The case study of block turbo decoders on a framework for portable synthesis on FPGA

Catherine Dezan †,Christophe Jégo‡, Bernard Pottier †
† Université de Bretagne Occidentale, LESTER/AS (CNRS), France
Christophe Gouyen †, Loic Lagadec †
‡ GET/ENST Bretagne, TAMCIC (CNRS) , France
dezan@univ-brest.fr, christophe.jego@enst-bretagne.fr, pottier@univ-brest.fr

Abstract— **On the case study of block turbo decoders for error correction, the paper shows that the use of unconventional framework like Madeo can produce interesting FPGA implementations.**

Error correction algorithms are known to be very important for communication data rate and reliability. As reconfigurable architectures are attractive for fast prototyping and flexibility, they are often considered as support for implementation of communication, including mobile communication.

Madeo is an open framework for designing physical FPGA architectures and their applications. On the basis of an abstract model for reconfigurable circuits, Madeo provides the necessary tools (logic synthesis and physical tool) to program them.

The paper describes the block turbo decoder principle, discusses existing solutions and presents characteristics of a partial implementation on the open framework Madeo. Three elements of a block turbo decoder have been designed using Madeo and the physical solutions compete very well with existing solutions for such problems.

I. Introduction

Communication activities such as protocol processing, content analysis, or error corrections are domains where adaptability and automatic implementation methods are critical to improve the objectives. The need for flexibility gives a strong motivation for the use of reconfigurable devices, and rises questions about development methods, portability, power and performance efficiency.

In the context of the fast development of the FPGA technology, an usual answer is to use HDL languages to develop functionalities, or domain specific tools. However, it remains a set of problems that could be solved in a more elegant way, such as the physical control of the target circuit, and how to address arithmetic components.

In this paper, we advocate the use of an open framework to model and address reconfigurable architectures. The idea is to use a common abstract model in which concrete descriptions are archived in terms of elementary active components (logic cells, arithmetic units, memories, sensors...), routing devices, and structural organization. The abstract model is a kind of windows allowing basic CAD tools to discover architecture capabilities, enabling the work of a generic place and route, floor planning, editing software. We define the model and the basic tools as the 'physical layer' of our framework (called Madeo). Several FPGAs have been described in this model, the Xilinx Virtex is serving for practical testing (section IV-B).

A second mission of the framework is to translate computations in terms of routes and active components. For fine grain FPGAs, this is logic synthesis. More complex platforms must allow arithmetic function allocations. The set of functionalities is define as the 'logic layer' which internal principles are given section IV-A. In the case of fine grain architectures, the logic synthesis functionality is restricted to the production of combinational or FSM circuits for a target FPGA technology.

Above the logic layer, higher level tools such as language compilers, or structural process compositions define the computation organizations. Feed-backs from the real architecture constraints and from the mapping tools allow to make decision with an accurate vision of the resources, as an example, to fold a linear network on a given surface under programmer control.

Madeo tools are applied on parts of a block turbo decoder, an error correction algorithm described in section II.

Error correcting codes (channel coding) are one of the solutions available to improve digital communication quality. The purpose of channel coding is to introduce some redundancy into the binary information sequence, in a controlled manner. This allows us to overcome the effects of noise and interference encountered in the transmission of the signal through the channel. Turbo codes are a family of error correcting codes built from a concatenation of elementary codes [1]. Turbo coding consists of two key design innovations: parallel concatenated encoding and iterative decoding. Iterative decoding is a sub optimal alternative that provides extremely good performance while requiring only a modest level of complexity. There are two classes of codes: linear block codes and convolutional codes. Nowadays, turbo codes are considered to be the most efficient coding schemes for channel coding (two to four dB are gained in comparison with classical error correcting codes). In the last few years, many block turbo decoder architectures have been designed at ENST Bretagne [2]. For these different studies, the design methodology has been as follow: C language is used for behavioral simulations to compare the performance and complexity of the decoders, next, VHDL or SystemC [3] RTL descriptions of the architectures are necessary for functional simulations and syntheses. This traditional design flow is usually used to design specific circuits (ASIC/FPGA).

Section V proposes to show how the framework allows

0-7695-2507-5/06/$20.00 (C) 2006 IEEE

elements of a block turbo decoder to be produced. The solution makes use of Galois Field properties, that are described in the development language as an arithmetic class, easy to test and reuse. Three elements are investigated. Two lead to a combinational circuit produced from an applicative description. The third one mix logic synthesis and structural placement. The development is concise and well supported at the language level. The circuits produced are inherently more parallel and more optimized than solutions obtained using previous tools.

II. Block turbo decoder

The iterative decoding process can be used with the block code family of product codes. Block turbo codes are an alternative solution to convolutional turbo codes. They are especially attractive for high-speed applications and offer a very good coding gain at high code rates.

A. Iterative process of product codes $(n,k,\delta)^2$

In 1954, Elias [4] proposed product codes that are series of concatenated codes. The product code inherits the properties of codes that compose it. Let us consider an example with the two identical BCH codes (n, k, δ) as illustrated in Figure 1. So, the parameters of the resulting product code are given by: n^2 (code length), k^2 (number of information symbols) and δ^2 (minimum Hamming distance). In this case, the elementary BCH (Bose Chaudhuri Hocquenghem) code is able to correct $t = \lfloor(\delta-1)/2\rfloor$ errors. In 1972, Chase [5] proposed an algorithm that approximates the optimum sequence decoding of block codes with low computing complexity and small performance degradation. In 1994, R. Pyndiah [6] presented a new iterative decoding algorithm for decoding product codes, based on the iterative SISO (Soft Input Soft Output) decoding of concatenated block codes. In fact, the iterative decoding algorithm consists of cascaded SISO elementary decoders for rows and columns. Moreover, a reconstruction of the matrix is necessary between each decoding.

