ALOE-based flexible LDPC decoder

Ismael Gomez, Massimo Camatel, Jordi Bracke,
Vuk Marojevic, Antoni Gelonch
Dept. of Signal Theory and Communications
Universitat Politecnica de Catalunya
Av. Canal Olímpic s/n 08860 Castelldefels, Spain
{ismael.gomez, massimo.camatel, jordi.bracke,
marojevic, antoni.gelonch}@tsc.upc.edu

Fabrizio Vacca, Guido Masera
Departamento di Elettronica
Politecnico di Torino
Corso Duca degli Abruzzi 24, Torino
{fabricio.vacca, guido.masera}@polito.it

***Abstract*—Radio communications terminals and infrastructure tend to support an increasing range of algorithms and radio access technologies. Flexible processing platforms are therefore needed for supporting multi-standard or heterogeneous radios. Channel decoding is one of the most computing demanding digital signal processing blocks of a radio transceiver. At the same time, it provides a high degree of implementation flexibility as well as facilitates dynamic parameter adjustments. This paper presents a flexible LDPC decoder implemented on an FPGA device following the ALOE middleware design paradigm. We analyse the middleware efficiency in terms of flexibility versus resource requirements. The results show a relative middleware area overhead of 32 %.**

ALOE middleware, flexible LDPC, reconfigurable logic, SDR

I. INTRODUCTION

The continuous improvements in the micro-electronic technology have made conceivable the integration on the same Integrated Circuit (IC) millions of MOS transistors and logic gates. This makes possible the design of novel integrated architecture with enhanced capabilities. These augmented possibilities require novel design paradigms in order to catch all of them, particularly the resorting to flexible architectures able to easily adapt to different applications and algorithms [1]. This evolution of digital processing architectures in the direction of an increasing level of flexibility is particularly evident in the field of wireless communication systems. Since the number of radio standards is growing very fast and the diversity among the standards is also increasing, there is a need for a processing solution capable of handling as many standards as possible. In particular, the idea of software-defined radio (SDR) implies the implementation, in the future, of flexible multi-standard radios, supporting all these different standards, with no degradation in terms of achievable data rate or transmission reliability. Flexible platforms are necessary to this purpose.

In this context, Multi-Processor System-on-Chip (MP-SoC) architectures are being widely investigated these last years in order to accommodate the increasing throughput and flexibility requirements of emerging wireless communication standards.

Among the several functionalities specified in wireless communication standards, one of the most demanding operations is channel decoding, which contributes at least 40 % to the total computational complexity of the physical layer of a wireless system. Each new wireless standard typically increases the data rate, while keeping low the occurrence of errors in the transmissions. Moreover, depending on some external conditions, each standard provides different profiles. Thus, an integrated circuit designed for telecommunication purposes has to exploit a certain degree of flexibility in order to tackle all these profiles. More flexible architectures can have also support for future out-coming standards.

In this context, the present work proposes and evaluates a new fully flexible solution for the implementation of multi-standard and multi-mode channel iterative decoder supporting generic Low-Density-Parity-Check (LDPC) codes [2]. These codes are able to achieve high performances in terms of bit error rate (BER) although they have very high computing requirements at the receiver side. At present, several applications, such as the digital satellite broadcasting system (DVB-S2), Wireless Local Area Network (IEEE 802.11n) and Metropolitan Area Network (802.16e) incorporated them.

In MP-SoC architectures for iterative decoders, several independent data blocks can be simultaneously decoded on different processors. In addition to node computational capabilities, an interconnect structure is necessary to support the iterative message exchange among variable and check nodes. In this context, Network-on-Chip (NoC) has recently emerged as a new paradigm [3] allowing coping with these major design issues, and more particularly with the on-chip interconnection needs. Efficient MP-SoC architectures assume heterogeneous processing elements (PE). Therefore it is necessary to define an optimum mapping of tasks to the set of PE maximizing computation efficiency [4]. Moreover, decoder throughput can be adapted in time balancing the total amount of resources assigned to it. Due to the ability of the decoder algorithm to be parallelized, the more PEs assigned to it, the higher performance.

