

An FPGA Implementation of a Memory Efficient, Low Complexity Turbo Decoder Architecture for TETRA Release 2 Application

Kasra G. Nezami^{1,2}, Stuart D. Walker¹, Peter W. Stephens² and Martin Fleury¹

¹University of Essex, Department of Electronic Systems Engineering, Wivenhoe Park, Colchester, Essex, CO4 3SQ, U.K., Tel.: +44 (0)1206 872413, Fax: +44(0)1206 872900

² Sepura Ltd, Radio House, St. Andrew's Rd, Cambridge, CB4 1GR, U.K., Tel.:+44 (0) 1223 876000
Email: {knezam|stuwal|fleum}@essex.ac.uk, {kasra.nezami|peter.stephens}@sepura.com

Abstract – The Terrestrial Trunked Radio (TETRA) Release 2 communication standard supports different channel bandwidths and modulation techniques, and has adopted turbo code as its error coding scheme. Turbo code's inherent high complexity and long latency presents a practical problem for DSP implementation of this algorithm for TETRA Release 2 applications. In this paper, we present a low complexity, memory efficient, parallel architecture for turbo decoding on an FPGA platform, which addresses all the requirements by the standard at a low development cost, as well as low power consumption.

1. Introduction

Turbo codes, first presented in 1993 [1], are able to approach the Shannon capacity limit in an additive white Gaussian noise channel (AWGN). For this reason, TETRA release 2, an ETSI proposed standard for private mobile radio communication [2], has adopted turbo-coding techniques. The decoding in turbo code is an iterative process, whereby a Log-Likelihood-Ratio (LLR) is calculated for every bit in the data frame [1], [3].

The Maximum a posteriori (MAP) [1], which generates the LLR, is a complex algorithm and impractical to implement. The prospect of using turbo codes in communication systems was made possible by a simplified version of the MAP algorithm, known as Log-MAP. In order to achieve near Shannon capacity limit performance, the decoder needs to work on relatively long buffers of data. This coupled with its iterative operation results in high complexity, long latency and a requirement for large amounts of memory for DSP implementation of the algorithm.

As we show in the paper, a DSP solution will become very costly, as a chip is required to run at extremely high clock speeds in order to be able to finish the task in the time frame required. Owing to this issue, VLSI implementation of the turbo decoding algorithm is beginning to attract some attentions as a viable solution to this problem. (see [4] to [10]). In [4], ways of reducing the dynamic range of the path matrices and soft-bit LLRs are discussed. A VHDL implementation of a turbo decoder is presented in [5]. The method used in [5], however, requires large amounts of memory and has a long latency.

A.J. Viterbi in [6] introduced a method that greatly reduced the required memory as well as the latency of the turbo decoder at very little extra computational cost. This method is adopted in various ways in other papers (see [7] to [9]) and here in this paper. Reference [7] discusses various VLSI implementations of a turbo decoder, from extremely complex to more efficient pipelined architectures. Herein, we have adopted and improved the latter method further to reduce the required memory and achieve better utilization of the available resources through pipelining and parallelism.

We start with a brief background on TETRA 2 and the general architecture of a turbo decoder in Section 2. A DSP implementation of the algorithm is presented and fully characterized in Section 3. The new architecture for hardware/VLSI implementation of the algorithm is presented in Sections 4 and in 5 this architecture is implemented for an FPGA target. Conclusions are drawn in Section 6.

2. Turbo Code in TETRA Release 2

The proposed TETRA release 2 standard aims to achieve higher data throughput by using wider channel bandwidths and more bandwidth efficient modulation techniques. TETRA 2 is a multi-carrier system with the specification given in Table 1.