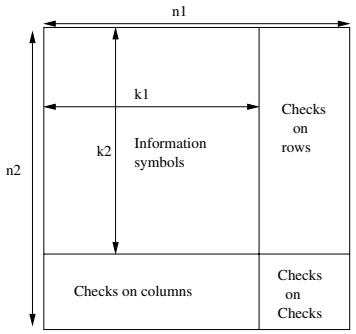

Fig. 1. Principle of product codes

B. Elementary SISO BCH decoder (n, k, δ)

BCH codes are a special type of linear cyclic block codes. Actually, a code is linear if and only if the set of code words forms a vector subspace over a finite field like a Galois field. Moreover, in a cyclic code, for every code word $c = (c_0, c_1, \ldots, c_{n-2}, c_{n-1})$, $c' = (c_{n-1}, c_0, \ldots, c_{n-3}, c_{n-2})$ is also a code word. Thus all n shifts of c are also code words. A Galois field is a mathematical structure that contains a finite number of elements upon which addition and multiplication is defined. So a BCH (n, k, δ) code forms a Galois field $GF(2^n)$ defined by a generator polynomial. This means that the size of the Galois field (i.e. the number of elements in the field) uniquely determines the field. An elementary SISO BCH decoder is shown in Figure 2, where k stands for the number of half-iterations.

- R'_k is the vector received from the previous half-iteration,
- R_k is the vector received from the channel ($R'_k = R_k$ for the 1st half-iteration),
- W_k is the vector that contains the extrinsic information, that is, the additional information given by the decoder concerning the reliability of the decoded bit,
- D_k is the result of binary decoding,
- α_k and β_k are constants that depend with the current half-iteration.

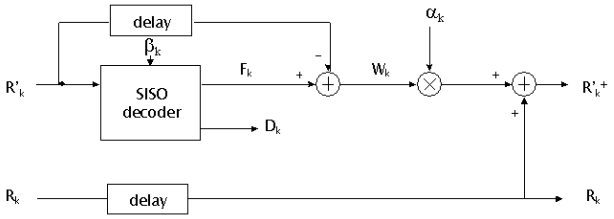

Fig. 2. Block diagram of half iteration elementary SISO decoder

The decoding algorithm of an elementary SISO BCH can be summarized as follows [2]:

1) Search for the p least reliable binary symbols of R'_k, their positions being called I_1, I_2, …,I_p,
2) Build the τ test patterns T^Q (Q∈[1..τ -1]), which are a combination of elementary test vectors T^0 with inversions for the least reliable binary symbols,
3) For each test vector T^Q, compute Z^Q : $Z^Q = T^Q \oplus$ *sign of* R',
4) For each test vector T^Q, do the BCH binary decoding,
5) For each vector C^Q found, compute the square Euclidean distance between $R'k$ and C^Q,
6) Select codeword C^d having the minimal distance with R'_k. Then $D_k = C^d$ is the result of the binary decoding,
7) Select the closest test vectors to D_k, called competitors,
8) Compute the vector F^k that contains the reliability for each binary symbol of D_k,
9) Compute the vector W_k that contains the extrinsic information

The BCH binary decoding that is used by each test vector, involves four major steps. We first calculate the syndrome values, which represent the sequence of errors in the frequency domain. The number of syndrome values is equal to the number of parity symbols 2t. Then we solve the key equation by determining the error locator polynomial $\gamma(x)$ from the

syndrome values. This calculation is realized by the Peterson algorithm [7]. The roots of the error locator polynomial $\gamma(x)$ are computed using the Chien search algorithm [8] in order to provide the errors location in the test vector. The received word is finally corrected.

The least reliable binary symbols of R'_k are obtained by a comparison of the reliability of all binary symbols. The number of least reliable binary symbols depends on the code length and the error correction power of the BCH code. For example, if the code length and the error correction power are 128 and 1 respectively, we keep the five least reliable binary symbols in the elementary BCH decoder. These least reliable binary symbols are then used for the construction of the test patterns.

The elementary SISO decoder that we have chosen is dedicated to BCH code (128,120,4) correcting one error in a 128-symbol codeword.

III. Related work

Nowadays, turbo codes are becoming a popular technology for communication systems as well as for memory and data storage systems. In fact, it is considered to be the most efficient coding scheme because it offers the best trade off between complexity and performance for error correction. Several digital communication system standards use turbo codes, like IEEE 802.16, DVB-RCS/RCT, UMST, CDMA2000 or CCSDS. Moreover, future mobile applications (wireless communication systems, high quality video players, global positioning systems etc.) will consider the use of turbo codes. Thus, these new applications present new challenges for designers. In particular, performance (e.g. throughput, latency, flexibility) and Quality of Service (QoS) in mobile appliances must be in accordance with power consumption. For this reason, the trade off between performance, QoS and energy consumption is the most important system design issue.

Traditional technologies do not respond to these new constraints. Current Microprocessors and Digital Signal Processors (DSPs) provide low power implementations but the computational efficiency is insufficient for new applications. Hardware implementations based on Application Specific Integrated Circuits (ASICs) are efficient in terms of throughput and latency compared to processor implementations. But, they are not suitable for applications that need flexible and adaptive architectures. For these reasons, future mobile applications need new hardware technologies.

Currently, reconfigurable technologies are being investigated to respond to the trade off between performance, QoS and energy consumption. This approach integrates flexibility with the computational efficiency of traditional hardware architectures. Moreover, dynamic reconfiguration enables some parts of the architecture to be configured during the application execution in order to enhance performance and/or to reduce power consumption.

A. Reconfigurable turbo decoder achitectures

Some recent research activities have focused on the reconfigurable architectures of decoders for error correcting codes. These works are characterized by reconfigurable technologies (FPGA, processor-FPGA or processor-ASIC), configuration type (static or dynamic) and flexibility criteria (power consumption, performance and/or processing complexity). All the experimentations concern convolutional turbo codes.

In [9], the software programming model is extended to the design flow for a reconfigurable processor called XiRisc. This reconfigurable processor is a processor architecture composed of a DSP Risc core coupled to a run-time reconfigurable hardware device (PiCoGA called Pipelined, Configurable Gate-Array) mapping application specific computational kernels. The implementation of a Universal Mobile Communication Systems (UMTS) turbo decoder on a XiRisc reconfigurable processor is described. This approach enables a software designer to assess the costs and gains due to executing selected pieces of C code as single instructions on an FPGA-based reconfigurable function unit.