In addition, future flexible radios based on large MP-SoC architectures must embrace platform-independent component-based designs in order to maintain profitable production costs. Operating Environments and Middleware for SDR applications envisages as an efficient solution to achieve the aforementioned run-time and design-time flexibility.

The Abstraction Layer and Operating Environment (ALOE) is an open source SDR framework with real-time computing resource management capabilities [5]. The middleware supports GPPs and DSPs through a lightweight static memory implementation in C. A reduced version of ALOE is also available in VHDL, supporting some of the middleware services, and currently targeted for Virtex-5 FPGA devices. Therefore, ALOE provides a single interface to manage processors and reconfigurable logic. The middleware is capable to map waveform components to processing devices on-line, as a function of real-time deadlines [4].

This paper presents a flexible LDPC decoder architecture based on the ALOE middleware for FPGAs. The decoder exploits flexibility at all levels: design-time, as the processing devices are designed without any target platform knowledge; and run-time, because the decoder is able to change the code without need to redesign the processing devices or the interconnection network. The aim of the project was never the performance or energy efficiency. Conversely, it tries to prove the suitability of middleware to increase global efficiency as well as to analyse its impact in the performance, energy and area consumption. The organization of the paper is as follows: next section presents related work in flexible LDPC decoders; Section III presents our decoder architecture based on the ALOE middleware; Section IV explains how the system has been validated at behavioural simulation level whereas Section V presents area occupation and performance results. The paper ends with some conclusions on the costs and benefits of introducing ALOE in hardware devices.

II. Related Work on LDPC Decoders

An LDPC code is a linear block code characterized by a very sparse parity check matrix, H. From the behavioural point of view, the decoding process can be divided in two sets of tasks, associated to the N Variable Nodes (VNs) related with the rows, which handle the codewords, and M=N-K Check Nodes (CNs) (where K is the information word size) , which implement the parity-check constraints, that are related to the columns of the H matrix. Multiple processing elements are normally allocated to execute these node tasks.

LDPC codes can be represented in term of a bipartite graph, called Tanner Graph. Such representation of error correcting codes is very useful since their decoding algorithms can be explained by the exchange of information along the edges of these graphs. The VNs receive the intrinsic information λ_i from the channel and update them depending on the results of the parity check equations computed at the CNs; this process is iterated several times until a converge criterion is met. This algorithm is known as "Two Phase Message Passing" (TPMP) or Belief Propagation Algorithm (BPA).

Usually, the more architecture flexibility the less performance it can achieve, as internal structures are not optimized to any specific application. A straightforward approach to the implementation of a decoder for a single LDPC code is to instantiate all nodes of the Tanner graph: a fully parallel architecture is obtained, with the potential for very high throughput [7], however routing congestions and the lack of flexibility make this approach impractical. In partially parallel architectures [8], the nodes of the Tanner's graph are processed in time multiplexed way by means of a number of processing elements lower than the check or variable nodes. The original two phase decoding has now given way to the so called layered or shuffled decoding [9] [10], which results in approximately two times faster convergence of the algorithm. As partially parallel architectures introduce a problem of conflicting accesses to the memories containing the exchanged messages, a proper network is required to connect processing elements and memories. The complexity of this network basically depends on the structure of the H matrix. Different approaches have been investigated in [11], [12], [13] among others.

The requirements of very low error rate, very high throughput and increased flexibility of the implementation, can be better achieved by means of multiprocessor architectures, or Multi-Processor System-on-Chip (MP-SoC) architectures, the inherent parallelism of the iterative LDPC decoding algorithms is exploited to efficiently partition the decoding task. Application Specific Instruction set Processors (ASIP) recently emerged as a promising solution for the implementation of flexible decoders, capable to greatly improve programmability, while still allowing high throughput and efficiency, thanks to specialised processing units. A recent example is given by the multi-mode decoder architecture for convolutional and structured LDPC codes presented in [14]. Another ASIP designed to decode convolutional, turbo and LDPC codes is presented in [15].