Modulation	4-QAM/16-QAM,64-QAM
Channel Bandwidth	25 kHz/50 kHz/100 kHz/150 kHz
Number of sub-carriers	8/16/32/48
Sub-carrier spacing	2.7 kHz
Multi-carrier symbol period	1/2400 s
Multi-carrier symbols in burst	34
Time slot length	14.1 ms
Channel estimation	2-dimensional channel estimation
Error coding	PCCC Turbo coding
Code rates	1/2 and 2/3

Table 1: TETRA 2 multi-carrier system technical specification

TETRA 2 should be able to deal with various frame length possibilities in terms of number of uncoded data bits, N . We consider the following three cases of TETRA 2 bursts in this paper:

Best case	25 kHz, 4QAM, code rate 1/2	$N = 204$
Typical case	50 kHz, 16QAM, code rate 1/2	$N = 880$
Worst case	150 kHz, 64QAM, code rate 2/3	$N = 5540$

Table 2: The three TETRA 2 burst cases considered here

The turbo code in TETRA 2 is a Parallel Concatenated Convolutional Code (PCCC), with 8-

state Recursive Systematic Convolutional (RSC) encoders. The Soft-in-Soft-out (SISO) decoders are the heart of the turbo decoder and account for most of the design complexity and decoding latency. Figure 1 shows the structure of the turbo decoder, with y^i the signal bit, $y^{p1,2}$ the parity bits, L^e extrinsic feedback, Π being a random (de)-interleaver, and with a hard threshold applied at output.

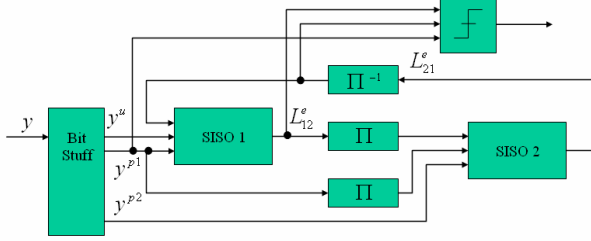


Figure 1 – Turbo Decoder Block Diagram

The aim of the turbo decoder is to iteratively estimate *a posteriori* probability (APP). Knowing the APP enables the turbo decoder to make optimal decision on the transmitted unencoded bits. This decision making process is called maximum *a posteriori* or MAP. Figure 2 shows a Matlab SIMULINK simulation of the turbo decoder for the three TETRA 2 burst conditions previously discussed.

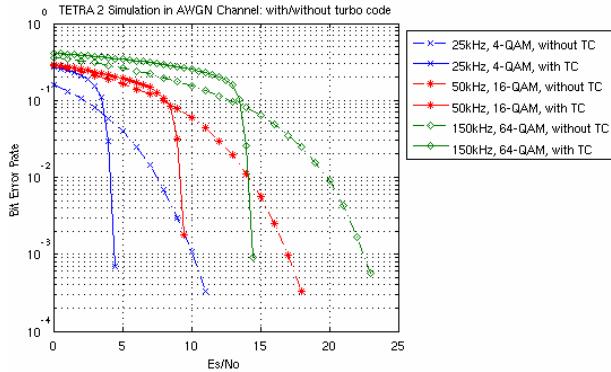


Figure 2 - MAP Turbo Decoder performance for TETRA 2

3. DSP Implementation of Turbo Decoder

It is shown in [3] that performing the calculations in log domain greatly simplifies the MAP algorithm, through a generalization of equation (1).

$$\max^*(A, B) \equiv \log e^A + e^B = \max(A, B) + \log 1 + e^{-|A-B|} \quad (1)$$

Using (1), the MAP algorithm [1] can be rewritten as in (2), (3) and (4).

$$\tilde{\alpha}_k(s) = \max_{s'}^* \tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) \quad (2)$$

Equation (2) represents the recursive forward calculation of the path log-likelihood metric at time k for state s, $\tilde{\alpha}_k(s)$ updated by the logarithmic branch metric, $\tilde{\gamma}_k(s', s)$, from state s' to state s.

$$\tilde{\beta}_{k-1}(s) = \max_s^* \tilde{\beta}_k(s) + \tilde{\gamma}_k(s', s) \quad (3)$$

Equation (3) represents the equivalent backward calculation.

$$L(u_k) = \max_{U_+}^* \tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s) \quad (4)$$

$$- \max_{U_-}^* \tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s)$$

Equation (4) represents the LLR.