A flexible architecture for Viterbi and turbo decoding was designed and implemented on an FPGA [10]. This architecture can provide throughputs in the range of 60 Mbps for constraint length 3-9 Viterbi decoding and 3.54 Mbps for SOVA based Turbo decoding (4 iterations). Configuration of the architecture characteristics requires a single clock cycle and does not require FPGA reprogramming. The architecture flexibility was obtained from the design of configurable data routers. Configurable data routers consist of banks of multiplexers that enable the routing of the path metrics to be controlled. Another implementation [11] mapped a reconfigurable turbo decoder. In this case, reconfiguration enables the appropriate algorithm to be selected in any circumstance. In fact, two main algorithms are employed in convolutional turbo codes: the SOVA algorithm and the Log-MAP algorithm. Thus, it is beneficial to be able to reconfigure the decoder for different data rates and delay specifications of different service channels.

In [12], a dynamically reconfigurable FPGA-based turbo decoder which has been optimized for power consumption is presented. The turbo decoders were designed and mapped to an Altera Stratix FPGA on a Nios development board. The Nios embedded processor is a configurable general-purpose Risc processor that can be integrated into Altera FPGAs. In this turbo decoder architecture, the Nios processor measures the Signal to Noise Ratio (SNR). When the SNR changes, the processor picks up a correct bit-stream stored in the board SRAM, reconfigures the FPGA and generates a new decoder appropriate for the current channel noise statistics. The key power-saving technique proposed is the use of a decoder run-time dynamic reconfiguration in response to variations in channel conditions. Based on the assumption that SNR can be sampled successively every 250,000 bits, the FPGA was periodically reconfigured during the transmission process.

B. Framework dedicated to reconfigurable architectures

Reconfigurable systems suffer from a great lack of generic tools at all design levels. Traditionally, digital hardware designers have created circuits by manipulating Boolean logic, by interconnecting logic gates or transistors, or perhaps by

writing a program in a hardware description language such as VHDL or Verilog. These methods of expressing hardware concepts are applied to reconfigurable systems especially for FPGA implementation. However, reconfigurable mapping software differs from standard hardware compilers and CAD algorithms not just because of the constraints of FPGA-based implementation, but also due to the types of users these systems hope to support. Moreover, for complete reconfigurable systems, other optimizations must be performed and these optimizations have to handle the constraints of reconfigurable contexts. Finally, dynamic reconfigurable systems face even greater challenges. Methods for detecting erroneous configurations, automatically scheduling reconfigurations and placing logic to minimize the amount of reconfiguration necessary, all still need to be developed.

One of the most unique considerations of reconfigurable systems, as opposed to standard hardware implementations, is the need for high performance mapping tools. Systems that can be programmed to implement an application in a matter of microseconds, and lengthy mapping times limit the usefulness of these systems. There are many different methods and approachs to the technology mapping of circuits for FPGA implementation. But, we need flexible CAD tools that can target a wide variety of FPGA architectures efficiently and hence rapid comparisons of the architectures. A VPR (Versatile Place and Route) CAD tool was designed [13]. This tool is flexible enough to allow comparisons of many different FPGA architectures. It can perform placement and either global routing or combined global and detailed routing. More recently, just-in-time (JIT) compilation for FPGA was developed [14], enabling the development of a standard binary for FPGAs . This compiler facilitates the portability of binaries across FPGA architectures. It consists of lean versions of technology mapping, placement and routing algorithms, which require an order of magnitude less execution time and memory requirements.

Some research teams are working on design space exploration methodologies for reconfigurable architectures that take place at the algorithmic level. In [15], a design space exploration method for reconfigurable architectures dedicated to data intensive applications is proposed. The designer can quickly obtain relative performance estimations (speed, power consumption and area) of several reconfigurable architectures from an algorithmic description of the application (a subset of the C language). Other exploration methods dedicated to reconfigurable architectures have been proposed. For example, an exploration method that is focused only on energy estimation has been developed [16].

In fact, research activities concern only one of the steps of the reconfigurable design flow. We thus propose a new framework, Madeo [18], [22], that is composed of generic CAD tools dedicated to reconfigurable architectures. This framework enables the designer to define and synthesize applications from a behavioral description and to implement the design on generic reconfigurable architectures with regards to several reconfigurability considerations at once. The next section details the features of the Madeo framework.

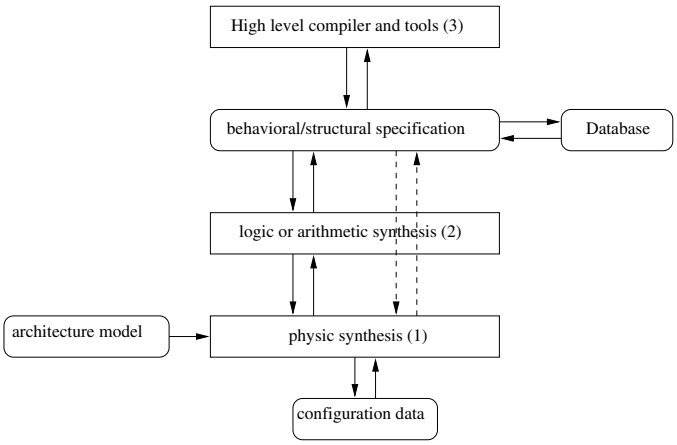

Fig. 3. Synthesis flow in an open framework for reconfigurable architectures

IV. Synthesis flow

The tool organization is shown figure 3. The flow is described as a synthesis scenario.

For the undertaken design, the top level is a structural composition of functions for a block turbo decoder. As we are working in an object-oriented system[1], the description is a complex hierarchy of objects carrying different parameters driving or resulting from synthesis, and a stack of intermediate formats ranging from the behavioral specification to a physical implementation. The designer knowledge is in the code located at this level, fixed in new general purpose support classes, in the developed algorithm and possibly a structural composition of a circuit. The developments are highly reusable, including for software synthesis[24].

Smalltalk-80 is used to define the circuit behaviors, and to control the physical composition. This language was selected in the initial steps of the project for its late-binding characteristics that ease greatly the integration and use of new numeric classes, such as the Galois fields.

With late-binding, the decision to execute one function or another one, is normally taken at runtime, based on the class of the object receiving a message. There is no need for type specification, and so an algorithm can work on new classes of data without recompilation, provided that the computation still make sense. For example, it is possible to test a linear algorithm on integer type and then switch to a Galois field, just by changing the operated data.

