EMBEDDING POLYNOMIAL TIME MEMORY MAPPING AND ROUTING ALGORITHMS ON-CHIP TO DESIGN CONFIGURABLE DECODER ARCHITECTURES

Saeed-ur-REHMAN [1], Awais SANI [1,2], Cyrille CHAVET [1], Philippe COUSSY [1]

[1] Lab-STICC, Université de Bretagne-Sud, Lorient, [2] SATT Ouest-Valorisation, Rennes

Abstract- **To fulfill the high data rate requirement of current telecommunication standards, error-correction codes decoders are implemented on parallel architectures leading to memory conflict problem. Different memory mapping approaches are proposed in the literature to solve this problem. However, these approaches can only be executed offline due to their computational complexity and resultant memory mapping is stored in dedicated ROM in order to drive the network for a particular block length. Unfortunately, to support several block lengths, multiple ROMs are required which results in huge hardware cost. In this article, we propose a novel online memory mapping architecture that consists of online mapping generator and RAM to support multiple block lengths on single chip. Online mapping generator performs two functions: First, it executes polynomial time memory mapping algorithm online and secondly, it generates command words for Benes network by using a simplified routing algorithm. Whenever new block length needs to be decoded, online mapping generator outputs addressing and command words at runtime to update the RAM. Experimental results show that significant reduction in time and memory cost is obtained while implementing polynomial time memory mapping algorithm on-chip as compared to state of the art approaches.**

1. INTRODUCTION

In telecommunications domain, turbo-like codes [1][2] have been adopted by many standards [15]-[17] to exploit their excellent error-correction capability. However, executing these iterative algorithms sequentially results in a prohibitive processing latency and parallelization of decoder architectures to achieve very high throughput at low power budget is thus required (e.g. [24]). Unfortunately, parallel architectures lead to conflicting memory accesses which can dramatically decrease the system's performances. Many works which can be gathered into two families of solutions are proposed in literature to avoid or reduce collision problems: (1) *run time approaches* in which extra memory elements and control logic are used in order to serialize conflicting accesses (e-g [5]-[9]) and (2) *design time approaches* that find a conflict-free memory mapping (e-g [11]-[13],[18], [23]). In the first family, [5]-[7] propose to design dedicated interconnection network called LLR distributor to tackle conflict problem for turbo codes. Butterfly and Benes are the two heterogeneous multistage networks investigated in [8] to increase the scalability and to meet higher throughput requirement on flexible communication network. Binary de Bruijn interconnection network is presented in [9] to provide scalability and allow any permutation to be routed efficiently. Communication conflicts are managed due to its path diversity by deflecting the conflicting packets appropriately until they reach the target processor rather than blocking or buffering them. However, all these flexible networks used in *run time approaches* suffer from large silicon area due to increased buffer control architectures necessary to manage conflicting packets. Furthermore, due to the conflict management mechanisms, delay is introduced which degrades the maximum throughput and makes sometimes these approaches inefficient for high data rate architectures.

The second family of approaches consists of different algorithms proposed to provide concurrent accesses to all processing elements without any conflict. For this purpose, pre-processing is realized off-line on a computer to determine the memory locations for each data element used in the computations. Coloring a conflict graph with a minimum number of colors is NP-complete problem as shown in [10]. In addition graph coloring is not able to find a conflict-free memory mapping as shown in [22] when data are accessed several times like it is the case in LDPC. In order to find conflict free memory mapping in turbo decoder, works in [11] and [12] present meta-heuristic-based methods. The authors proved that conflict free memory mapping always exists to tackle collision problem for every code. However, the proposed approach is based on a simulated-annealing algorithm, so the user cannot predict the complexity of the algorithm. In [13], another heuristic is proposed which finds conflict free memory mapping for a given interconnection network. The approach given in [23] also finds conflict free memory mapping by adding extra logic elements like registers to the architecture in order to respect a targeted interconnection network. However, all these heuristics fail to remove the computational complexity of the problem and requires off-line preprocessing to map data in different memory banks for different block lengths and parallelism degree (see [21]). To overcome the problem of computational complexity, the authors of [18] recently introduced a polynomial time approach which is based on Euler partitioning. Complexity is greatly improved but authors still use the algorithm off-line to find conflict free memory mappings. In all the cases, existing design time approaches all require that resulting memory mapping is implemented by storing a set command words in dedicated ROM in order to drive the architecture. Unfortunately, to support several block lengths, multiple ROMs are required which results in important hardware cost.

