Implementation and performances evaluation of turbo code in SystemC

Khemaies GHALI
ISIMG, IResCoMath
BP 122, 6033 Cité El Amel 4, Gabès, TUNISIA.
khemaies.ghali@isimg.rnu.tn

Khadija BELHASSEN
ISIM de Gabès
BP 122, 6033 Cité El Amel 4, Gabès, TUNISIA.
khedija.belhassen@gmail.com

Khaled JELASSI
ENIT L.S.E.-LR11ES15
BP 37, Le Belvédère 1002 Tunis, TUNISIA
khaled.jelassi@enit.rnu.tn

***Abstract*— As demands for high performance products are becoming highly urgent due to the technological advancement stimulating the world, designers are urged to offer more compact, energy- saving and low-cost systems that would manage the complexity of the available ones. This would happen through using different level of abstraction in order to define and model the system. In this context, SystemC is a language of modeling system which allows enriching RTL level through successive refinements. In the present paper, SystemC -as a system language modeling- is used to implement the turbo encoder. Accordingly, the current work's main objective is to study turbo encoders, to understand the mechanisms of their functioning and to examine them after being modeled using SystemC.**

I. Introduction

Embedded systems have become increasingly complex due to the technological innovations stimulating the world. The design of an embedded system includes a preliminary phase of modeling attained via certain tools (languages or platforms) devised for this purpose. The choice of these tools depends simultaneously on specific constraints of the system and on the level of abstraction in which it will be modeled. The advent of new technologies requires a higher level of performance in terms of channel coding. In this context, new principles of channel coding such as Turbo codes as well as their associated algorithms of decoding are authorized. Turbo codes offer unequal levels of performances with regard to the error codes corrector, by approaching most the limit of Shannon [1].

Apart from the introduction and the conclusion, this research paper includes a first section defining SystemC, a second one devised for TLM, and another one for describing the target application, and a final section for the results.

II. Design approach through SystemC

A. SystemC Library

SystemC is a library of C++ classes allowing a feasible modelling of embedded systems including the material components at the RTL level and clock, the software components, the architecture (microprocessors, buses, memories, etc.) and the interfaces. It endows a certain application with some specification at system level in which all system abstraction levels are scheduled via the same language. By having choosing C++, SystemC bases itself on an approved and known language supported by advanced tools of development and successful compilers. One of the features of SystemC is its being a project open-source, thus, an easily extended project. [2] Indeed, the conception is slowly refined to add the necessities of hardware and of time. Using SystemC, a certain system can be modelled from the system level to the RTL level. This principle is explained by figure 1.

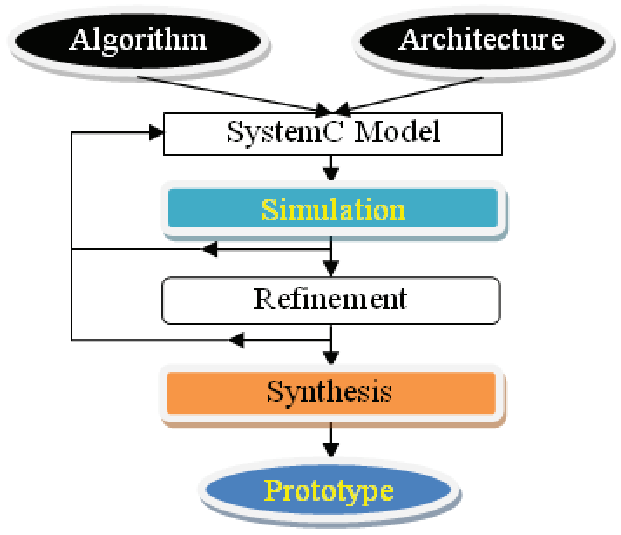

Figure 1. Design Methodology with SystemC

To represent the hardware and the software, SystemC defines a library that has to be added to the C ++ language. It contains material models which are not included in the C ++ library with a simulation kernel which serves to feign the whole system described in SystemC. The usual data of C ++ are also enriched by other types. The basic elements of a model in SystemC are either structural elements (modules, ports) or communication elements (signals, mutex, hierarchical channels). The figure 2 summarizes the architecture of this library [3].