Also for the MP-SoC implementation approach, a network has to support the communication demands of the different processors without degrading the throughput of the overall system. Conventional on-chip buses become inefficient in large systems and the nanotechnology integration issues (propagation delay, crosstalk, etc.) make their use impractical. In this context, Network-on-Chip (NoC) has emerged as a new paradigm allowing coping with these design issues. A NoC solves the problem of scalability, facilitating the on-chip integration of several hundreds of processing components.

A first attempt to design a NoC based LDPC decoder is described in [16], which presents an architecture that supports LDPC codes up to the size of 1024 variable nodes at the cost of a high area occupation and power consumption.

Two NoCs for the decoding of generic LDPC codes are discussed in [17], where a binary de Bruijn NoC with on-line dynamic routing is presented and compared to a communication structure called Zero-Overhead NoC with offline static routing. This approach allows full flexibility, optimises the FIFO sizes, and minimizes the network latency.

III. Decoder Architecture

The middleware based decoder proposed in this paper accesses middleware services through an ALOE controller. This controller corresponds to an Application Programming Interface (API) of common operating systems: a standard interface hides device-specific peculiarities. Therefore, components can be ported from one device to another (or from one memory section to another) without redesigning them. The computing resource manager, the time reference and the execution control management is centralized by the ALOE managers. These elements, however, are executed more efficiently on general-purpose processors; ALOE permits to run managers in a Linux-PC.

For simplicity reasons current decoding processing elements (PEs) incorporated, both, CNs and VNs processing nodes building a homogeneous array of PEs. Nevertheless, next step can potentially improve efficiency by differentiating the PEs carrying out CN and VNs nodes, setting up then a heterogeneous MP-SoC. Additionally, in current version, control and management tasks are assigned to the GPP (Linux) what in some sense assume the emulation of a heterogeneous MP-SoC environment mixing ASIPs, GPP and logic. The middleware offers the designer a single interface to manage all kinds of PE.

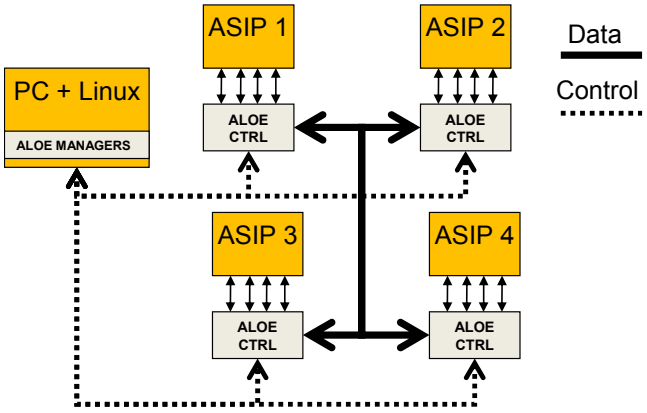

Fig. 1. LDPC ALOE based Decoder

Since the aim of the work is not achieving certain throughput, it is not necessary to compute the number of required PE or the degree of parallelism attaining that threshold. One ALOE controller is needed for each PE; therefore, the degree of concurrency is irrelevant to the relative overhead analysis. However, it is interesting to prove the validity of the middleware in a relatively complex scenario, hence we chose four concurrent PE. Fig. 1 depicts the architecture of the decoder. It consists of four processing elements (PEs), four ALOE controllers and one Linux-PC running the software-based ALOE managers. Although the network topology in the figure uses one interface for each remote processor, ALOE abstracts the network topology and, thus, supports any kind of network topology, for example, shared channels.

A. Node Processor

A general distributed LDPC decoder assigns CN and VN nodes to a set of processors. The throughput scales with the number of processors (parallelism), in general. Our architecture uses an Application-Specific Instruction-Set Processor (ASIP) for the node processing. Typically, ASIPs combine a general-purpose with an application-specific instruction set. Therefore, although tailored for LDPC decoding, other processing tasks can be accomplished. If the ASIP is underutilized, the resource manager can map tasks to it making a better use of the resource. Our implementation, however, provides only a specific instruction-set for LDPC decoding.