Recently, an approach has been introduced in [21] to combine design and run time approaches by embedding [11] and [13] on-chip to propose flexible decoders. Target architecture includes several processing elements and memory banks interconnected through a crossbar network. Decoder architecture executes mapping algorithm online whenever new block length needs to be decoded and stores the generated command words in two RAMs that replace the set of classically used ROMs to provide conflict free access to the memory for this block length. However, the computational complexity of mapping approaches existing at that time made the solution infeasible for the current telecommunication standards.

In this article, we propose to embed the polynomial time memory mapping algorithm [18] on-chip in order to execute it at runtime to solve conflict problem. In addition, Benes network is targeted and a simplified routing algorithm that is executed on-chip along with the mapping algorithm is proposed. Indeed, crossbar network, as targeted in [21], provides fast memory access and automatic command words generation, whereas Benes network provides small latency but requires routing map to access memories. However, for medium and high level of parallelism, the important hardware complexity of crossbar network makes it unaffordable. Hence, significant reduction in time and memory cost can be obtained by embedding polynomial time memory mapping algorithm on-chip compared to state of the art approaches.

The rest of the paper is organized as follows. Section 2 motivates the importance of memory mapping approaches by highlighting the memory conflict problem. Section 3 describes the proposed approach in which we define the mapping algorithm, the routing algorithm and the target architecture. Section 4 presents results in which different processors are used to measure the computational complexity of the memory mapping and of our embedded routing approach. Finally, section 5 concludes our work.

2. PROBLEM FORMULATION

In this section, we briefly explain memory access conflict problem to motivate the use of memory mapping approaches. The problem

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

can best be explained through a simple example of turbo codes. We introduce two matrices, one is related to the natural order access and the other is related to the interleaved order access. Each matrix has *P* rows, related to the processing elements, and *T/2* columns, related to the time instances. Data in each row are processed by the processing element associated with this row. Similarly, the *P* data elements in each column need to be accessed in parallel by *P* processing elements for parallel decoding architecture. Figure 1 depicts two access matrices in which we have block length *L=12, P=3, T = 8.*

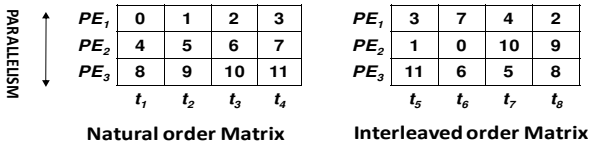
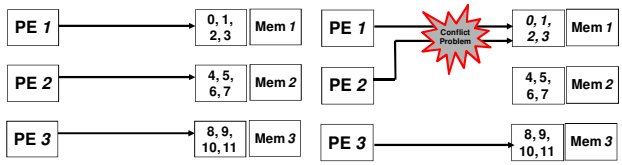


Figure 1: Data Access Matrices

To increase memory bandwidth, *B=3* memory banks are used so that each processing element can concurrently get data in parallel. Data are stored in banks in such a manner that at each time instant in natural order, all the processing elements always access different memory banks concurrently as shown in Figure 2.a. However, by using this memory mapping, multiple processing elements will access the same memory bank at each time instance in interleaved order as shown in Figure 2.b for time instance t_5 where *PE1* and *PE2* try to access to memory bank *Mem1*.



a. Conflict free natural order Access b. Interleaved order with conflicts

Figure 2: Memory Conflict Problem

This results in memory access conflict problem that increases the memory access latency by two in this example and reduces system throughput in addition to increase the final area (due to the hardware components introduced in the architecture to serialize conflicting memory accesses).