For hardware design, this is valuable since it will become possible to tune a solution by fixing the algorithm and just varying data classes, or details in a class (a floating point number, as example). This can be more natural than keeping the data constant and varying the algorithm in many cases (tuning on numeric errors, as example). It will be shown that the properties of late-binding are preserved by exchanging

[1] Smalltalk-80, variant Visualworks

interpretation for static compilation on a determined set of data.

As shown in figure 3, components can be stored in a database for different architectures and different degrees of implementation.

A. Logic layer

At layer 2, we are considering components that are combinational or sequential functions described in *methods*. Methods are functions that allow an object to answer a message.

The first compilation stage is to build a program tree using the standard compiler. A directed acyclic graph (DAG) is derived from this tree and usual compiler optimizations are applied. In the case of fine grain, the synthesis proceeds by building a representation of the computations in terms of pre-computed tables organized in a hierarchy of graphs.

An important characteristic of the synthesis is a *compilation context* (CC) that defines the data that will be processed. Data are all the objects of the input parameters. They are described as intervals, sets of constant objects, classes, or composition of these items. CCs reject the type specification outside of the algorithm, letting an automatic type inference mechanism to retrieve other CCs everywhere in the computation description. The initial CC can be computed or extracted from another synthesis.

The DAG is visited top-down, breadth-first, and the nodes are progressively fitted with their own CC based on results upward in the graph. During the visit, the simple nodes are translated into an equivalent table by calling the real method in the high level environment, while the more complex nodes are processed by a new compiler call. The control of the visit is given to the programmer who make choices such as maximum depth, maximum table capacity, speculative execution of conditional branches, loop unrolling.

CCs are Cartesian products of object collections which size must be kept under control, which is a restriction similar to the problem of general purpose programming on an 8-bit microprocessor. Under the constraint that CC collections are small enough to avoid huge table apparitions, processing will produce a hierarchical oriented graph of tables that is a symbolic form equivalent to the future network of FPGA LUTs.

Optimizations are achieved on the table network with the aim of reducing the number of nodes and the number of values in the CCs. These optimizations are more efficient than the ones based on operation semantics because they are based on the real effect of the computation. As example of optimizations achieved on the table graph, all unary and/or bijective nodes are removed, boolean condition status are enforced in the conditional blocks, un-taken tuples appearing in CCs are removed.

These mechanisms allow to be retrieved the semantic behavior on specific classes without developing a specific compiler. They provide opportunities of mapping, such as the production of C programs simulating on a network of tables, or computing on distributed memories. The translation system is robust,

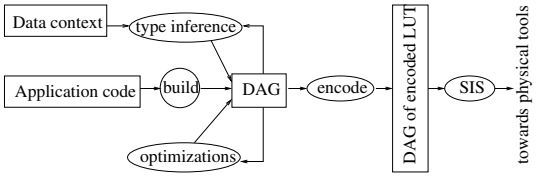

Fig. 4. Main steps of the logic layer

because the flow of data is always decreasing downward the graph: the number of different values produced by a function is smaller or equal to the number of different tuples in the CC.

The next stage of layer 2 is to produce a binary representation of the tables. This representation is given by encoding support based on the object classes, or index of the objects in an enumeration. The encoding can have an important impact either on the logic generated and the data path width between nodes.

After logic generation, two techniques can be used to obtain efficient circuits for an FPGA technology:

- synthesis, where the binary representation is a simple PLA on which logic minimization and partitioning are achieved externally[2].
- re-synthesis, when a first general solution is optimized by the production of a *don't care* set passed to the logic minimization programs[26].

In the second case, reducing the CCs always improve the solutions and logic processing is generally very fast. In the case of common functions an existing general solution is known or easy to find. At the opposite, synthesis can be very slow in an unpredictable way.

Sequential circuits are usually synthesized as combinational functions with registers allocated in the loop between current state and next state.

Finally, layer 2 has translated a symbolic function with a precise input specification into a hierarchy of optimized networks adapted to a particular LUT FPGA. The synthesis layer groups optimization at the expression level given a restricted set of operands, and at the logic level using synthesis or re-synthesis techniques.

The main steps of the logic layer are illustrated in figure 4 and one simple example is described in figures 5 and 6.

B. Physical layer

The abstract model for reconfigurable architectures covers all the resources and organization necessary to achieve basic mapping operations:

- placing and routing logic networks, by resource allocation for logic computation and for signal interconnection.
- floor-planning, by computing a minimal arrangement of placed subgraphs from the network.
- interactive editing, with usual functions such as signal routing and more original capabilities such as operating

[2]We use SIS[21] and variants

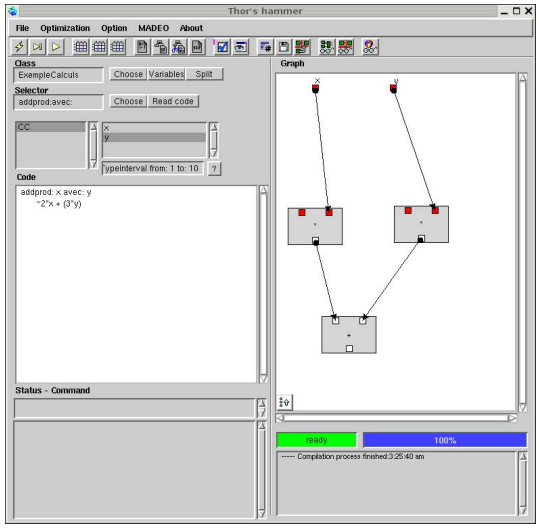

Fig. 5. DAG of the expression 2*x + (3*y) associated to the Smalltalk method named *addprod:avec:*, the compilation context for x and y is an interval of integer values from 1 to 10

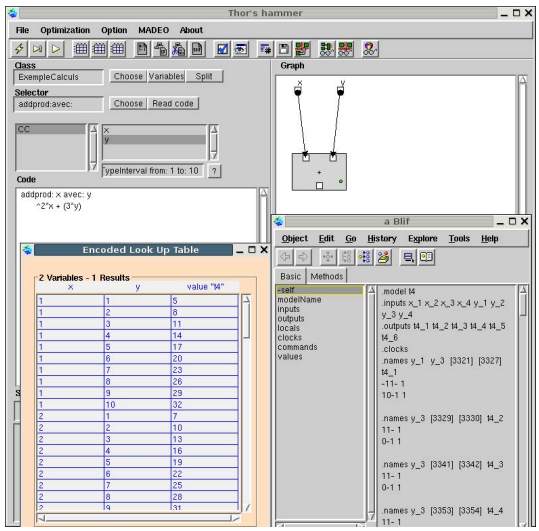

Fig. 6. DAG after high level optimizations (after unary node removal), LUT production of the remaining node (still maned + but it is no more an addition) and Blif format corresponding to the result of SIS logic synthesis tool

geometric transformations on a module, and programming complex circuit structures in a general purpose language.