Several logic nodes (CNs and VNs) are assigned to each physical processor. They receive messages from other nodes, perform computations and send messages back. As the decoder is not designed for a specific H matrix, each message must carry the destination node index within the data. The node's physical location is not known to the sender node; ALOE manages the routing of messages through physical channels.

Besides the program memory, the processor accesses a set of other data memory banks dedicated to several purposes. The data in the memory defines the interconnection of CN and VN nodes and hence specifies the LDPC code. During the initialization phase, the ASIP retrieves from the ALOE manager the contents of these memories determining the operating code. The following memory banks are defined:

- *VNi_mem* contains the connections between each VN node and the CN nodes and the data processed by the VN. Each memory position, 32-bit wide, indicates the pair of nodes (VN and CN, 12-bit each), the information data (6-bit width) and the interface towards the destination node PE (2-bit).
- *CNi_mem* is basically organized like *VNi_mem*, except for the field order. (It contains the connections between each CN node and the VN nodes with the R messages.) The data field is filled with the R message produced by CN to the VN.
- *VN_to_CN_mem* has the same format of *VNi_mem*. It contains the Q messages received by the VNs from other PEs to the CNs in the local PE.
- *CN_to_VN_mem* has the same format as *CNi_mem* but containing the R messages.
- *Input* memory contains the input LLR data of VN nodes (b-bits).
- *Output* memory contains the output LLR representing the decoded data.

The purpose of having different memory banks instead of a single one is because of the algorithm's concurrent memory accesses. Instead of accessing multiple offsets in sequential cycles, the node computation is performed in a single cycle and the data is concurrently read from multiple buses.
Regarding the processing logic, four specific instructions are available:

- *Create_VN* performs the VN node computations. It reads R messages from the *CN_to_VN_mem* and stores Q messages in the *VNi_mem*;
- *Send_VN* delivers Q messages stored in the *VNi_mem* to the network or, in case that the remote CN is in the same PE, to the *VN_to_CN_mem*;
- *Create_CN* performs the CN node computations. It reads Q messages from the *VN_to_CN_mem* and stores R messages in the *CNi_mem*;
- *Send_CN* delivers R messages stored at the *CNi_mem* to the network or, in the case where the remote VN is in the same PE, to the *CN_to_VN_mem*.

B. ALOE Controller

Each ALOE controller can manage one or more waveform components. A controller is necessary when an interface or service needs to be abstracted. For example, one controller suffices if the device has only one external interface. Nevertheless, components can exploit NoC efficient routing, placing an ALOE controller before each network node. Fig. depicts the latter concept: several types of processing devices coexist in the same environment—in this case, the environment is a single silicon die. Processors with an operating system run ALOE services as background tasks. Conversely, single-threaded devices e.g. ASIPs without an operating system or Dynamically Reconfigurable Areas (DRA) interface ALOE through a parallel logic block. Note how platform services—the NoC, converters, RF front-ends and external interfaces—are *abstracted* by the middleware and therefore accessible by the waveform components.