Here $\Delta = \log 1 + \exp -|A - B|$, the correction factor in (1) can be realized by a simple lookup table (LUT) [3], with no more than 8 entries as in (5).

$$\text{LUT} = [0.693, 0.313, 0.123, 0.049, 0.018, 0.007, 0.002, 0.001] \quad (5)$$

The MAP algorithm, offers slightly better performance of almost 0.2 dB at a much higher complexity cost but the Log-MAP is preferred compared to the MAP due to its simplicity. As DSPs do not inherently support floating-point (real) representation of numbers, all the numbers need to be represented as fixed-point (integer) values. It can be shown (see [4]) that two-bit precision is enough for numbers representing the path and branch metrics. The Log-MAP correction LUT should also be rounded to the nearest representation of Δ using $p = 2$ precision bits. Therefore, Δ can be approximated as in (6).

$$\Delta \approx \log \left[1 + \exp \left(\frac{-|A - B|}{2^p} \right) \right] \quad (6)$$

Then Δ can be quantized to 2^p levels by first applying (7):

$$\Delta_q = 2^p \Delta + 0.5 \quad (7)$$

Hence, after rounding Δ_q to the nearest integer, the

correction look-up-table will look as (8):

$$\text{LUT} = [3, 2, 2, 2, 1, 1, 1, 1] \quad (8)$$

In DSP targeted turbo decoder, the SISO decoding involves storing the result of calculations at each step for every bit in the frame. This means that the SISO decoder requires a large memory space to store these calculation results. The total memory required for internal operations of the SISO decoder will amount to 355 KB. This is on the high side and cannot be handled by the fast internal memory of the DSP, but will need to be stored on slower external memory, which affects the overall speed of the operation.

To estimate the speed, the DSP needs to run the turbo decoder algorithm for TETRA release 2, we need to know the total number of clock cycles. This is found by summing up the number of generic-operations in the algorithm code, weighted by their complexity factor.

The turbo decoder involves N_{itr} iterations of two SISO decoding and two interleaving operations. Hence, the total number of clock cycles needed for N_s states to execute turbo decoder will be:

$$G_{opt}[TDC] = 2 * N_{itr} * (N * N_s * 80 + 2 * N) \quad (9)$$

Considering the three cases of TETRA 2 bursts discussed in Section 2, the clock speed the DSP needs to run in order to decode the burst, inside the 14.1 ms time-slot length will be as below (with $N_s = 8$, $N_{itr} = 10$):

- Best case: $G_{opt} \approx 2.6M \text{ cycles} \rightarrow$

$$\text{Speed} > 2.6M/14.2ms = \underline{183 \text{ MHz}} \quad (10)$$

- Typical case: $G_{opt} \approx 11.3M \text{ cycles} \rightarrow$
Speed $> 11.3M/14.2ms = \underline{796 \text{ MHz}}$ (11)

- Worst case: $G_{opt} \approx 71.1M \text{ cycles} \rightarrow$
Speed $> 71.1M/14.2ms = \underline{5.007 \text{ GHz}}$ (12)

DSPs capable of running at these speeds are tend and power hungry, which makes them unsuitable for wireless mobile applications such as TETRA 2. To test the turbo decoder code on the “Blackfin” DSP, the ‘EZ-KIT Lite’ evaluation platform (see [10]) from ‘Analog Devices Inc.’ was used. The code was successfully executed on the target hardware, but not in the time frame required. Multi-core design, special DSPs with internal hardware accelerators, or multi pipelined DSP architectures can improve the number of G_{opts} that can be carried out in one clock cycle. But these even further increase the cost of decoding operation on the DSP platform. For these reasons we need to look into an alternative approach, which not only can achieve the required decoding speed, but also do it in an efficient way in terms of cost and power. VLSI solution as an alternative approach to the problem is presented next.