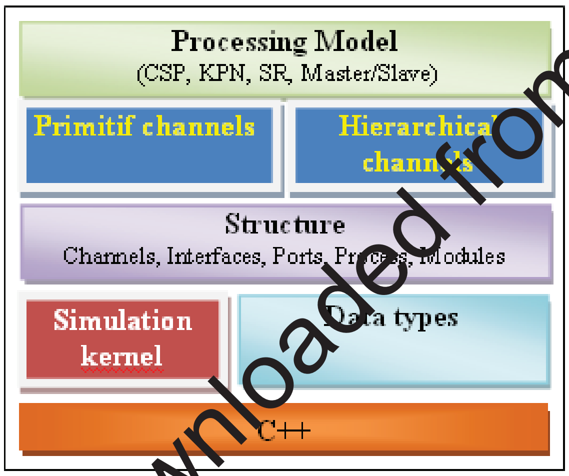

Figure 2. SystemC Organisation

The behaviour of a system described in SystemC is specified by a set of processes. A process can be seen as a transformation made on input signals to generate output signals. The notion of signal is thus essential and we can define it [4] as a sequence of event = {e1, e2,…}. An event is a couple formed by a label and a value e=(t, v) Є TxV, with t is the moment in which occurs e; v its value at this moment.

Input signal
Process
Output signal

Figure 3. SystemC process as a transformation of signals

The causality in SystemC is guaranteed by the notion of delta cycle. That is an appointment in a signal within a process is not considered at once but rather in a near future. The distance between this moment and the current moment is named delta. This virtual notion is present only within simulator and is not thus visible outside.

B. *TLM Library*

The TLM library is an extension of SystemC planned to model systems at high level of abstraction. The main objective of this library is to allow a fast and effective modeling of communications (network on chip, shared bus, etc.) used by the systems on chip [5]. This library is constituted of a communication interface set and bosses of conception who allow modeling a system at various levels of abstraction. The level of abstraction of a TLM model is generally characterized according to the following two parameters [6]:

- The granularity of the data which determines the precision and the size of the data conveyed through a transaction,

The temporal precision which defines the temporal behaviour of a model with regard to the real system.

There are two types of TLM models: TLM-PV (TLM-Programmer View) and TLM-PVT (TLM-Programmer View more Timing), proposed by STMICROELECTRONICS [7]. At the PV level a TLM model does not contain temporal information (software or hardware) of the system which it models. These models can thus be available very early in a design stream. Furthermore, this implementation details abstraction, associated with the low granularity of the exchanged data, allows obtaining very effective simulation platforms. These models are adapted to validate perfectly the feature of the system; however they do not favor the evaluation of the performances. The generic structure of a TLM model consists of initiator and target models which are interconnected via one or several interconnection modules. An initiator is a module which initiates a transaction towards a target module. A transaction is equivalent to a call of distant procedure where an initiator process calls a method of a target via one or several modules of interconnection. (An initiator and a target module can be also directly bound). These modules are bound by initiator ports and target ports which are in charge of propagating the transaction method calls. These ports can also convert the calls of methods according to the modules which they connect.

III. TURBO CODES

The ceaseless increase of the communication system complexity such as the radio applications returns their design at low level more difficult. The quality of these systems is more often estimated by determining the probability of error of the transmitted symbols [8]. Turbo codes are met doubtless at present most usually because they get closer most to the Shannon limit. The principle of the turbo-codes is described by the figure (5). A turbo-encoder is constituted by two recursive systematic convolutive encoders (C1 and C2) separated by an interleaver (π), a block of perforation (P) and a multiplexer.

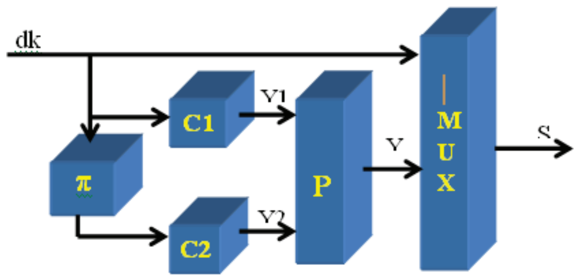

Figure 4. Principle of turbo-codes

To avoid the loss of the whole data when the errors are produced in bursts, we interleave the data before the transmission so that these errors are isolated in the time. The permutation increases the minimal Hamming distance of the concatenated code and she has to guarantee a good exchange of the extrinsic information between decoders. This interleaving has to be made in a determinist way to be able to put back the data in the good order during the decoding; it is built on the basis of a regular permutation. Various types of interleaving are specified in [9]. Both blocks C1 and C2 are convolutive encoders. Each of the output is equal to the convolution product (where from the expression "convolutive") between the binary suite presented to the entry of the encoder and the response of the same encoder defined by its generative sequences [8]. For the example of the figure 5, these sequences are given by:

$$\forall i \in \{0,1\}, g_i = [g_{i0}, g_{i1}, g_{i2}] \qquad (1)$$

with

$$\forall i \in \{0,1\}, c_k^i = \sum_{j=0}^{2} g_{ij} d_{k-j} \; et \; g_{ij} \in \{0,1\} \qquad (2)$$