As shown figure in 3, there are two inputs to this part of the framework. In a similar way as the logic layer has isolated data and functions, the physical layer isolates support architecture and application functions. Searching for performances can be achieved by a programmer acting on a fixed architecture, or by a computer architect tuning FPGA details for a fixed benchmark.

The low level layer of Madeo proposes a tool to describe interactively a new architecture in the abstract model. Another alternative is to specify the architecture textually, in a grammar

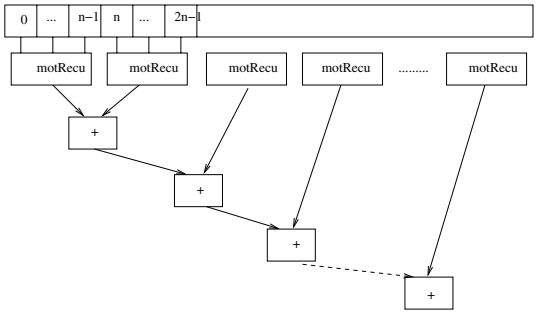

Fig. 7. Cascade organization of syndrome computation with partial sum decomposition

representing the model. As an effect of this open structure, it is also obviously possible to compute evolutions on circuit architectures.

V. Designing parts of block turbo decoder application using Madeo framework

In this section, we present some results of the syndrome computation using the Madeo approach. We also show how Madeo can be used to investigate some new decoder implementations by taking the benefits of a generic code specification and how we can deal with Galois field arithmetic. The specification style may have an impact on the final results: we show in the search of the least reliable symbol and in the parity computation that the algorithmic decomposition and the grain of the elementary cell synthesized by SIS may influence final results. Finally, we investigate different LUTs based FPGAs for future work and their implementation from application to physical synthesis using Madeo physical tool. Eventually, we compare Madeo results to a more classical approach for different parts of the block turbo.

A. Syndrome specification

The syndrome is evaluated using equation 1. This equation represents the computation of the first syndrome where α^j is an element of $GF(128)$ and r'_j elements are test vector bits of 128 bits. If the value of the syndrome is not null then it reveals transmission error.

$$S_1 = \sum_{j=0}^{127} r'_j * \alpha^j = \sum_{j=0}^{n-1} r'_j * \alpha^j + \sum_{j=n}^{2n-1} r'_j * \alpha^j + \cdots + \sum_{j=128-n}^{127} r'_j * \alpha^j \quad (1)$$

With Madeo, partial sum of the equation 1 are defined in the operator '*motRecu*' where the α^j elements are considered as constants from $GF(128)$. The r_i bit of the test vector allows its associated Galois field element to be considered in the sum only if the r_i value is equal to 1. Partial sum can be organized as shown in figure 7 for the final computation.

High level optimizations can be done on the syndrome expression such as constant folding and 'unary node removal', even if the elements belong to Galois Field. These optimizations allow reducing drastically the initial specification density, preserving only the + operators as shown in figure 8.

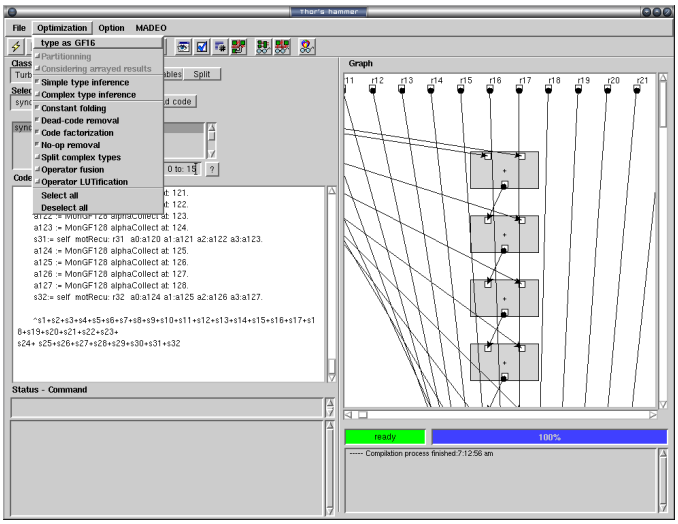

Fig. 8. Graph reduction after high level optimizations (each r_i represents 4 binary elements of the test vector)

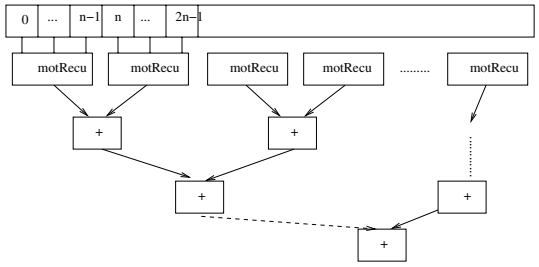

Fig. 9. Tree organization of the syndrome computation

The synthesis of the syndrome depends on the granularity of the partial sum (number of binary elements of the test vector). It contributes to the complexity of the syndrome computation as shown in table I for a 4-input LUT based FPGA. The minimum case is obtained for a grain of partial sum equal to the grain of target FPGA which the more favorable case for logic synthesis with SIS.

Basic cell grain for partial sum	2 bits	4 bits	8 bits	16 bits
nb LUTs	424	217	502	725

TABLE I

SYNDROME COMPLEXITY ASSOCIATED WITH THE GRANULARITY OF PARTIAL SUM FOR A 4-INPUT LUT TARGET FPGA

Another algorithmic organization can be defined as proposed in figure 9. This new type of organization affects the complexity of the syndrome as shown in table II. This can be explained by the fact that the filter element (represented by the operator *motRecu*) is not integrated in the same manner into the operator $+$ for the global computation.

