# Turbo Receivers with IT++

Bogdan Cristea*
Sytron Technologies Overseas
266-268 Calea Rahovei
Bucharest, Romania
cristeab@ieee.org

## ABSTRACT

This paper presents an extension of the IT++ library useful for the study of turbo receivers. Turbo receivers are implemented using basic Soft-Input Soft-Output (SISO) modules. From a programming point of view, SISO modules are represented by methods of one C++ class. Each SISO module is defined by a specific instantiation of the Maximum A Posteriori (MAP) algorithm. Thus, various configurations of turbo receivers can be obtained. Several examples of turbo receivers together with simulation results are given.

## Categories and Subject Descriptors

[**open source simulation tools**]: digital communications

## Keywords

signal processing, turbo receivers, compiled programming language, C++

## 1. INTRODUCTION

Scientific research relies heavily on computers and simulation software in order to obtain new results and to validate the theoretical developments. In this paper we will consider one application of simulation software in order to study turbo receivers for digital communications.

Turbo receivers were initially proposed for the decoding of Parallel Concatenated Convolutional Code (PCCC) [4] and have attracted the attention of the scientific community due to their very good performance in Additive White Gaussian Noise (AWGN) channels. Their success is based on the application of a suboptimal decoding algorithm relying on the iterative exchange of soft information between two SISO modules [3]. Thus, performance close to channel capacity can be obtained with reasonable complexity [4]. The SISO modules use the well known MAP algorithm which computes the output soft information, based on the input soft information and code structure [2]. The principle of PCCC was generalized to serial (hybrid) concatenated turbo codes, equalization [6], synchronization [13] and multi-user detection [22], all relying on some modified (sometimes simplified) form of the MAP algorithm.

When simulating turbo receivers, the main difficulty is the implementation of the MAP algorithm in a form suitable for a given application (decoding, equalization, etc.). In digital communications, a generally accepted tool among scientists and engineers is MATLAB [20] offering a high-level language, primarily intended for numerical computations. Open source replacements for MATLAB are GNU Octave [7] or SciLab [17], with similar functionality and programming language. MATLAB offers a powerful set of functions suited for numerical computations together with very good graphical capabilities. However, since MATLAB uses an interpreted language, the source code is not compiled but is interpreted on the fly. Thus, programs written in MATLAB could be significantly slower than progams written in a compiled language (e.g. C++ or Fortran). This particularity makes MATLAB impractical for the implementation of MAP algorithms (and not only) due to their computational complexity, so one needs to consider compiled languages. Actually, many of the functions offered by MATLAB follow the same approach and are implemented in a compiled language, offering only an interface for MATLAB. Since in MATLAB (and its open source replacements) there is no set of functions specifically designed for the implementation of turbo receivers, it seems to be a good idea to implement several SISO modules from which turbo receivers can be constructed. We have chosen C++ as the implementation language using also the IT++ library [19] for some basic mathematical and communication specific functions.

The rest of the paper is organized as follows. First, the IT++ library is presented. The main features of this open-source library are emphasized. In section 3 the implementation of SISO modules using the IT++ library is described. Several examples of turbo receivers together with simulation results are also given. The paper ends with some conclusions and directions for future work.

## 2. IT++ LIBRARY

IT++ is an open-source library written in C++. Its primarly use is in numerical computations, signal processing and digital communications. The kernel of the library con-

*Bogdan Cristea has a PhD in telecommunications from National Polytechnic Institute of Toulouse, France, his research interests including turbo receivers and multiple access systems. Currently he is working as software engineer in Bucharest, Romania.



*Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564*
*http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564*

sists of generic vector and matrix classes and a set of accompanying routines. Such a kernel makes IT++ similar to MATLAB or GNU Octave.

According to the reference documentation [19], the IT++ library originates from the former department of Information Theory (IT) at the Chalmers University of Technology, Gothenburg, Sweden. Because the library is coded in C++, the name IT++ seemed like a good idea at the time. IT++ is now released under the terms of the GNU General Public License (GPL) and developed and maintained by the users' community. In 2005, 2006 and 2007, IT++ was also developed as a part of the European Network of Excellence in Wireless Communications (NEWCOM).

