J. Costello Jr. Turbo codes with recursive non-systematic quick-look-in constituent codes. In *Proceedings of the IEEE International Symposium on Information Theory*, page 141, Washington, D.C., June 2001.

[24] Vikas Bhatia. VHDL implementation of two-state multiple turbo codes. Master's thesis, Department of Electrical Engineering, Indian Institute of Technology, Kanpur, may 2009.