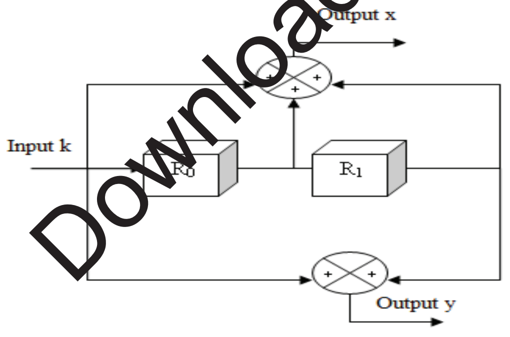

Figure 5. Example of systematic convolutive encoder

A convolutive code is said to be systematic if the initial word of information is directly got back in the word of received code.

IV. Turbo Encoder Implementation

A. *SystemC Model*

Our system consists -as illustrated in figure 4- of a generator, two convolutives encoders "C1" and "C2" which are separated by an interleaving module "π" and followed by a perforation module "P", and finally by a multiplexing module. The multiplexer makes no treatment that if both data coming from generator and from perforator are presents. Also for the perforator, it supplies the information that if both data coming from "C1" and "C2" are presents. Our design is translated to five modules which are the generator, the convolutive encoder, the interleaver, the perforator and the multiplexer. As an example we can give the specification of convolutive encoder module. This module is constituted by four input ports (data_in, data_valid, perf_req and clk), three outputs ports (perf_valid, data_out and data_req) and a process type SC_CTHREAD: "coder".

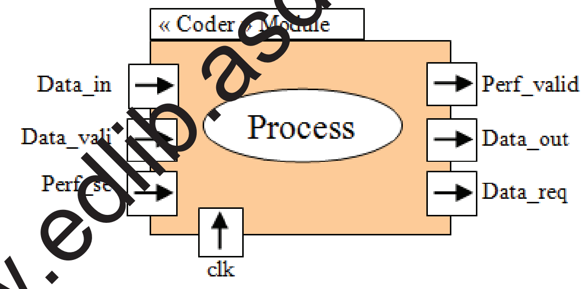

Figure 6. Coder module

The process serves to make the convolutive coding of the data coming from generator "data_in". Ports "data_req" and "data_valid" are the points of synchronization with the generator. If the encoder is at need of a data it puts "1" on "data_req". The ports "perf_req" and "perf_valid" are the points of synchronization with the perforator. If a new data is coded "perf_valid" pass in "1". The port "data_out" represents the coded information. The global system is designed by interconnecting the different modules already implemented through signals. The global architecture of the system is given by the figure 7.

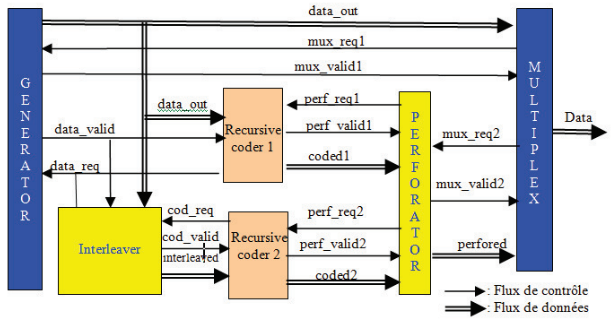

Figure 7. Turbo encoder at system level

B. *Results verification*

1) *Trace file generation*

GTKWave[10] is an electronic waveform viewer built using GTK+ toolkit, which reads VCD (Value Change

Dump file) and displays waveforms. A VCD[11] file is an ASCII file that contains header information, variable definitions, and the value changes for all variables specified in the task calls. Such a file is typically generated using simulation tools. It lets you see variables vs. time and the transition between different values. The signals interchanged in our system, represented by GTKWave, are given by the following figure:

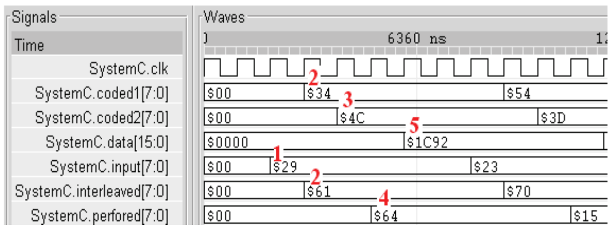

Figure 8. System data flow circulating on the system

Figure 7 shows that different modules are well executed in the planned order. Let us take the cycle of the clock where the generator generates the first vector as the reference cycle (1). In the following cycle, the first encoder and the interleaver generate their outputs (2). In the 3rd cycle, the second encoder generates its output (3), whereas in the 4th cycle, the perforator generates its output (4). Finally, in the 5th cycle, the multiplexer generates its output (5).