IT++ makes an extensive use of existing open-source or commercial libraries for increased functionality, speed and accuracy. In particular BLAS, LAPACK and FFTW libraries can be used. Instead of the reference BLAS and LAPACK implementations, some optimized platform-specific libraries can be used as well, i.e.:

- Automatically Tuned Linear Algebra Software (ATLAS) - includes optimised BLAS and a limited set of LAPACK routines
- Intel Math Kernel Library (MKL) - includes all required BLAS, LAPACK and FFT routines (FFTW not required)
- AMD Core Math Library (ACML) - includes BLAS, LAPACK and FFT routines (FFTW not required)

It is possible to compile and use IT++ without any of the above listed libraries, but the functionality will be reduced.

IT++ works on GNU/Linux, Sun Solaris, Microsoft Windows (with Cygwin, MinGW/MSYS or Microsoft Visual C++) and Mac OS X operating systems.

Features offered by this library include:

- basic mathematical functions: templated vector and matrix classes, sparse vector and matrix classes, elementary functions on vectors and matrices, statistics classes and functions, random number generation (Marsenne Twister generator), integration of 1 dimensional functions, etc.
- signal processing: filter functions and classes, frequency domain filtering, Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Hadamard transforms, etc.
- communications: modulators (Binary Phase Shift Keying (BPSK), Phase Shift Keying (PSK), Pulse Amplitude Modulation (PAM), Quadrature Amplitude Modulation (QAM)), multipath fading channels (both frequency flat and frequency selective), convolutional and punctured convolutional codes, turbo codes, low density parity check codes, interleavers, Orthogonal Frequency Division Multiplex (OFDM) and Code Division Multiple Access (CDMA) modulators, etc.
- protocol simulation: event-based simulation classes, Transmission Control Protocol (TCP) clients and servers, packet generators, etc.
- source coding: Gaussian mixture modeling, reading and saving several different audio and image file formats, etc.
- functions for saving variables into files and loading those variables in MATLAB or GNU Octave (useful when results need to be graphically displayed since IT++ does not have graphical capabilities)
- Application Program Interface (API) for programs written in MATLAB or GNU Octave

The use of the IT++ library has several advantages: computational speed, since it uses a compiled language; a large number of routines already written, making easier the simulation of communication systems; a MATLAB-like vector and matrix kernel classes and its open-source nature, allowing to benefit from improvements written by the users' community. Also, it allows easy integration with MATLAB code, thus making possible to reuse code already written in MATLAB by implementing in C++ only the most computationally expensive routines.

In the following sections, an extension of the IT++ library will be presented for the simulation and study of turbo receivers using generic SISO modules.

## 3. TURBO RECEIVERS

In our approach, turbo receivers are implemented using basic SISO modules [3]. Thus, greater flexibility is obtained by defining the turbo receivers structure using only these basic modules. All SISO modules are implemented as methods of one C++ class. Within the SISO class, the following modules are implemented:

- decoder for Recursive Systematic Convolutional (RSC) codes
- decoder for Non-recursive non-Systematic Convolutional (NSC) codes
- descrambler for Interleave Division Multiple Access (IDMA) or Direct-Sequence Code Division Multiple Access (DS-CDMA) systems
- equalizer for multipath channels (with and without precoding)
- Multi-User Detector (MUD) for IDMA systems (with and without precoding)
- demappers for Bit-Interleaved Coded-Modulation (BICM) systems and Space Time Bit Interleaved Coded Modulation (ST-BICM) systems

These modules are described in the next subsections, starting with a description of a generic SISO module.

### 3.1 Generic SISO Module

A generic SISO module (figure 1) has two inputs with intrinsic and *a priori* information and two outputs with extrinsic information [4]. The relashionship between its outputs and its inputs is computed based on the knowledge of the coder structure by employing some form of the MAP algorithm [2].

For completeness, the definitions of intrinsic, *a priori* and extrinsic information are given below.