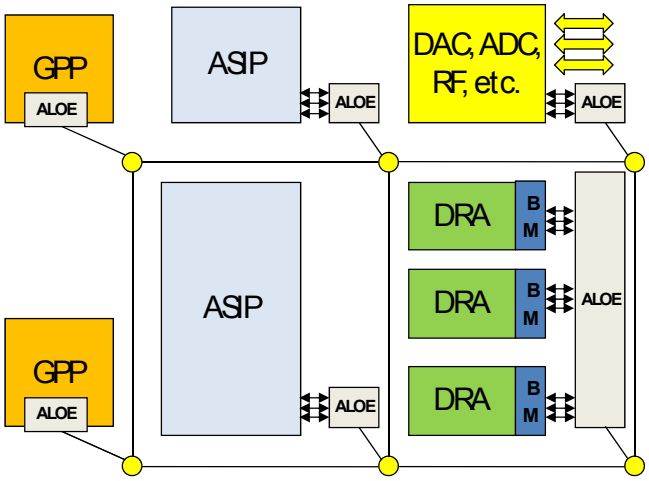

Fig. 2. ALOE Abstraction Architecture

The integrated ALOE computing resource manager enables to automatically map application tasks to a set of PEs using the tw-mapping algorithm [4]. The algorithm does not address loops and is, thus, inappropriate for Tanner Graphs. The code selection is performed offline, whereas the node mapping and routing is computed with the algorithm presented in [17]. ALOE controls the synchronization and scheduling of components following a specific execution pattern: computing resource *time* is split in discrete slots where each component (ASIP) is executed once; a message produced by one component at slot *n* cannot be read by destination component until slot *n*+1 (pipeline). Besides synchronizing components, the pipeline exploits concurrency at the cost of additional delay. The component designer is not aware of the network latency between processors and, therefore, does not need to define any scheduling or synchronization mechanisms. These issues are automatically controlled by ALOE pipelined execution pattern.

The ALOE controller has been designed to be modular, scalable and portable. Device-specific services or functions are implemented in the platform-dependant part, the HW API. For instance: FIFO internal interfaces, external signals, timers, memories and so forth. The rest of the part is organized in components, each with a different set of functions (Fig. 3):

- FRONT-END: routes control packets. Accesses the external control interfaces through the HWAPI. At boot registers to the ALOE managers obtaining a unique Id.
- EXEC: Controls the execution status of the components and monitors real-time operation.
- STATS: Requests initialization configuration to the manager.
- SWLOAD: Allocates component resources and configures internal and external interfaces.
- SYNC: Synchronizes local time with master reference
- BRIDGE: Route data packets. Accesses external data signals through the HWAPI. At boot, sends own identification to neighbours to automatically discover network topology.
- SWAPI: The fixed interface to the components.

The last part, the SWAPI, is the only part the application component designer has to take into account. The interface provides the following signals:

- A set of input/output data FIFO-like interfaces.
- A unique time reference for all application components in all processors (synchronized by SYNC).
- Two status signals to control component execution. One to set the status to INIT, RUN, PAUSE or STOP and another to check a successful status change.