3. PROPOSED APPROACH

We propose to embed the polynomial time memory mapping algorithm we introduced in [18] in order to execute it on-chip at runtime. As opposed to [21] where a Crossbar was considered, we target in this paper a Benes network**.**

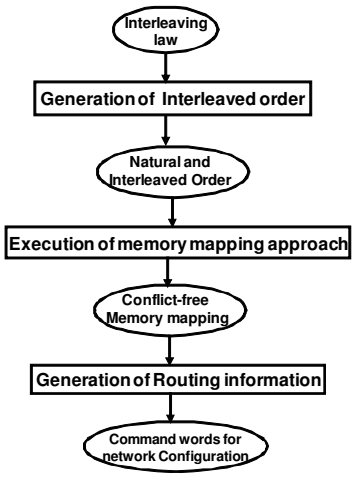


Figure 3: Design Flow

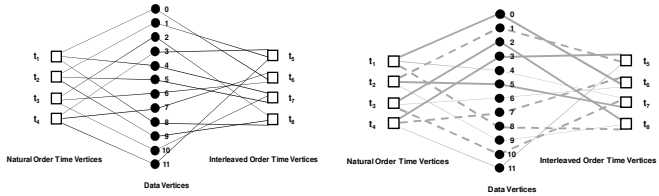
The process of generating routing information for crossbar network is automatic whereas Benes network needs a routing algorithm to generate routing information. For this purpose, we define a simplified routing algorithm for Benes network that can be executed on-chip along with the mapping algorithm. Next three subsections respectively present briefly the basic concepts of our polynomial time memory mapping algorithm, the routing algorithm we propose to embed on-chip and details of the target architecture.

3.1. Memory Mapping Algorithm

The design flow of our approach is shown in Figure 3. First, the interleaved order is generated based on the particular interleaving law along with required parameters like block sizes, level of parallelism and scheduling.

The second step generates the conflict free memory mapping by executing memory mapping approach. The polynomial time algorithm used in this article is briefly described with the help of an example given in Figure 1. The algorithm is based on two steps. In the first step a bipartite graph is constructed based on two data access matrices. Whereas in the second step a polynomial time bipartite edge coloring algorithm is used to find conflict free memory mapping.

In the first step in order to construct a bipartite graph, a tripartite graph $G' = (T_{NAT} \cup T_{INT} \cup L, E)$ is constructed based on natural and interleaved data access matrices (e.g. Figure 1) in which vertex sets T_{NAT} and T_{INT} represent all the time instances used in natural order access and interleaved order access respectively whereas vertex set L represents all the data elements used in the computation. An edge (t_{NAT}, l) is incident to the data vertex l and to the natural order time vertex t_{NAT} if l needs to be processed at t_{NAT} (i.e. data l will be read and next written at time t_{NAT}). Similarly, an edge (t_{INT}, l) is incident to the data vertex l and to the interleaved order time vertex t_{INT} if l needs to be processed at t_{INT}. This tripartite graph Figure 4.a is converted into bipartite graph G by first joining two edges at each data vertex and then removing all the data vertices from the tripartite graph. G is regular with the degree of each time node, $k=P$.



a. Tripartite graph b. Complete Edge Coloring

Figure 4: Bipartite Edge Coloring Algorithm

After constructing bipartite graph, the next step is to apply bipartite edge coloring algorithm to color the edges of that graph in polynomial time. For this purpose, we find an Euler partitioning by taking every other edge to obtain two $(k/2)$-regular subgraphs. In this way the problem is reduced to two $(k/2)$-regular graphs. However, to find euler partitioning, it is necessary that k is even to divide a regular graph into two regular subgaphs of equal degree. So, if K is odd then the algorithm first finds perfect matching Mp in G, assign one color to the edges of Mp and remove Mp from G. The problem is reduced to even $(K-1)$-regular graph. The perfect matching algorithm runs in $O(kD)$ time. The complete edge coloring of G' after attaching data vertices in G is shown in Figure 4.b. In this figure, three colors of the edges, corresponds to three memory banks, are represented with gray bold, gray narrow and gray dotted lines. Further details on this algorithm can be found in [18]. The resultant memory mapping is given as:

Bank A={0,2,3,5}, Bank B ={1,7,8,10}, Bank C = {4,6,9,11}

Afterwards, addressing and network control logic are generated based on this mapping and stored in the memory. So, if we change

the interleaving law then we get a new mapping that is different from the previous one using memory mapping approach. For example, new interleaved order and memory mapping are:
Interleaved order = 2, 7, 10, 8, 9, 6, 1, 5, 11, 3, 4, 0.
Bank A={0, 1, 2, 3} Bank B={4, 5, 6, 11}, Bank C={7,8 9,10}

So, the real disadvantage of executing the memory mapping approaches offline is the requirement of multiple memory elements to support different block lengths within a standard or multiple standards. This results in huge hardware cost that is utilized in storing addressing and control logic to design flexible decoder architecture. In order to reduce hardware cost, either we optimize memory required to store addressing and control logic or run these algorithms on chip. Current, memory mapping approaches are unable to completely remove the use of multiple ROMs to store control information. So, the solution is to run mapping approaches on chip in order to calculate new mapping information as soon as new block length needs to be decoded and updates new addressing and control information in memory. In this article, we explore the possibility of executing the polynomial algorithm on FPGA using different embedded processors to show the advantages of embedding mapping approaches on chip.

3.2. Routing Algorithm

The third step of the design flow for our proposed approach is the execution of the routing algorithm to generate the routing information for the targeted interconnection network. We adopt the routing algorithm given in [20] for Benes network. In this algorithm (P x P) Benes network is viewed as a concatenation of two sub-networks SNI and SN2. The first (log P - 1) stages of a Benes network correspond to SN1, and the remaining log P stages correspond to SN2. SN1 is controlled by a full binary tree of set partitioning functions, called a Complete Residue Partition Tree (*CRPT*) and SN2 is bit controlled.
The control algorithm sets switches one stage at a time, stage by stage. A Complete Residue System modulo m, CRS-(mod m), is a set of m integers which contains exactly one representative of each residue class mod m. Whereas, a Complete Residue Partition, *CRP*, is a partition of a CRS (mod 2^k) into two CRS's(mod $2^{(k-1)}$), k>0. A CRP can be performed in many different ways. As it is the same problem as dividing a pile of 2^k objects consisting of pairs of $2^{(k-1)}$ distinct objects into two piles of $2^{(k-1)}$ distinct objects, there are $2^{\wedge}(2^{(k-1)}-1)$ different ways to perform a CRP. Further details on this algorithm can be found in [20].

3.3. Target Hardware Architecture

The hardware architecture we target to allow for embedding the polynomial time memory mapping algorithms on chip is given in Figure 5. Control unit includes a dedicated processing element (General Purpose Processor GPP, Application Specific Instruction set Processor ASIP or Application Specific Integrated Circuit ASIC) to execute the mapping algorithm. As already discussed in the introduction, network and addressing ROMs are replaced by only two RAMs i.e. a *Network RAM* and an *Addressing RAM*. Online mapping generator executes mapping and routing algorithms. Afterwards, it updates these RAMs with command and addressing words for each time block length changes. A fully connected non-blocking network can be used to support multiple block lengths as proposed in [21]. Crossbar is a non-blocking network in which connection of a processing element to a memory bank does not block the connection of any other processor to any other memory bank. It is feasible for online approaches to use crossbar network due to its high speed as they can be configured automatically. But, their use is normally limited to low level of parallelism, due to high complexity and cost. Hardware constraints such as the number of available pins and the available wiring area limits the number of physical connections of a switch. These issues prevent the use of crossbar networks for large network sizes [19]. Size of addressing_RAM = $B^* T * \log_2(T/2)$, where size of each word is $B^* \log_2(T/2)$ bits. For a Crossbar network, the size of network_RAM= $T^* (P * \log_2 P)$, while for a Benes network, the size of network_RAM = $T^*(P/2*((2*\log_2 P)-1))$. For Benes network, the size of bus from network RAM to network is $P/2*((2*\log_2 P)-1)$ bits and the size of each bus from addressing RAM to banks is $B^* \log_2(T/2)$ bits.