A circuit obtained by synthesis from equation 1 corresponds to the first syndrome computation. In fact, this can be used for the computation of all other syndromes because new results can be obtained just by modifying r_j values. As the circuit produced initially is implemented for any possible values of r_j bits, it can be used to compute the other syndromes and more if required.

B. Generic code for algebraic operators

The code used for the syndrome computation is based on the operator $+$ achieving addition in Galois Field. This operator is specific of $GF(128)$, that is determined by the application context. In fact, the syndrome specification can be used for other contexts like for $GF(32)$ (in the case of $BCH(32,26,4)$ decoder), just by adapting the constants. In the next section, we show how to define specific operator in Galois field like $GF(2^n)$.

Arithmetic over Galois Field

For the finite fields like $GF(2^n)$, the elements may correspond to integer values or binary vectors associated with the binary coefficients of the corresponding polynomial form. The arithmetic operations are defined as follows:

- $+$: is an exclusive-or of associated binary values
- $*$: corresponds to an Euclidian division using the generator polynomial. This is classically achieved using logarithm tables in order to replace the main operation by an addition.

In the case of $GF(128)$, we defined the logarithm tables associated to the generator polynomial (for instance x^7+x^3+1 for the turbo decoder $BCH(128,120,4)$) using a generic approach that can be adapted to generate any $GF(2^n)$ arithmetic. Other implementations of this type of arithmetic are proposed in [20].

Impact of the encoding approachs

For the implementation of the $GF(128)$ adder, we can choose different binary representations for the Galois field values. The table II shows that the best choice is the one that corresponds to binary vectors associated with the binary coefficients of the corresponding polynomial form, other binary representation may produce poor results.

Algorithmic organization	binary polynomial form	indexation (ordinal encodage)
Cascade org.	217	217
Tree org.	393	1876

TABLE II

SYNDROME COMPLEXITY DEPENDS ON ALGORITHMIC ORGANIZATION AND BINARY REPRESENTATION OF $GF(128)$ VALUES (CASE OF 4 BIT PARTIAL SUM)

C. Architectural investigation

In this section, we show how to use Madeo framework to investigate the complexity of the final circuit. This investigation is done on two other elements of block the turbo decoder (the

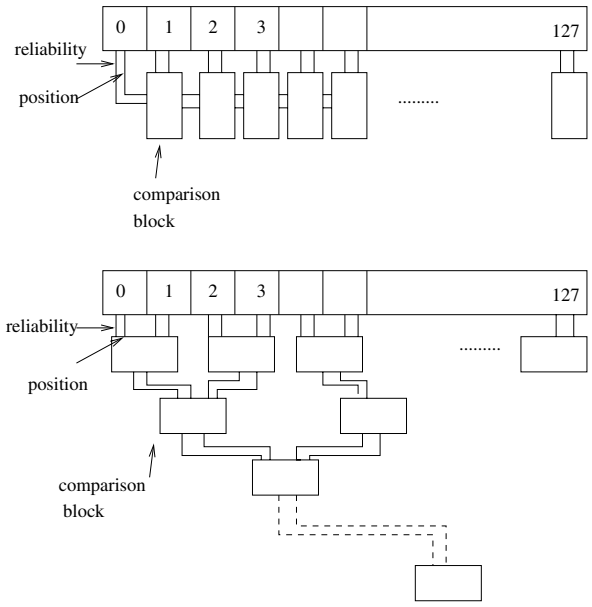

Fig. 10. Algorithmic organization of the search of the least reliable symbol (first cascade decomposition and second tree decomposition)

search of the least reliable symbol and parity computation). The criteria taken into account are the algorithmic decomposition (a tree or a cascade decomposition), the grain of the target FPGA and finally the hierarchical level considered for applying logic synthesis with SIS.

Case of the least reliable symbol

The search of the least reliable symbol consists in finding the minimum reliability of the n symbols and its corresponding position in the received vector. The computation is achieved using a basic cell that is composed of two elements: one gives the minimum of two reliabilities (each defined with 4 bits) and the second one gives the index position of the minimum in the vector (index contains between 0 and 127). These two blocks are separately synthesized by SIS. No variation of the basic cell is done here but we observe the effect of algorithmic decomposition of the problem (a cascade decomposition or a tree decomposition as shown in figure 10) onto the complexity of the final implementation. In both decompositions, the variation of the values (index value for reliability position) are not really the same. So the complexity of the basic cell varies.

The global complexity of the circuit can be evaluated as follows:

- for a tree organization, $min * \lceil n/2 \rceil + max * (\lfloor n/2 \rfloor - 1)$ LUTs
- for a cascade organization, $min + max * (n-2)$ LUTs

where *min* correspons to the minimal complexity of the basic cell, *max* to the maximum complexity of the basic cell and n of the problem size. The min-max values of the complexity of the basic cell are presented in the table III. They are associated to the grain of the target FPGA and depend on the dynamic variation of the index element of the basic cell through the computation flow. When the index variation increases, the complexity of the basic cell increases as well but not in a linear way.

grain of FPGA	LUT-2	LUT-3	LUT-4	LUT-5	LUT-6	LUT-7	LUT-8
Min-max compl. of basic cell	72 93	42 49	31 38	21 28	8 15	7 14	7 12

TABLE III

MIN-MAX COMPLEXITY OF BASIC CELL FOR THE SEARCH OF THE LEAST RELIABLE SYMBOL IN TERMS OF NUMBER OF LUT-NS

The complexity can be reduced, if we investigate the dynamic variation of the values. For instance in the case of LUT-4 and for a tree organization of the search of the least reliable from 128 symbols, we obtain : $31*64+33*32+34*16+35*8+36*4+37 = 4045$ LUTs (31 for the first stage of the tree, 33 for the second stage,...). For the cascade organization, the reduction is not so regular and more difficult to predict.