The intrinsic information of coded bits is defined as:

$$L(v(n);I) = \ln \frac{p(y(n)/v(n)=1)}{p(y(n)/v(n)=0)} \quad (1)$$

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564

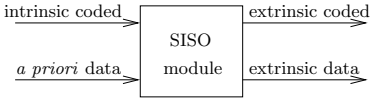


**Figure 1: Generic SISO module**

where $p(y(n)/v(n) = 1)$ is the probability density of the received modulated symbol, $y(n)$, knowing the coded bit $v(n) = 1$.

The *a priori* information of data bits is defined as:

$$L(u(k); I) = \ln \frac{P(u(k) = 1)}{P(u(k) = 0)} \quad (2)$$

where $P(u(k) = 1)$ is the *a priori* probability of the data bit, $u(k) = 1$. The relationship between the data bits, $u(k)$, and the coded bits, $v(n)$, defines the coding algorithm (e.g. convolutional coding, multipath propagation, multiple-access channel, symbol mapping, space-time coding, etc.).

The extrinsic information of coded bits is defined as:

$$L(v(n); O) = \Lambda(v(n)) - L(v(n); I) \quad (3)$$

where $\Lambda(v(n))$ is the Logarithm of Likelihood Ratio (LLR) of coded bits. The extrinsic information of data bits is defined as:

$$L(u(k); O) = \Lambda(u(k)) - L(u(k); I) \quad (4)$$

where $\Lambda(u(k))$ is the LLR of data bits. Both LLRs, $\Lambda(v(n))$ and $\Lambda(u(k))$, are computed using the MAP algorithm or some simplified version of it.

After the last iteration of the turbo reception algorithm, data bits can be recovered from the LLR of data bits:

$$\tilde{u}(k) = 1 \text{ if } \Lambda(u(k)) \geq 0 \quad (5)$$

Following the above described structure, SISO modules can be specified by using a particular instance of the MAP algorithm. These SISO modules are described in the following subsection.

## 3.2 Detailed Description of SISO Modules

The **SISO RSC module** (figure 2) decodes RSC codes of coding rate 1/2 and uses an adaptation of the MAP algorithm for RSC codes [14]. Both versions, log MAP and max log MAP are implemented. As can be seen from figure 2, its inputs and outputs are specific for this type of code [4], their detailed description being out of the scope of our paper.

An important detail related to the MAP algorithm is the specification of initial conditions. In our implementation, the trellis describing the RSC code must begin in zero state and can end in zero state (tail bits must be added to input data bits of the RSC code) or in an unknown state (no tail bits needed). Thus, one knows how to set the RSC code initial and final states in order to correclty realize the decoding operation.

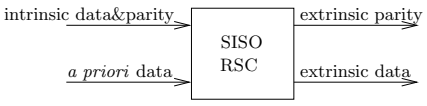


**Figure 2: SISO RSC module**

When the BPSK modulated received symbols, $y(n)$, are fed into the SISO RSC module input, the intrinsic information of data and parity bits has the expression:

$$L(v(n); I) = -\frac{2}{\sigma^2} y(n) \quad (6)$$

where $\sigma^2$ is the variance of the AWGN. The sign minus in the above expression comes from the way the BPSK mapping is done: $0 \rightarrow +1$ and $1 \rightarrow -1$. This situation appears when the symbols arriving at the decoder input, $y(n)$, after propagation through the AWGN channel, must be fed direcly to the SISO RSC module.

If otherwise specified, all SISO modules assume the use of BPSK mapping. The choice of this particular mapping ($0 \rightarrow +1$ and $1 \rightarrow -1$) was made in order to use the same mapping as the IT++ library and allows to simplify the implementation of MAP algorithms.

Applications of the SISO RSC module include turbo decoders for PCCC and Serial Concatenated Convolutional Code (SCCC). An example of a turbo decoder for PCCC will be discussed in subsection 3.3.

Another SISO module used in turbo decoders for SCCC is the **SISO NSC module** (figure 3).

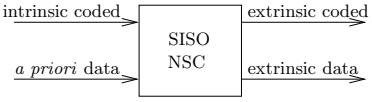


**Figure 3: SISO NSC module**

The NSC codes must have a coding rate of $1/r$, where $r$ is the number of outputs of the NSC code. Both variants of the MAP algorithm can be used (log MAP and max log MAP) and the trellis must have the same properties as in the case of RSC codes.