4. DSP Implementation of Turbo Decoder

To address the issues associated with DSP implementation of the turbo decoder, a VLSI architecture is proposed. A VHDL implementation of a VLSI turbo decoder is presented in [5], which concentrates on presenting an optimized architecture for the inner-loop. However, this architecture cannot achieve very high data throughputs, because the Log-MAP algorithm, as presented in the Section 3, is an inherently sequential process (equations (2)-(4)). Its other side effect is the amount of memory required, which makes it almost unfeasible for VLSI implementation.

A. J. Viterbi in [6] addressed the issue of latency and the high memory requirement. Calculating $L(u_k)$ for k equal to zero requires knowledge of $\tilde{\alpha}_k$ and $\tilde{\beta}_k$ for the same bit. Starting from the beginning of the frame, $\tilde{\alpha}_k$ and $\tilde{\gamma}_k(s',s)$ can be calculated straightaway, since all the information needed is available, but this is not the case for $\tilde{\beta}_k$, when k is equal to zero because, in theory, its calculation involves a backward recursion from the end of the frame all the way to the beginning. However, as shown in [6], to get an accurate path metric for $\tilde{\beta}_k$ a recursion of length N_{NDB} , where $N_{NDB} \ll N$, is enough. It is shown in [6] that, after N_{NDB} recursions equal to 5 times the length of the encoder RSC memory, the path metric $\tilde{\beta}_k$ becomes as accurate as if it was starting from the final node. This N_{NDB} calculation can be

regarded as a ‘learning phase’, during which the metrics produced are not usable. So

$$N_{NDB} = 5 \times \log_2(N_s) = 15 \quad (13)$$

To reduce the required memory, [6] proposes to break the frame into smaller chunks and perform decoding on each section. The extreme condition for the approach in [6] is to divide the data frame into N/N_{NDB} sections. This method requires calculation of learning phase backward recursion ($\tilde{\beta}_k$) for each section, but this does not have to take place in sequence. This method is regarded as sliding window SISO decoding, is illustrated in Figure 3.

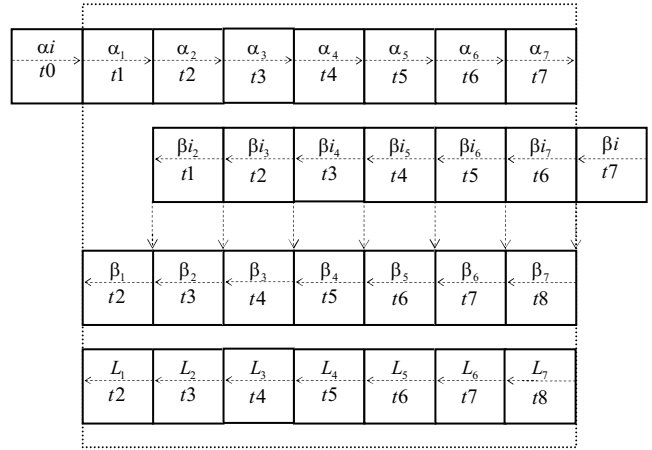


Figure 3 – Parallel backward metric calculation for N/N_{NDB} sectioned data.

Because the learning phase backward recursion calculation ($\tilde{\beta}_k$) for each stage is performed during the previous stage and also because $\tilde{\alpha}_k$ and $\tilde{\beta}_k$ can now be calculated in parallel, the latency of the SISO decoding can be reduced further by a factor of three.

5. Proposed Memory Efficient, Low Complexity Architecture Turbo Decoder

SRAM-based FPGAs make an ideal candidate for TETRA release 2. Instead of having a parameter configurable architecture, we can have several highly optimized fixed architectures stored in a flash memory outside the FPGA and depending on the frame type selected, the configuration code associated to that will be loaded into the FPGA and, from then on, the FPGA can perform the specific turbo decoding algorithm. The Xilinx Virtex-II FPGA was chosen for implementation of the turbo decoder design. An RC200 FPGA development board from Celoxica Ltd, UK with ‘xc2v1000’ Virtex II was employed. Celoxica’s associated Handel-C language, called from within the DK integrated design environment, proved a powerful tool to compile and synthesize the algorithmic procedures, originally written in C, and convert them to HDL/EDIF ([11]).