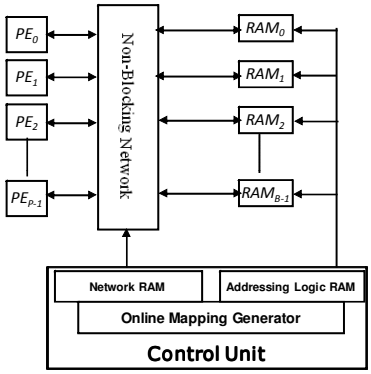


Figure 5 : Parallel decoder architecture to embed memory mapping algorithms on chip

As a consequence, different alternative topologies have been proposed in which data have to cross several switches before reaching the destination. Therefore, a multistage interconnection network such as Benes network suitable for very large systems is targeted in this article. Benes network is also a non-blocking network with complexity much lesser than Crossbar networks. However, unlike crossbar network, the routing information in Benes network is not automatically generated. Therefore, we propose to embed a routing algorithm presented in section 3.B along with the mapping algorithm to generate command words for Benes network. The complexity of the routing algorithm is $T^*P^*(\log P)$ for a complete block length which increases with the size of the block length for same P. However, for a given block length, the complexity remains almost the same since the number of accesses (T) decrease as the parallelism level (P) increases.

4. RESULTS

In order to show the advantage of the proposed approach, different experiments have been performed using different embedded processors to measure the runtime performances. The experiments consider three parameters: block size, parallelism and processor type. In this section, the amount of memory required to store command words both in case of on chip and off chip execution of memory mapping approaches is also compared. For experimental purpose, one hard processor PowerPC embedded in Xilinx FPGA and one soft processor NIOS-II used in Altera FPGAs are considered to execute our approach along with [11] and [13]. The execution times for [11] [13] and our approach are measured for different processors. Moreover, HSPA interleaver used in 3GPP-WCDMA [16] is implemented on parallel architecture. To design parallel conflict free architecture for this interleaver, memory mapping approaches are required to generate commands and addressing words to support all the block sizes with different parallelisms.
For the experiments, the first processor we considered is PowerPC which is a hard processor embedded in Xilinx Virtex-5 ML507 board. Processor clock frequency of 400MHz and System clock

frequency of 100MHz was used to perform experiments. The second processor we considered in our experiments is NIOS II. NIOS II is a soft processor used in Altera FPGAs. NIOS II has been implemented on Cyclone-III NIOS II Embedded Evolution Kit with Processor clock frequency of 195MHz and System clock frequency of 50MHz. Normalized time values are used to measure the impact of architecture of embedded processors on execution time. PowerPC execution time is used as a reference for normalized time and execution times of NIOS II is normalized with respect to the PowerPC clock frequencies.

The normalized times to execute our approach and [11][13] on embedded processors for different L with P = 4,8,16 and 32 are studied. Moreover, Crossbar network is used for [11] [13] whereas Benes network is considered to implement our approach. The normalized times to execute [11][13] and our approach for different L with P = 32 are shown in Figure 6 for NIOS II and PowerPC. From processor perspective, PowerPC executes the mapping algorithm in the least time as compared to NIOS II. From this figure, it is evident that significant reduction in execution time for our proposed approach is achieved as compared to [11] and [13] for all block lengths. For L = 5120 with P = 32, using PowerPC the execution time in case of [11] and [13] is about 2 hours where as by using our proposed approach the execution time is reduced significantly to only 127ms. Furthermore, the results for our approach also include the delay introduced by routing information generation process whereas the routing information is automatically generated as crossbar is considered for [11] and [13]. An additional delay of 0.2ms is required in order to generate routing information for complete block length of size L=256 with P=*32*, which increases to 4ms for L=5120 with same P. However, for the same block length the delay in command word generation remains almost the same for different level of parallelism as explained in previous section.

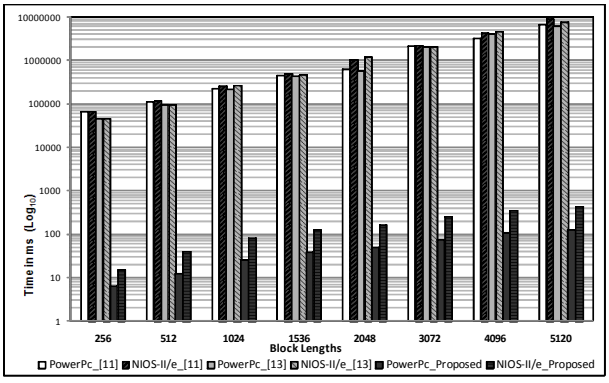


Figure 6 : Normalized Run time Values of [11] [13] and our approach for different embedded processors with PE = 32