2) *Exchanged signals verification*

To make sure of correct behaviour of each module and of their interconnection we are going to take this vector and to make our encoder for the hand. The used interleaver is the one right left / high bottom. Thus, we are going to fill in the matrix of dimension 4x2=8 by our sequence of 8 bits in the normal order and we are going to read it from the right to the left and from the top downward. The sequence $29 is written in the matrix as follows:

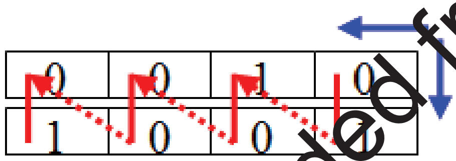

Figure 9. Principle of used interleaver

What gives 0110.0001 = $61 as interleaved data, 0011.0100=$34 as coded systematic entry and 0100.1100 = $4C as interleaved coded data. We thus find the same results represented by GTKWaves. Both sequences will be then perfored. We consider taking the first bit of strong weight of vector coming from first encoder. The perforator gives $64 as perforated data. The multiplexing of two sequences input and perfored data gives 0001.1100.1001.0010 = $1C92 = data.

3) *System inconvenients*

The major inconvenience of our system shows itself when the flow of the source is brought up with regard to the system which risk to lose data. The figure 10 shows that at the time the generator generated 13 packages, the turbo encoder produced, only, 2 coded packages. There are thus data which are lost.

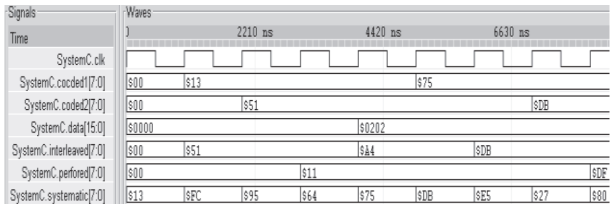

Figure 10. Data flow for 1 paquet/cycle source

To overcome this problem, it is necessary to think of a means allowing getting back the data even if flows are not equal. We propose the use of FIFO to adapt the flows of modules.

C. *Use of FIFOs*

The use of FIFOs allows synchronizing modules without need of control signals. In SystemC, such a channel is called "sc_fifo". To refine our modelling, we have to calculate the run time of different modules as well as the time put in the communication. To make it we use functions of operating system Microsoft Windows [12]. The following figure represents the increase of run time according to package size.

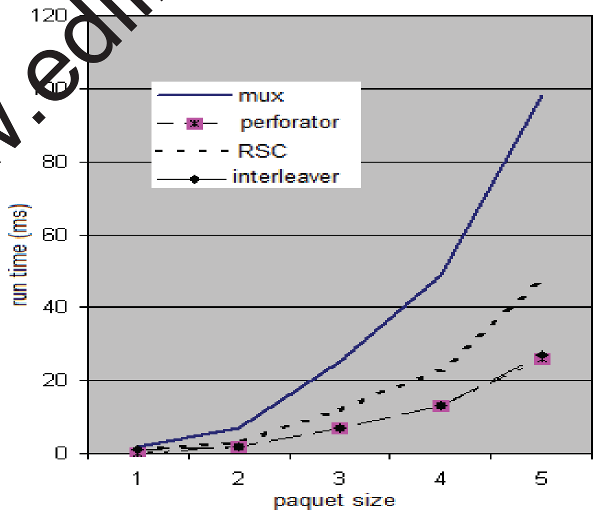

Figure 11. Execution time variation

Figure 11 shows that the module multiplexer sets more time to run. If the target processor possesses even characteristic as the simulation processor and if we are obliged (with the aim of reducing the run time) to choose the implementation of some modules in hardware, it is clear that the multiplexer is the first one. Up to now, we are able to model the features of the various modules but the communications inter-modules remain still abstracted. The TLM library allows modelling the data transfer as well as the communications protocols.