5.1. Proposed Turbo Decoder hardware architecture and state-machine for TETRA 2

In order to fully utilize the logic, allocated to perform the algorithm aspects, only one SISO

performs both stages of turbo decoding, as shown in Figure 4.

Blocks with grey background are on-chip dual-port-memories, while others are functional blocks.

The Finite State Machine (FSM) block controls the operation of the functional blocks and data transfer between these blocks and the memories.

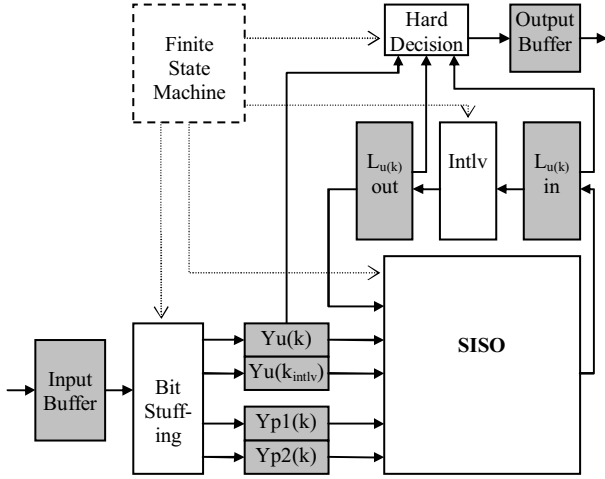


Figure 4 – Turbo Decoder architecture for FPGA implementation.

5.2. Proposed SISO architecture

By careful pipelining strategy the number of input register buffers are reduced from three as suggested in [7] to only two buffers. These buffers used to constantly provide the internal Add-Compare-Select (ACS) logic blocks with data. The size of these buffers is only N_{NDB} . Unlike the external buffers to SISO block, which are of type RAM, the internal buffers are of register type; hence, all the memory locations in a register type buffer can be addressed and used simultaneously. This enables parallelism to be used in the ACS functional blocks. Figure 5 illustrates the operation of the SISO architecture.

$S(0)$	
$\alpha(0)$	$S(1), \beta_{init}(1)$
$\beta(0), L(0), S(2), \beta_{init}(2)$	$\alpha(1)$
$\alpha(2)$	$\beta(1), L(1), S(3), \beta_{init}(3)$
...	...
$\alpha(n-2)$	$\beta(n-3), L(n-3), S(n-1), \beta_{init}(n-1)$
$\beta(n-2), L(n-2)$	$\alpha(n-1)$
	$\beta(n-1), L(n-1)$

Figure 5 – The SISO sequence and register buffer usage

S in Figure 5 denotes operations involved in storing the old extrinsic information as well as the received data and parity into the register buffers. α denotes forward recursion ACS operations, β_{init} denotes learning-phase backward recursion calculation and β is the actual backward recursion ACS

calculation. L indicates extrinsic information generation.

This method not only requires less memory in comparison to [7] but also the latency is reduced from $3 * N_{NDB}$ down to $2 * N_{NDB}$. It might be interesting to note that the latency of DSP implementation of SISO was $3 * N$.

5.3. SISO structure

Figure 6, in the next page, shows the fully pipelined and concurrent structure of the SISO decoder.

The Finite State Machine (FSM) in Figure 6 controls the operation of the ACS modules. All the ACS modules work in parallel and with the help of the FSM, pipelining is implemented to ensure constant activity of the blocks.

5.4. Complexity calculation

The total number of memory bits for internal operation of the SISO decoder is:

$$mem[total] = N_{NDB} * 102 = 1632 \quad (14)$$

The internal memory requirement does not change for different types of frames, unlike the DSP implementation. For the typical TETRA 2 case, the memory requirement is reduce from 451000 bits (DSP case) to 1632 bits (FPGA case). That is an improvement by a factor of 276. The total number of operations for SISO decoding of the whole frame is:

$$opt[total] = N * N_s * (32 + 32 + 32 + 26) \quad (15)$$

The hardware implementation of the SISO decoder requires (6+1) clocks cycles per bit. So the total number of clocks to decode the whole frame will be:

$$clocks[total] = N * 7 \quad (16)$$

This means that by adopting a fully pipelined and concurrent architecture, almost 1000 arithmetic operations are carried out at each clock cycle. After synthesis, the algorithm just fitted on the Virtex II, with almost total slice usage.