Case of parity computation

It is used to extend BCH code by adding an extra bit to the code associated to the parity of the bit vector (or-exclusive of the bits of the vector). The parity block can be designed considering the grain of the target FPGA (LUT-N, N inputs, one output) and the grain of the basic computation block (cellP for P input bits). The table IV gives the complexity of a basic computation cell in terms of number of LUT-N (grain of target FPGA). Figure 11 details the real occupation of the implementation of parity computation for 32 bits. We observe that a basic cell with 4 inputs on a FPGA LUT-4 is quite a good compromise for parity complexity. In this case, the algorithmic decomposition has no impact because there is no dynamic variation of the values (only binary local outputs).

LUT-N	cell2	cell3	cell4	cell5	cell6	cell7	cell8
LUT-3	1	1	2	2	3	3	4
LUT-4	1	1	1	2	2	2	3
LUT-5	1	1	1	1	2	2	2
LUT-6	1	1	1	1	1	2	2
LUT-7	1	1	1	1	1	1	2
LUT-8	1	1	1	1	1	1	1

TABLE IV

COMPLEXITY IN TERM OF NUMBER OF LUT-NS FOR THE BASIC CELL FOR PARITY ASSOCIATED TO THE GRANULARITY OF THE LUT-BASED FPGA

D. Layout definition and comparison with classical approach

The layout definition of the different elements of the turbo decoder can be effectively done using Madeo physical tool (figures 12 and 13) . It can take advantage of the regular structure to deal with complex circuit like in the case of the search of the least reliable symbol (figure 13).

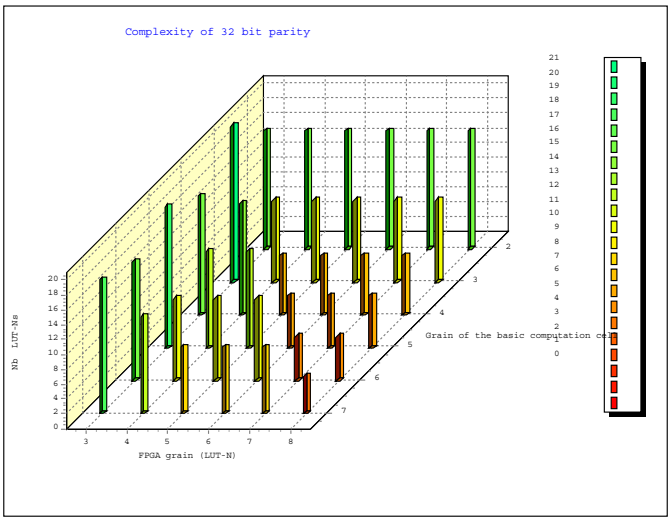

Fig. 11. Complexity of the parity computation for 32 bits depending on the grain of basic computation cell and on the grain of target FPGA

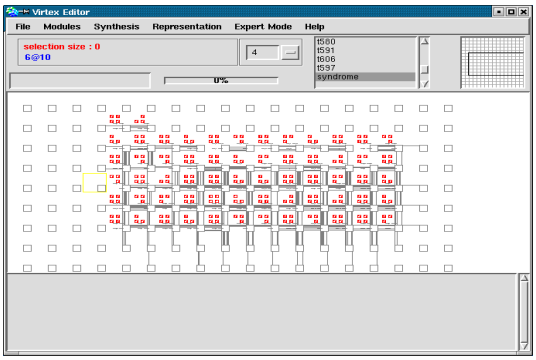

Fig. 12. Syndrome circuit placed and routed on Virtex structure using Madeo physical tool

The result obtained in this project were compared to other implementation obtained using classical approach, like SystemC [3].

The systemC description implements the block turbo decoder $(128,120,4)^2$.The design was integrated into an Altera's Nios® Development Kit, *Stratix*^{TM} Professional Edition. The implementations obtained with Madeo were done for VirtexI of Xilinx and concern only the evaluation of the syndrome, the search of the least reliable symbol and the parity computation. One LUT from Virtex 1 has 4 inputs and 2 outputs and is similar to one Logic element from Stratix.

Madeo adopts a parallel and combinatorial approach to compute the syndrome, the least reliable symbol and the parity. In the case of syndrome, the procedure is quite similar with the SystemC approach and the area efficiency is about 50% with Madeo. This is essentially due to high level optimizations and the ability to manage constant values and non-standard arithmetic (GF(128) arithmetic). For the search of the least

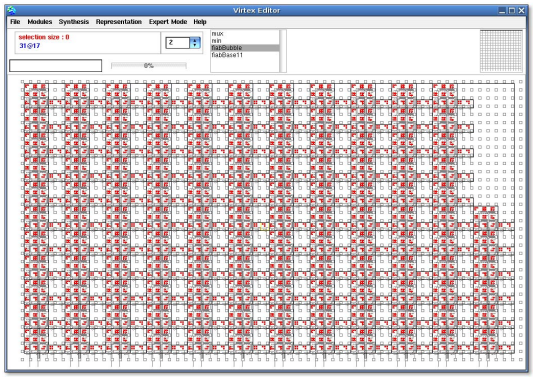

Fig. 13. The least reliable symbol (cascade organization) for $BCH(128,120,4)^2$ placed and routed on Virtex structure using Madeo physical tool

	Madeo	SystemC
Syndrome	217LUTs	~580LUTs
least reliable search	4045LUTs	5×127 comparisons(4 bits) 4×127 shifters with 2 registers of 5×7 and 5×4 flip-flop
parity computation	43 LUTs	8 LUTs (sequential version)

TABLE V

COMPLEXITY COMPARISON FOR THE SYNDROME, THE LEAST RELIABLE SYMBOL AND PARITY

reliable symbol and for the parity computation, the SystemC description is completely sequential, thus it is not possible to make comparison.

VI. CONCLUSIONS

The Madeo framework respects to common practices in the domain of object-oriented programming, and makes use of known CAD algorithms.

The major innovations in the layer decomposition of synthesis tools structure are:

- fine grain reconfigurable architectures does not need a fixed set of operators as it is the case for VLSI and general purpose processors. The definition of these operators and their use can be moved upward in the layers, at the specification language level.
  Common components can just be extracted from a database.
- the optimization usually achieved on operators under compiler control are processed on the intermediate form of the table network in an efficient and more general way. This is also interesting for logic minimization and partitioning software.
- the usual HDL layer hide the component on which the synthesis is achieved. Using a simple grammar, the fine grain can be more efficiently used by system software and placing algorithms.