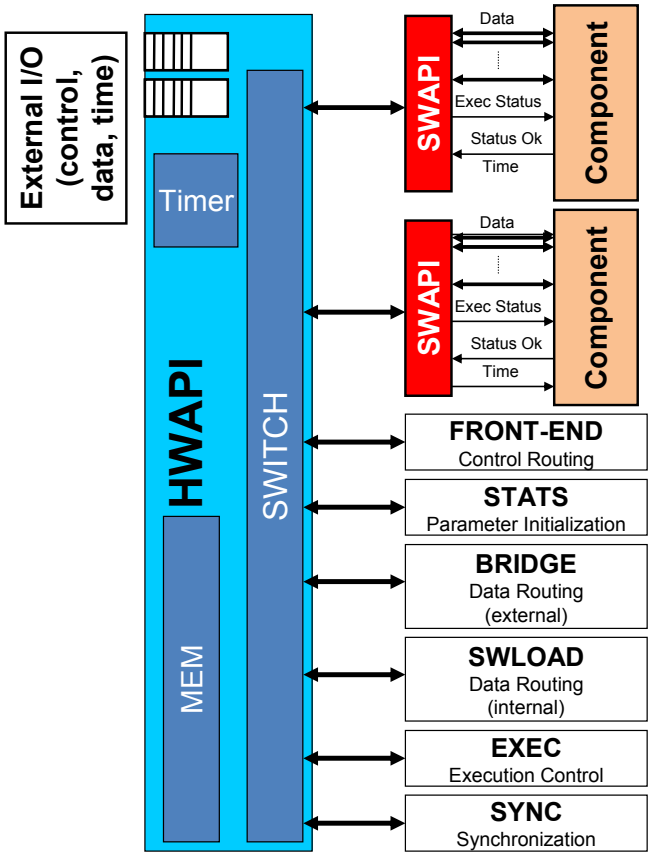

Fig. 2. LDPC ALOE based Decoder

C. *Network Abstraction*

In order to increase the communications efficiency, the signals between components are shared in common channels. Typically, the NoC is dimensioned as a function of the expected link loads for a certain application (e.g. LDPC code). Conversely, a platform-independent design cannot make such a consideration, achieving higher flexibility at the cost of lower efficiency. Given certain network topology and link bandwidths, efficiency is obtained through the process of task mapping. For reconfigurable devices the NoC is assumed to be static as it, generally, exhibits the maximum contribution to the energy consumption. A static interface from the reconfigurable component to the NoC is, therefore, necessary. Interfacing only with the ALOE SWAPI enables to increase the design reutilization as the NoC router improvements will not force a component redesign.

The ALOE framework is not limited to any specific NoC topology or architecture. ALOE enables the application to take benefit of the NoC, if any, hiding its internal mechanisms.

IV. Simulations

The correct behaviour of the system has been tested through behavioural simulation. Timing simulations and implementation verifications have not been realized. The simulation scenario presents some challenges. It is impossible to verify the functionality of the ALOE controller without the interface to the Linux ALOE host. A dynamic testbench needs to write to the model signals the bits of the received packet. In addition, ASIPs are simulated with the CoWare Processor Designer Debugger [18], which is incompatible with other VHDL simulation tools.

Therefore, a C-VHDL interface is required (VPI interface). The GHDL simulator [19] is a GNU behavioural simulator with VPI support. This interface communicates C programs with the signals of the VHDL models. We developed two components for integrating the whole scenario: one from the model signals to the TCP/IP ALOE interfaces and one towards the CoWare debugger. The model generates a periodic *wait* event which, when simulated with GHDL, calls our bridging custom function. The function receives TCP/IP packets and writes the data to the model input ports. The data generated at the output ports of the ASIP is written to the model input ports it is connected to. Then a model cycle is simulated. When it finishes, model output ports data is written either to the ASIP input ports or to the TCP/IP sockets.

Due to licensing limitations, each CoWare simulator must run in a separate PC. Fig. 4 depicts the complete scenario where, for simplicity, only two PC are shown. The simulation is performed as follows:

1. we start ALOE managers in one computer;
2. we start the GHDL-CoWare simulators on four other computers;
3. the ALOE FRONT-END in each controller registers to the ALOE Linux manager;
4. the ALOE BRIDGE in each controller identifies data interfaces towards its neighbours;
5. the ALOE Linux manager generates the processor interconnection matrix [4] with the data collected from the BRIDGE in each PE. Note that this step proves that in a reconfigurable logic scenario the system is able to dynamically detect changes in the number and interconnection topology of PEs;
6. we use tool in [17] to compute CN and VN routing and node assignation for PE network architecture and save result in a configuration file;
7. we use the ALOE Linux interface to set the application status to INIT. Components request configuration to the ALOE STATS who requests to the Linux manager who reads the previously produced file. Now CN and VN node assignation and routing is performed, thus code is selected;
8. we use the ALOE Linux interface to set the application status to RUN;
9. we use the ALOE Linux interface to set the application to STOP. We repeat step 7 and 8 with another code. Thus we are able to dynamically change the LDPC code at run-time.