To characterize the turbo decoder design, the bit-error-rate was measured for Table 3's four conditions, which yielded results consistent with the SIMULINK simulation of Figure 2.

50 kHz, 16-QAM, Rate 1/2	SNR = 9dB	SNR = 11 dB
With Turbo Decoder	0.01	0.00
Without Turbo Decoder	0.09	0.07

Table 3 – Bit-Error-Rate at different Signal-to-Noise Ratios

The number of clocks required to execute the turbo decoder for the three TETRA 2 possible frame conditions is as below:

- Best case: $N = 204 \rightarrow$
No. clock cycles = 29k
- Typical case: $N = 880 \rightarrow$
No. clock cycles = 110k
- Worst case: $N = 5540 \rightarrow$
No. clock cycles = 655k

On average the number of clock cycles required to perform turbo decoding on FPGA platform is reduced by a factor of 100 from the DSP generic operations calculations case. Hence, running the FPGA at 50 MHz, is enough to decode the worst case scenario (*i.e.*

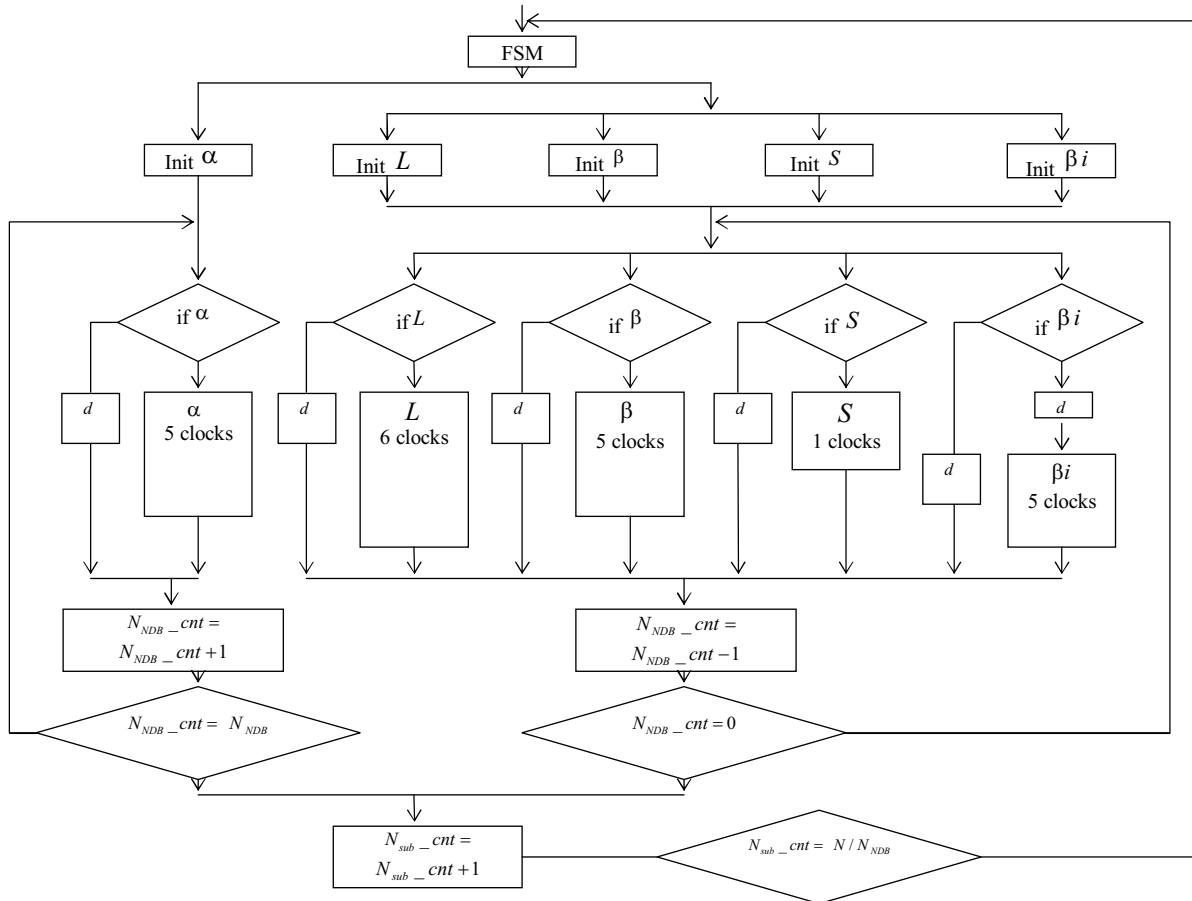


Figure 6 – Proposed SISO architecture

$N = 5540$) inside the 14.1 ms required time-slot, while in the DSP case a clock frequency of over 5 GHz was required to deal with the same type of frame.

6. Conclusion

In this paper, we have proposed a low complexity, memory efficient and optimized architecture for turbo decoder, suitable for TETRA release 2 application with its unique requirements. We showed that a DSP implementation is not a viable solution but that an FPGA implementation is. The next step is making use of the dynamic partial reconfiguration property of the FPGAs by programming an optimized version of the algorithm for the type of channel and modulation technique required. This will lead to an adaptive, on-the-fly reconfigurable hardware, which can run a highly optimized design, suitable for the wireless mobile application.

ACKNOWLEDGEMENT