To measure the performance of our approach with respect to [11] [13] for different types of parallelism, we performed experiments with P=4, 8, 16 and 32 for L=5120 as shown in Figure 7. From these experiments, significant reduction in execution time for proposed approach can be seen as compared to [11] and [13]. Furthermore, the execution time for our approach has no significant change with increase in parallelism whereas execution time for [11] [13] increases almost 25 times with the increase in parallelism from P = 4 to P = 32.

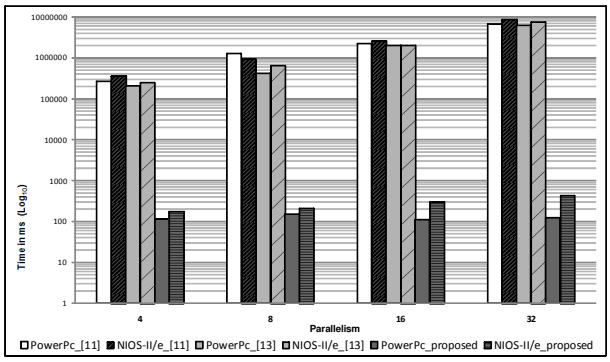


Figure 7 : Normalized Run time Values of [11] [13] and our approach for different types of parallelism using block length 5120

From memory perspective, our approach requires the same memory for different parallelism to support multiple block lengths. However, for offline memory mapping approaches, high memory cost is required to support several block lengths. For P = 32, size of memory required in case of off-chip approach to store command words is 64-Mbits to implement all the block sizes used in 3GPP-WCDMA. Thanks to the extensive reuse of RAM only 128Kbits of memory is required in case of on-chip execution of mapping algorithms. Figure 8 shows the comparison between the memory required to store command words with P =4, 8, 16 and 32. Same memory is reused to store command words as soon as the parallelism is changed in order to support this parallelism in case of proposed approach whereas off-chip approaches require additional memory to store new set of command words with each type of parallelism.

The significant reduction in execution time and area obtained using our approach encourages embedding memory mapping and routing algorithm in future telecommunication devices.

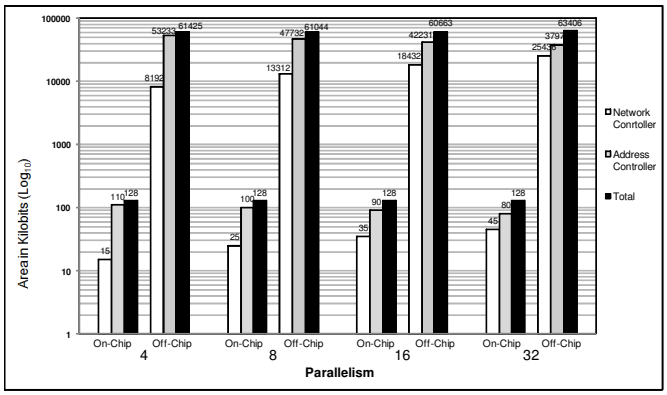


Figure 8 : Area Comparison (Log Scale) using on-chip and off-chip approaches for different types of parallelism

5. CONCLUSION

In this article, we proposed to embed polynomial time memory mapping approach and routing algorithm on-chip to solve memory conflict problem in parallel hardware decoders based on Benes network. Different experiments have been performed by using existing memory mapping approaches executed on several embedded processors. Results shown that the proposed approach allows to greatly improve timing performances and to reduce memory footprint. Future perspective of this work is to further improve the execution time by using ASIP or hardware accelerators to target real time flexible decoder architectures.

REFERENCES

[1] C.Berrou, A.Glavieux, and P.Thitimajshima, "Near-Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Commun.*, vol.2, pp.1064–1070, 1993.

[2] R. G. Gallager, "Low-Density Parity-Check Codes*", Cambridge, MA: MIT Press*, 1963.