Optionally, a scrambler can be used at the NSC code output in order to lower the coding rate. In this case, the SISO NSC module is used for decoding both the NSC code and the scrambler. The combination between an NSC code and a scrambler is employed in IDMA systems in order to achieve error correction capabilities together with spectral spreading [15].

When only a spreading code is used (e.g. in IDMA or DS-CDMA systems [16]) a **SISO descrambling module** is also available (figure 4). The scrambler is seen as a repetition code, thus allowing the implementation of a simplified version of the MAP algorithm [12].

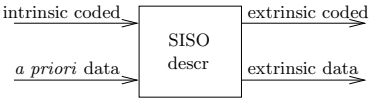


**Figure 4: SISO descrambler module**

In the case of equalization, in single-user multipath channels, a **SISO equalizer module** can be used (figure 5). The multipath channel is represented by a Finite Impulse Response (FIR) filter with real coefficients, seen as a code of coding rate 1. The channel order must be at least $L = 1$ (2 paths). The channel inputs must be BPSK modulated symbols.

*Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564*
*http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564*

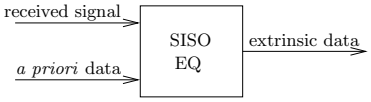


**Figure 5: SISO EQ module**

As can be seen from figure 5, the general model of the SISO module (figure 1) is not followed and the received signal is directly used as one input of the SISO module (instead of the intrinsic information). The second input is represented by the *a priori* information of the channel input symbols. There is a single output represented by the extrinsic information of the input symbols.

Optionally, a precoder can be used at the channel input in order to obtain a recursive equivalent channel. When a precoder is used, the channel order must be at least $L = 0$ (one path). Using a precoder allows to improve the performance of turbo equalization at high Signal to Noise Ratio (SNR).

The MAP algorithm (log MAP and max log MAP) is implemented according to [10]. The channel trellis must begin in zero state (that is, the channel cases must be filled with +1 BPSK-modulated symbols). The channel trellis can end in zero state or in an unknown state.

The **SISO MUD module** can be used in IDMA systems in order to decode the Multiple-Input Single-Output (MISO) equivalent channel represented by the multipath channels of all users. Its inputs are the received signal (sum of signals from all users after multipath propagation) and the *a priori* information of emitted symbols from all users (figure 6). The output of the SISO MUD is represented by the extrinsic information of all users. In our implementation, we have defined the *a priori* and extrinsic information as matrices with rows corresponding to different users. The BPSK modulation is assumed for emitted symbols and the multipath channels must have real coefficients.


**Figure 6: SISO MUD module**

The following algorithms are implemented [12], [11]:

- max log MAP
- Gaussian Chip Detector (GCD)
- simplified GCD

When the max log MAP algorithm is used, the channel trellis must begin in zero state (that is the channel cases of each users must be filled with +1 BPSK modulated symbols) and can end in zero state or in an unknown state. When the GCD and the simplified GCD are used, a zero padding technique is necessary in order to eliminate the interblock interference (a block is defined by the interleaver length) [23].

As in the case of turbo equalization, a precoder can be used at the input of each channel in order to obtain recursive equivalent channels. In this case, only the max log MAP algorithm can be used in the SISO MUD module [5].

For the more general case of Multiple-Input Multiple-Output (MIMO) channels, a **SISO demapper** was implemented (figure 7). The MIMO channel is assumed to be flat-fading,


**Figure 7: SISO demapper**

represented by a single complex attenuation at each time instant. The channel inputs must be complex symbols from QAM constellations.

The SISO demapper was designed for two types of MIMO systems: BICM and ST-BICM. For BICM the following algorithms are implemented [21]:

- log MAP
- max log MAP

and for ST-BICM [9, 24]:

- max log MAP algorithm adapted for Space-Time (ST) block codes
- max log MAP algorithm adapted for Alamouti code [1] (much lower complexity than the max log MAP algorithm for ST block codes)
- Gaussian Approximation (GA)
- simplified GA
- Minimum Mean Square Error (MMSE) Parallel Interference Canceller (PIC)
- Zero Forcing (ZF) PIC

The ST block codes are implemented using the model proposed in [8] with an an additional C++ class. Based on this model, all algorithms implemented in the SISO demapper for ST-BICM use an equivalent channel model with real coefficients including the ST block code and the MIMO channel.