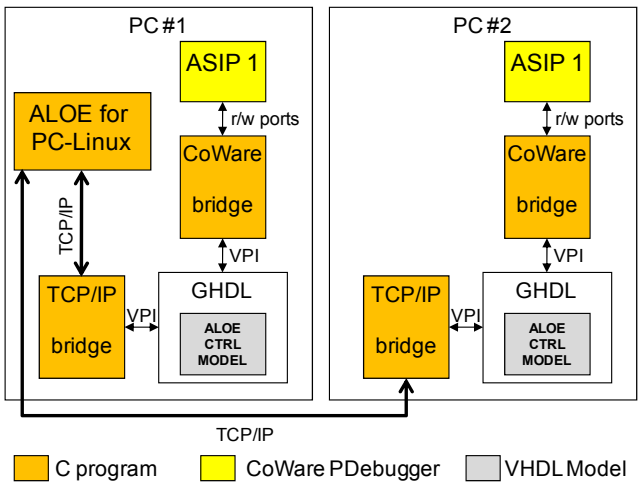

Fig. 4. Simulation Scenario

V. Results

The decoder has been synthetised for a Xilinx Virtex-5 FX70 FPGA. Table I shows the resource occupation for the ASIP component, optimized for speed.

TABLE I. ASIP Resource Occupation

Slice Registers	1892
Slice LUTs	3360
Memory (Kbits)	1120

The ASIP's VHDL code was automatically generated by the CoWare Processor Generator from the SystemC model. The processor uses 7 memories in total:

- the ROM containing the program code uses 4 Kbytes of memory;
- 2 RAMs of 4 Kbytes for the input and output memories (*input_mem* and *output_mem*);
- 4 RAMs of 32 Kbytes the *VNi_mem*, *CNi_mem*, *VN_to_CN_mem* and *CN_to_VN_mem* memories.

Clearly, the amount of memory used by the ASIP is significant. However, and as stated previously, optimization was not the aim of the project. Table II shows the resource occupation for a single ALOE controller, which was optimized for area in this case. The controller and the PE can operate synchronously since the maximum achievable frequency of the former is greater than the frequency of the latter.

TABLE II. ALOE Controller Resource Occupation

Slice Registers	756
Slice LUTs	1748
Memory (Kbits)	768

Considering the entire decoder contribution without the NoC, four ASIPs and four 4 ALOE controllers, the total decoder resource utilization can be computed as in Table III. Table IV depicts the relative overhead of ALOE.

TABLE III. Decoder Resource Occupation (4 ASIP and 4 ALOE)

Slice Registers	10592
Slice LUTs	20432
Memory (Kbits)	7552

TABLE IV. ALOE Relative Overhead

Slice Registers	28 %
Slice LUTs	34 %

The relative overhead for this implementation is considerable. Before optimizing the design, we should mention that the flexibility obtained by the management elements increases if the PE size increases. Then, the relative overhead of ALOE would be lower, since additional tasks can be assigned to the same PE. In these situations the resources spent for implementing the resource managers have even more sense, because the coordination of the available computing resources is more complex. Relative overhead can also be reduced if the same ALOE controller manages several PE. In this case, however, the NoC efficiency can not be exploited since ALOE has to route signals between PE.

VI. Conclusions

Future user terminals or base stations will require higher levels of flexibility. Consider an MP-SoC dimensioned for wideband channel decoding that is sporadically used to decode multiple narrowband channels. Since throughput requirements will be lower, all CN and VN nodes can be assigned to a single PE being able to operate other channels simultaneously on the remaining PEs. Furthermore if the supporting hardware is also flexible, e.g. dynamically reconfigurable circuits, the range of possibilities increases the management complexity. The system thus needs computing resource awareness and management support while providing tools to dynamically synchronize and schedule the execution of components. In this paper we have proven that ALOE is capable to automatically detect the number and network topology of PEs in a reconfigurable logic device.

It is important to recognize that the increasing flexibility of channel decoders increases the area and energy consumption, as shown in this paper. A flexible decoder compromises modifications of code or coding schemes. Some allow only offline modifications (at design time), whereas others also facilitate runtime modifications. The ALOE provides a set of tools for centralizing the management of runtime decoder parameterization. This flexibility comes at the price of resource overhead. Although our design has not been optimized, the incorporation of the ALOE middleware will always incur in more area occupation. Nevertheless, future flexible terminals can overcome these additional resource overheads using reconfigurable logic and shared resource infrastructure.