The practical case of block turbo decoders has been investigated with good results in terms of number of cells, also

leading to speed and power improvements. The specification is expressed in a general purpose object-oriented language in a mix of behavioral and structural high level code developed on the top of a support for Galois Field.

These mechanisms can be reproduced for other input languages and we are working to facilitate the use of the framework for different input syntaxes or programming styles. There are difficulties in using Madeo that are inherent to the synthesis method. The algorithm criteria are radically different from the ones in sequential/imperative programming style. The programmer must have in mind concurrency, speed of convergence, and possible logic complexity. This is the drawback of the concise specifications.

Beside FPGAs, Madeo is currently used to describe data-path, emerging nano-fabrics[23] and to support system synthesis[24]. It is also an efficient tool for reconfigurable architecture design, due to the fast prototyping capabilities.

Acknowledgements.

This project is supported by Brittany region through PRIR contract. Thanks also to Caaliph Andrianisaina and Herve Le Guen for their technical participations to this project.

REFERENCES

[1] C.Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo codes ". IEEE Int. Conf. on Comm. ICC'93, vol 2/3. May 1993, pp.1067-1071.

[2] S. Kerouedan, P. Adde, R. Pyndiah, " How we implemented Block Turbo Codes", annals of telecommunication, Vol.56, Number 7-8, pp.447-454, July-August 2001.

[3] E. Piriou C. Jgo, P. Adde, M. Jézéquel, "System level design using SystemC: a case study of block turbo decoder". XIX Conference on Design of Circuits and Integrated Systems, November 2004.

[4] P. Elias, "Error-free coding", IRE Trans. on Inf. Theory, vol. IT-4, pp.29-37, Sept. 1954.

[5] D. Chase, "A class of algorithms for decoding block codes with channel measurement information", IEEE Trans. Inform. Theory, vol IT-18, pp 170-182, January 1972.

[6] R. Pyndiah, A. Glavieux,A. Picart, S. Jacq, "Near optimum decoding of product codes", GLOBECOM94 , November 1994.

[7] W. W. Peterson, "Encoding and error correcting procedures for the Bose-Chaudhuri codes", IRE Transf. Theory, vol IT-6, pp. 459-470, September 1960.

[8] R. T. Chien, "Cyclic decoding procedures for the Bose-Chaudhuri Hocquenghen codes", IRE Transf. Theory, vol IT-10, pp. 357-363, October 1964.

[9] A. La Rosa, L. Lavagno, C. Passerone, "Implementation of a UMTS turbo decoder on a dynamically reconfigurable platform", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Volume 24, Issue 1, Jan. 2005.

[10] J. R. Cavallaro, M. Vaya, "VITURBO: A reconfigurable architecture for Viterbi and turbo decoding", IEEE ICASSP2003, pp.497-500, April 2003.

[11] I. Atluri, T. Arslan, "Reconfigurability-power trade-offs in trubo decoder design and implementation",IEEE ISVLSI'04, Februray 2004.

[12] J. Liang, R. Tessier, D. Goeckel, "A dynamically-reconfigurable, power-efficient turbo decoder", in the Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, California, April 2004.

[13] V. Betz and J. Rose, A. Marquardt, "Architecture and CAD for deep-submicron FPGAs", Kluwer Academic Publishers, 1999.

[14] R. Lysecky, F. Vahid, S. X.-D. Tan, "Dynamic FPGA routing for just-in-time FPGA compilation", IEEE/ACM DAC2004, June 2004.

[15] L. Bossuet, G. Gogniat, J.-L.Philippe, "Fast design space exploration method for reconfigurable architectures", ERSA'03, June 2003.

[16] S. Choi, J.W. Jang, S. Mohanty, V.K. Prasanna, "Domain specific modeling for rapid system level energy estimation of reconfigurable architectures", ERSA'02, June 2002.

[17] Goldberg A., Robson D., *Smalltalk-80: the language and its implementation*, Addison-Wesley Longman Publishing Co., Inc., 1983.

[18] Lagadec L., Pottier B., Villellas-Guillen O., "An LUT-Based high level synthesis framework for reconfigurable architectures", *Domain-Specific Processors : Systems, Architectures, Modeling, and Simulation*, Marcel Dekker, p. 19-39, November, 2003.

[19] Llopis J.-L., Pottier B., "Smalltalk blocks revisited, a logic generator for FPGAs", *in* , J.-M. Arnold, , K. Pocek (eds), *Field programmable Custom Computing Machine (FCCM'96)*, IEEE press, Napa, CA, 1996.

[20] Paar C., Rosner M., "Comparison of Arithmetic Architectures for Reed-Solomon Decoders in Reconfigurable Hardware", *IEEE Symposium on FPGAs for Custom Computing Machines*, Los Alamitos, CA, p. 219-225, 1997.

[21] Sentovich E. M., Singh K. J., Lavagno L., Moon C., Murgai R., Saldanha A., Savoj H., Stephan P. R., Brayton R. K., Sangiovanni-Vincentelli A., SIS: A System for Sequential Circuit Synthesis, Rapport Technique n° UCB/ERL M92/41, DEECS, Berkeley, May, 1992.

[22] Fabiani E., Gouyen C., Pottier B., "Intermediate level components for reconfigurable platforms", *Synthesis, Architectures and Modeling of Systems (SAMOS 3)*, vol. 3133, Springer-Verlag, Samos, Grece, 2003.

[23] Fabiani E., Lagadec L., Pottier B., Poungou A., Yazdani S., "Abstract execution mechanisms in a synthesis framework", *in* , N. Carter, , S. Goldstein (eds), *Workshop on Non-Silicon Computations (NSC3), (conjoint avec ISCA 2004, ACM et IEEE)*, June, 2004.

[24] Fabregat G., Leon G., Le Berre O., Pottier B., "Embedded system modeling and synthesis in OO environments. A smart-sensor case study", *in* , G. Gao, , K. Palem (eds), *CASES'99*, Oct, 1999.

[25] Gao M., Jiang J.-H., Jiang Y., Li Y., Sinha S., Brayton R., "MVSIS", *International Workshop on Logic Synthesis*, June, 2001.

[26] Lin B., Whitcomb S., Newton A., "Symbolic don't care and equivalence in high level synthesis", *Logic and architecture synthesis*, Elsevier, 1991.