The next subsection shows how the above described SISO modules can be used to implement turbo decoders/receivers for different types of communication systems.

## 3.3 Examples of Turbo Receivers

As a first example, we have chosen the PCCC (figure 8). A PCCC is constructed from two RSC codes, usually with the same generator polynomials and separated by an interleaver [4]. In our example, the generator polynomials are: $G_1 = 037_8$ and $G_2 = 021_8$. The first RSC code has the trellis terminated (tail bits are added) and the second has unterminated trellis (no tail bits). The separating interleaver is pseudo-random and has length of $2^{14}$ bits. The output is formed by using the systematic bits and the parity bits from both RSC codes (figure 8). The symbols are BPSK modulated before being send through an AWGN channel.

The IT++ library offers several classes for the implementation of the above described system. There is even a Turbo Codec class implementing both the encoder and the turbo decoder. However, our approach allows for greater flexibility with respect to the coder structure (e.g. serial or hybrid concatenated codes) and interleaver choice. Further, separating

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564

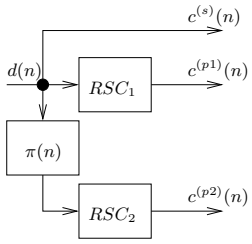


**Figure 8: Structure of PCCC**

the turbo decoders into SISO modules allows the application of EXtrinsic Information Transfer (EXIT) charts in order to study the convergence of the turbo decoder [18].

The turbo decoder for PCCC uses two SISO RSC modules (figure 9). The input of the first SISO RSC module is represented by the intrinsic information of data and parity bits. The input of the second RSC module is represented only by the intrinsic information of parity bits since the intrinsic information of data bits has been used in the first SISO RSC module. The extrinsic information of data bits is exchanged between the two SISO RSC modules during several iterations after which the decision is made in order to recover the data bits.

The performance of PCCC is evaluated using the max log MAP (figure 10) and the log MAP algorithms (figure 11). It can be seen that the use of the max log MAP algorithm has worser performance at low SNR than the log MAP algorithm. This is due to the fact that the approximation $\log(\exp(a)+\exp(b)) \approx \max(a,b)$, used in the max log MAP algorithm, does not hold at low SNR.

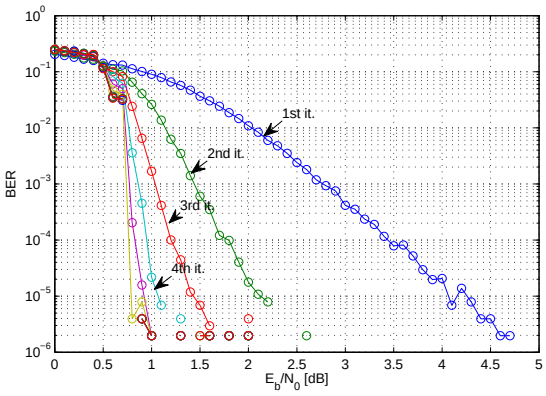


**Figure 10: Performances of the turbo decoder for PCCC using max log MAP algorithm**

Simulation time of programs written in C++ and MATLAB are shown in table 1. Both programs use methods from the SISO class. In order to call SISO class methods from MATLAB, the API offered by IT++ has been used.