[3] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, M. Jezequel, "Designing good permutations for turbo codes: towards a single model," in *Proc. of ICC* 2004, vol. 1, June 2004, pp. 341-345

[4] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.

[5] M. Thul, N. Wehn, and L. Rao, "Enabling high-speed turbo decoding through concurrent interleaving," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1, 2002, pp. 897–900.

[6] M. I. Thul, F. Gilbert. and N. Wehn. "Optimized Concurrent Interleaving for High-speed Turbo-Decoding". *In Proc. ICECS 2002, Dubrovnik, Croatia, Sept. 2002.*

[7] M. Thul, F. Gilbert, and N. Wehn, "Concurrent interleaving architectures for high-throughput channel coding," *in Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2003, pp. 613–616 vol.2.*

[8] H. Moussa, O. Muller, A. Baghdadi, and M. Jezequel, "Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding," in *Proc. of the conference on Design, Automation and Test in Europe*, pp. 654-659, April 2007.

[9] H. Moussa, A. Baghdadi, and M. Jezequel, "Binary de Bruijn interconnection network for a flexible LDPC/turbo decoder," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2008, pp. 97–100.

[10] P. Keyngnaert, B. Demoen, B. De Sutter, B. De Sus,and K. De Bosschere. "Conflict Graph Based Allocation of Static Objects to Memory Banks" *Informal proceedings of the First workshop on Semantic, Program Analysis, and Computing Environments, pages 131–142*, September 2001.

[11] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures", *IEEE Trans.Inf.Theory*, *vol. 50, no.9, pp.2002-2009*, Sep. 2004.

[12] Jing-ling, "Parallel Interleavers Through Optimized Memory Address Remapping" *IEEE Trans. VLSI Systems* vol. 18, no.6, pp.978-987, June. 2010.

[13] C. Chavet, P. Coussy, P. Urard and E. Martin, "Static Address Generation Easing: a Design Methodology for Parallel Interleaver Architecture". *In proceeding ICASSP 2010.*

[14] J.L. Gross, J.Yellen, "Handbook of Graph Theory", *353, CRC Press*. 2003.

[15] "Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access; Multiplexing and Channel Coding (Release 8)", *3GPP Std. TS 36.212*, Dec. 2008.

[16] 3GPP, "Technical specification group radio access network; multiplexing and channel coding (FDD)" *(25.212 V5.9.0).June 2004.*

[17] *IEEE P802.16e, Part 16.* "Air Interface for Fixed and Mobile Broadband Wireless Access Systems," Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, and Corrigendum 1, Feb. 2006.

[18] A.H. Sani, C. Chavet and P. Coussy, "A First Step Toward On-Chip Memory Mapping for Parallel Turbo and LDPC Decoders: A Polynomial Time Mapping Algorithm", *IEEE Transactions on Signal Processing, vol. 61, issue: 16, p.4127 - 4140, 2013.*

[19] Jose Duato, Sudhakar Yalamanchili and Lionel Ni. "Interconnection Networks an Engineering Approach", *Morgan Kaufman Publishers, 2003,p.20-30*

[20] K. Y. Lee, "A new Benes network control algorithm," *IEEE Trans. Comput., vol. C-36, no. 6, pp. 768–772, June 1987*

[21] S. Ur Rehman, A. Sani, P. Coussy and C. Chavet, "On-Chip Implementation Of Memory Mapping Algorithm To Support Flexible Decoder Architecture"*, In Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, May, 2013.*

[22] C. Chavet and P. Coussy, "A memory Mapping Approach for Parallel Interleaver design with multiples read and write accesses", *In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2010, page 3168-3171, Paris, France, june 2010*

[23] A. Briki, C. Chavet and P. Coussy, "A Conflict-Free Memory Mapping Approach To Design Parallel Hardware Interleaver Architectures With Optimized Network And Controller", *In Proceedings of IEEE Workshop on Signal Processing Systems (SiPS), page XX-YY, Taipei : Taiwan, Province De Chine, oct. 201*

[24] *G. Masera, "*VLSI for turbo codes," in Turbo Code Applications: AJourney From a Paper to Realization. *Berlin, Germany: Springer-Verlag, 2005, ch. 14.*