ACKNOWLEDGMENT

This work was supported by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM++ (contract n. 216715).

REFERENCES

[1] A. Polydoros. “Algorithmic aspects of radio flexibility”. Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on, pages 1-5, Sept. 2008.

[2] R. G. Gallager, “Low-density parity-check codes”, IEEE Transactions on Information Theory, pages 21-28, Jan. 1962

[3] F. Vacca, H. Moussa, A. Baghdadi, and G. Masera. “Flexible architectures for ldpc decoders based on network on chip paradigm”, in Euromicro Conference on Digital System Design, 2009.

[4] V. Marojevic, X. Reves, A. Gelonch, "A Computing Resource Man agement Framework for Software-Defined Radios," IEEE Transactions on Computers, pp. 1399-1412, October, 2008

[5] Ismael Gomez, Vuk Marojevic, Jose Salazar, Antoni Gelonch, "A Lightweight Operating Environment for Next Generation Cognitive Radios," Digital Systems Design, Euromicro Symposium on, pp. 47-52, 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008

[6] G. Masera, F. Quaglio, F. Vacca, "Finite precision implementation of LDPC decoders", IEE Proceedings - Communications, Volume 152, Issue 6, December 2005, Pages 1098 – 1102

[7] A. Blanksby and C. J. Howland, “A 690-mW 1-Gb/s, Rate-1/2 Low-Density Parity-Check Code Decoder,” IEEE Journal of Solid-State Circuits, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[8] T. Brack, F. Kienle, and N.Wehn, “Disclosing the LDPC code decoder design space,” Proc. DATE 2006, vol. 1, pp. 200–205, Mar. 2007.

[9] M. Mansour and N. Shanbhag. Low-power VLSI decoder architectures for LDPC codes. Low Power Electronics and Design, 2002. ISLPED02. Proceedings of the 2002 International Symposium on, (Monterey,USA), pages 284.289, Aug. 2002.

[10] D. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. Signal Processing Systems, SIPS 2004. IEEE Workshop on, (Austin, USA), pages 107-112, Oct. 2004.

[11] E. Boutillon, J. Castura, and F. Kschischang, “Decoder-first code design,” in Proc. 2nd International Symposium on Turbo Codes & Related Topics, Brest, France, Sept. 2000, pp. 459–462.

[12] J. Tang, T. Bhatt, and V. Sundaramurthy, “Reconfigurable Shuffle Network Design in LDPC Decoders,” in Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP’06), Sept. 2006, pp. 81–86.

[13] A. Tarable, S. Benedetto, and G. Montorsi. Mapping interleaver laws to parallel turbo and ldpc decoders architectures. IEEE Trans. Inform.Theory,, 50(9):2002-2009, Sept. 2004.

[14] Steffen Kunze, Emil Matus and Gerhard P. Fettweis, “ASIP Decoder Architecture for Convolutional and LDPC Codes”, in Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'09), Taipei, Taiwan, 24.-27. May 2009

[15] M. Alles, T. Vogt, N. Wehn. “FlexiChaP: A Reconfigurable ASIP for Convolutional, Turbo, and LDPC Code Decoding”, In Proc. IEEE 5th International Symposium on Turbo Codes & Related Topics, pages 84 - 89 , September, 2008, Lausanne, Schweiz.

[16] T. Theocharides, G. Link, N. Vijaykrisham, and M. J. Irwin, “Implementing LDPC Decoding on Network–On–Chip,” in Proc. 18th Int. Conf. on VLSI Design (VLSID’05), 2005, pp. 134–137

[17] F. Vacca, H. Moussa, A. Baghdadi, G. Masera, "Flexible Architectures for LDPC Decoders based on Network On Chip Paradigm", DSD 2009, Digital System Design, Patras, Greece, Aug. 2009

[18] CoWare Processor Designer Web Site http://www.coware.com/

[19] GHDL Simulator Web Site http://ghdl.free.fr/