Packets of $2^{14}$ pseudo-random data bits are send till 1500 bit errors are obtained at the decoder output. In order to limit the simulation time, for high SNR where the number of bit errors is small, the simulation stops if $10^6$ data bits were send. The data bits, together with parity bits, are send, after BPSK modulation, through an AWGN channel

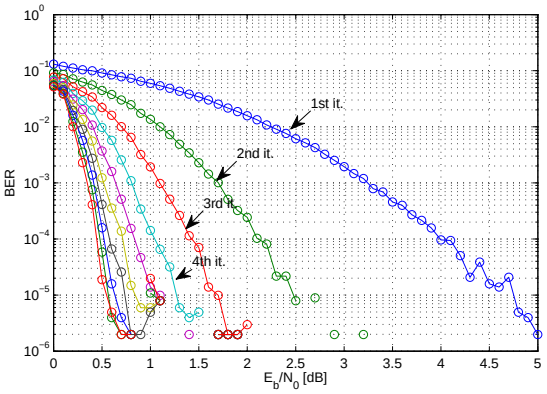


**Figure 11: Performances of the turbo decoder for PCCC using log MAP algorithm**

represented by an additive Gaussian random variable with zero mean and variance determined by the SNR. Thus, the simulation time represent the mean processing time needed to encode and decode at least $10^6/2^{14} \approx 61$ packets.

Note that the time difference between simulation times in C++ and MATLAB is almost constant for both MAP algorithms since the most complex routines are written in C++. Even so, simulation times show that there is an improvement in simulation time for programs written in C++ with respect to MATLAB programs. Also, using the IT++ library, the development time in C++ is shortened with respect to plain C/C++, since IT++ offers several classes for the simulation of digital communication systems and a MATLAB-like kernel.

**Table 1: Simulation times for PCCC in C++ and MATLAB**

| | C++ | MATLAB |
|---|---|---|
| max log MAP | 1 hr, 0 min, 22 sec | 1 hr, 35 min, 40 sec |
| log MAP | 3 hr, 3 min, 57 sec | 3 hr, 32 min, 55 sec |

As we have already mentioned above, using a turbo decoder based on SISO modules allows to study the convergence of the turbo decoder with EXIT charts. Functions needed for EXIT chart analysis are implemented as a separate EXIT class using IT++. The EXIT diagram of the turbo decoder is shown in figure 12 at a fixed SNR of $\frac{E_b}{N_0} = 0.8\ dB$. Both SISO RSC modules use the log MAP algorihm. Note that in [18] the extrinsic information of data bits is computed differently, by substracting from the LLR both *a priori* and intrinsic information, not only the *a priori* information as in our approach. Otherwise, our results are identical with those presented in [18].

A second example of the use of SISO modules is the turbo MUD for IDMA systems (figure 13). The presentation of IDMA system goes behond the scope of our paper and will not be made here. The interested reader is directed to [5] and references therein. The turbo MUD (figure 13) uses a SISO MUD module and several SISO descrambler modules, corresponding to each user. In figure 14 it is shown the performance of this multiple-access system. In this case, the performance converges toward the performance of a single-

*Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564*
*http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564*