V. USE OF TLM

A. *The global system*

The TLM library is based on the architecture initiator / target. As illustrated in the figure 12, the generic structure of our TLM model consists of initiators and targets modules that are interconnected via the bus of communication. An initiator is a module in which a process initiates transaction towards a target module. Our system consists of twelve initiators, twelve targets and a bus of communication serving to interconnect them. The communication initiator / target is made through transactions. The modelling of our system in TLM level is given by the following figure:

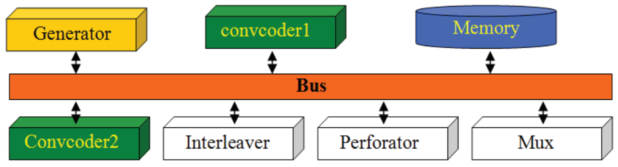

Figure 12. The system at TLM level

To preserve the data, we subdivided the range of address memory into 5 blocks. The first one is the working range of generator that is the data generated by the generator are going to be stored in this zone. The first encoder, the interleaver and the multiplexer are going to read the systematic data of this zone. Once these data are coded by the first encoder, they are written on the second zone. The interlaeaved data, which are registered in the third zone will be read by the second encoder and written on the fourth zone. The perforator is going to take in alternation one of two coded data that is it is going to read these data from the second and fourth zone and to write the result on the fifth zone.

B. *Logic of communications*

We used the library "reporting.h" in all files of implementation. Every exchanged transaction between modules is reported. It is worth noting that we have to make sure of the synchronization between the various modules, that is, an address should not be read before the corresponding datum is ready for use. For example, before reading a datum on behalf of the second encoder it is necessary to make sure that this datum is interleaved. Regrettably modules initially designed are not synchronized. So that the exchanged data are correct we use the primitive "wait" and we increase the period so much that the execution is not in the wished order. This is going to increase number of clock cycle. The generator is scheduled to generate 32 parquets. The following table resumes the flow of several modules:

TABLE I. FLOW OF DIFFERENT MODULES (TRANSACTION/CYCLE)

Module	Identifier	Nomber of transactions	Simulation periode	Flow (transaction/cycle)
Generator	101	32	2560	0.0125
Convencoder1	102	64	4160	0.0153
Mux.	103	64	5600	0.0114
Entrelacer	104	64	4840	0.0132
Convencoder2	105	64	4791	0.0133
Perforator	106	96	4800	0.02

The second column represents identifier of modules. For every module we give a unique identifier. A module is known by the bus through this number. The period of simulation is the difference between the end and the beginning of simulation of each module.

VI. CONCLUSION

Starting from an algorithmic description of turbo encoder, we managed to implement it in SystemC while taking into account the synchronization between modules and order of their execution. The lower are the levels of abstraction, the more sophisticated are the lines of the codes and the more increased the number of modules is going to be. The media of communication (abstract) is replaced by a simple bus which interconnects the different modules. The general objective is to develop a turbo encoder on the same chip by means of an innovative stream of conception.

This work allowed us to implement the system at TLM level based on the architecture target initiator through a communication media.

BIBLIOGRAPHY

[1] [1] F.Rouissi, F.Tlili, L.Ben Hadj Slama and A.Ghazel, "Improved HomePlug 1.0 FEC with SOVA Algorithm and Erasure Decoding", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.2, February 2010.

[2] [2] OSCI Language Working Group, "SystemC 2.0 User guide Update for SystemC 2.0.1".

[3] [3] N.Blanc & D.Kroening, "Race analysis for systemc using model checking", ACM Transactions on Design Automation of Electronic Systems (TODAES) , Volume 15 Issue 3, May 2010.

[4] [4] E.Riccobene & al, "SystemC/C-based model-driven design for embedded systems, ACM Transactions on Embedded Computing Systems (TECS), Volume 8 Issue 4, July 2009.

[5] [5] C.Koch-Hofer, « Modélisation, Validation et Présynthèse de Circuits Asynchrones en SystemC », thèse, Institut Polytechnique de Grenoble, 26 Mars 2009.

[6] [6] A.Mello.; I.Maia.; A.Greiner.; F.Pecheux "Parallel simulation of systemC TLM 2.0 compliant MPSoC on SMP workstations", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010.

[7] [7] L.Ferro "Vérification de propriétés logico-temporelles de spécifications SystemC TLM" these au sein de laboratoire TIMA université de GRENOBLE, Juillet 2011.

[8] [8] J.Boutros; I.Guillén; A.Fàbregas.; E.Biglieri.; G.Zémor;, " Low-Density Parity-Check Codes for Nonergodic Block-Fading Channels, Information Theory, IEEE Transactions on , September 2010.

[9] [9] R.Asghar & D.Li ," Low Complexity Multi Mode Interleaver Core for WiMAX with Support for Convolutional Interleaving", International Journal of Electronics, Communications and Computer Engineering 1:1 2009.

[10] [10] http://gtkwave.sourceforge.net/

[11] S.Penolazzi, I.Sander, A.Hemani" Predicting energy and performance overhead of Real-Time Operating Systems, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010.