The author would like to acknowledge the ESE department of the University of Essex for providing the hardware equipment and especially Dr. Martin Fleury for his support and helpful suggestions and comments, throughout the course of this work.

REFERENCES

- [1] C. Berrou, A. Glavieux, and P. Thitimajshma, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo Codes", IEEE proc. ICC 1993, pp. 1064-1070
- [2] "TETRA Summary", Mar 2005, online at <http://portal.etsi.org/tetra/Summary.asp>
- [3] W. E. Ryan, "Concatenated Convolutional Codes and Iterative Decoding", in Encyclopædia of Telecomms., ed. E. G. Proakis, Wiley, 2003
- [4] G. Montorsi, S. Benedetto, "Design of Fixed-Point Iterative Decoders for Concatenated Codes with Interleavers", IEEE Journal on Select. Areas in Communications, Vol. 19, No. 5, May 2001, pp. 891-882
- [5] Y. Tong, T. Yeap and J. Chouinard, "VHDL Implementation of a Turbo Decoder with Log-MAP-Based Iterative Decoding", IEEE Trans, on Instrument. And Measure., Vol. 53, No. 4, Aug. 2004, pp-1268-1278
- [6] A. J. Viterbi, "An Intuitive Justification and Simplified Implementation of the MAP Decoder for Convolutional Codes", IEEE Journal on Select. Areas in Communications, Vol. 16, No. 2, Feb. 1998, pp. 260-264
- [7] G. Masera, G. Piccinini, M. R. Roch and M. Zamboni, "VLSI Architecture for Turbo Codes",

- IEEE Trans. on VLSI Systems, vol. 7, No. 3, September 1999, pp. 369-379
- [8] Z. Wang, Z. Chi, and K. K. Parhi, "*Area-Efficient High Speed Decoding Schemes for Turbo/Map Decoders*", IEEE Trans. on VLSI 10(12), 2002.
 - [9] X. Zeng, Z. Hong, "*Design and implementation of a Turbo Decoder for 3G W-CDMA Systems*", IEEE Trans. on Consum. Elect. , Vol. 48, No. 2, May 2002, pp. 284-291
 - [10] Analog Devices, "*ADSP-BF533 EZ-KIT Lite Evaluation System Manual*", Revision 2.0, Jan. 2005
 - [11] K. Nezami, "*Design of a Software Configurable Modulator for a TETRA Channel*", Proc. EUWT, 2004.