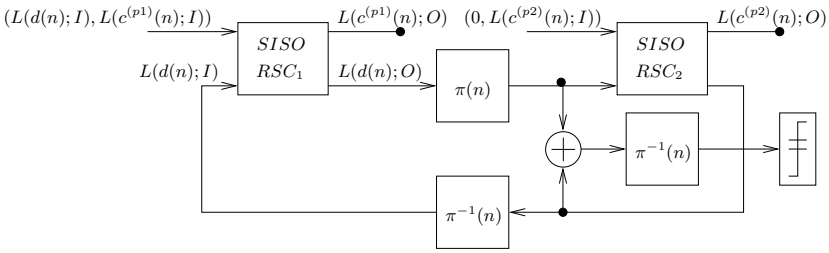


**Figure 9: PCCC turbo decoder**

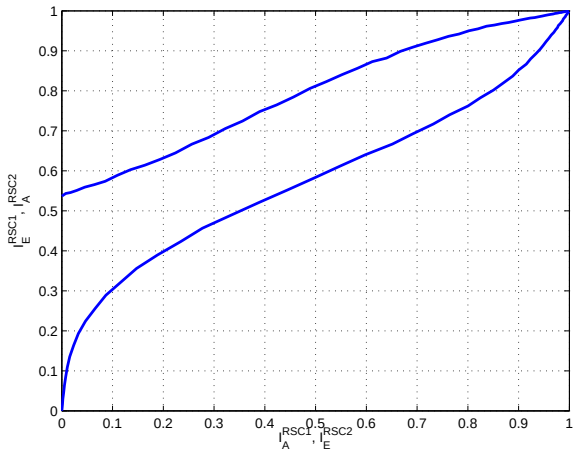


**Figure 12: EXIT diagram of the turbo decoder for PCCC**

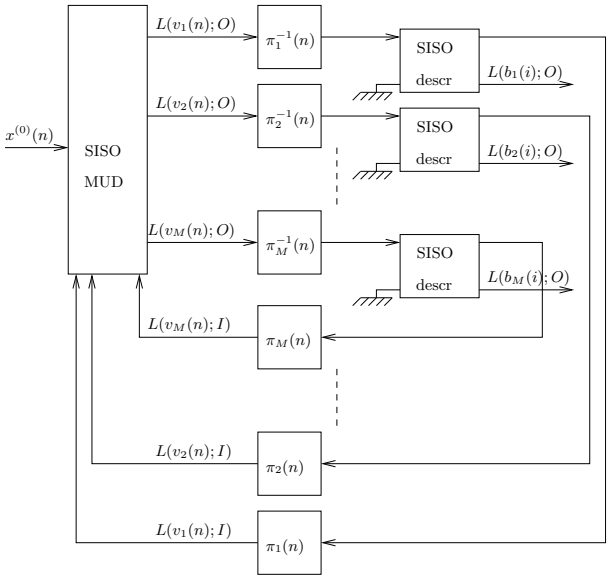


**Figure 13: Turbo MUD for IDMA systems**

user system in AWGN channel. Variations at high SNR of Bit Error Rate (BER) courves in figure 14 are due to insufficient simulation samples and can be improved using several techniques (e.g. using as stopping criterion for the simulation a greater number of bit errors obtained at the receiver output). As in the case of PCCC, the EXIT chart analysis could also be applied to study the convergence of the turbo MUD.

We have thus shown two examples of turbo receivers implemented using SISO modules. These modules are generic enough to be used for other configurations of turbo receivers. The main advantage of using this approach is the computational speed obtained from the use of a compiled language (C++). Further, the IT++ library already implements many functions and classes useful in simulations of digital communication systems.

The sources of the above presented examples, together with the SISO class, are available at
`http://cristeab.googlepages.com/SISO_index.html`.

## 4. CONCLUSIONS

In this paper we have presented an extension of the IT++ library useful for the study of turbo receivers employed in digital communications systems. The extension consists in the implementation of SISO modules for the construction of various configurations of turbo decoders/receivers. Further, this approach allows the study of turbo receivers using EXIT charts.

First, a brief overview of the IT++ library was given. Then, several SISO modules, implemented as methods of one C++ class, were described. Based on these modules, we have discussed several examples of turbo receivers. Simulation results were also presented together with simulation times for programs written in IT++ and MATLAB. Thus, the advantage of using a compiled language (C++) for complex algorithms implementation (in our case MAP algorithms) was clearly demonstrated. The IT++ library, with its MATLAB-like kernel, is in our opinion a good replacement for MATLAB and other interpreted languages used by scientists and engineers as simulations tools.

However, further work still can be done in order to extend and optimize IT++ classes, for example by taking advantage of the parallel processing capabilities of modern processors.

## 5. ACKNOWLEDGMENTS

The author would like to thank Adam Piątyszek for maintaining the IT++ library and for kindly reviewing this paper.

*Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564*
*http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564*

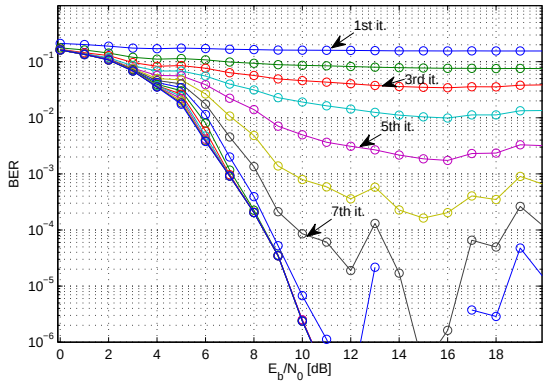


**Figure 14: Performance of the IDMA system with scrambler and using the simplified GCD for 8 users**

## 6. REFERENCES


[1] S. M. Alamouti. A simple transmit diversity technique for wireless communications. *IEEE Journal on Selected Areas in Communications*, 16(8):1451–1458, Oct. 1998.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, pages 284–287, Mar. 1974.

[3] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara. A soft-input soft-output APP module for iterative decoding of concatenated codes. *IEEE Communications Letters*, 1(1):22–24, Jan. 1997.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo-codes. In *Proceedings of ICC'93*, pages 1064–1070, May 1993. Geneva, Switzerland.

[5] B. Cristea, D. Roviras, and B. Escrig. Turbo receivers for Interleave-Division Multiple-Access systems. *IEEE Transactions on Communications*, 2008. to appear.

[6] C. Douillard, M. Jezequel, C. Berrou, A. Picart, P. Didier, and A. Glavieux. Iterative correction of intersymbol interferrence: turbo-equalization. *European Trans. Telecomm.*, pages 507–511, Sept. 1995.

[7] GNU Octave. `http://www.gnu.org/software/octave/index.html`.

[8] B. Hassibi and B. M. Hochwald. High-rate codes that are linear in space and time. *IEEE Transactions on Information Theory*, 48(7):1804–1824, July 2002.

[9] C. Hermosilla and L. Szczecinski. Turbo receivers for narrow-band MIMO systems. In *Proc. ICASSP*, 2003.

[10] R. Koetter, A. C. Singer, and M. Tuchler. Turbo equalization: an iterative equalization and decoding technique for coded data transmission. *IEEE Signal Processing Magazine*, Jan. 2004.

[11] L. Liu and L. Ping. Iterative detection of chip interleaved CDMA systems in multipath channels. *Electronics Letters*, 40(14):884–886, July 2004.

[12] R. H. Mahadevappa and J. G. Proakis. Mitigating multiple access interference and intersymbol interference in uncoded CDMA systems with chip-level interleaving. *IEEE Transactions on Wireless Communications*, (4):781–792, Oct. 2002.

[13] N. Noels, C. Herzet, A. Dejonghe, V. Lottici, H. Steendam, M. Moeneclaey, M. Luise, and L. Vandendorpe. Turbo synchronization: an EM algorithm interpretation. In *Proc. IEEE International Conference on Communications ICC '03*, volume 4, pages 2933–2937, 11–15 May 2003.

[14] S. S. Pietrobon and A. S. Barbulescu. A simplification of the modified Bahl decoding algorithm for systematic convolutional codes. In *Proc. ISITA*, pages 1073–1077, Nov. 1994. Sydney, Australia.

[15] L. Ping. Interleave-division multiple access and chip-by-chip iterative multi-user detection. *IEEE Communications Magazine*, 43(6):S19–S23, 2005.

[16] J. G. Proakis. *Digital Communications*. McGraw-Hill, New York, fourth edition, 2001.

[17] Scilab Consortium. `http://www.scilab.org/`.

[18] S. ten Brink. Convergence behavior of iteratively decoded parallel concatenated codes. *IEEE Transactions on Communications*, 49(10):1727–1737, Oct. 2001.

[19] The IT++ library. `http://itpp.sourceforge.net`.

[20] The MathWorks, Inc. `http://www.mathworks.com/`.

[21] A. Tonello. Space-time bit-interleaved coded modulation with an iterative decoding strategy. In *Vehicular Technology Conference*, volume 1, pages 473–478 vol.1, 2000.

[22] X. Wang and H. V. Poor. Iterative (turbo) soft interference cancellation and decoding for coded CDMA. *IEEE Transactions on Communications*, 47(7):11–22, July 1999.

[23] Z. Wang and G. B. Giannakis. Wireless multicarrier communications: where Fourier meets Shannon. *IEEE Signal Processing Magazine*, pages 29–48, May 2000.

[24] X. Yuan, K. Wu, and L. Ping. The jointly Gaussian approach to iterative detection in MIMO systems. In *ICC '06. IEEE International Conference on Communications*, volume 7, pages 2935–2940, 2006.



*Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5564*
*http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5564*