ON-CHIP IMPLEMENTATION OF MEMORY MAPPING ALGORITHM TO SUPPORT FLEXIBLE DECODER ARCHITECTURE

Saeed-ur-REHMAN, Awais SANI, Philippe COUSSY, Cyrille CHAVET

Lab-STICC, Université de Bretagne-Sud, Lorient

***Abstract*- Parallel hardware architectures are used to design turbo-like iterative decoders to meet the requirement of high data rate applications. However, parallel architectures suffer from memory conflict problem due to interleaving law used in turbo-like codes. To solve conflict problem, different memory mapping approaches have been developed. These methods automatically generate a set of control words stored in ROM to drive the architecture. These approaches are used off-chip by the designer (i.e. prior the decoder implementation) to generate different set of control words i.e. one set for each block length used in the target telecommunication standard. This requires multiple ROMs to store mapping information for multiple block lengths and results in huge hardware cost. In this article, we propose to embed memory mapping algorithms on-chip. Hence, each time word-length changes, memory mapping algorithm is executed. Command words are thus generated at runtime and stored in a RAM. This is a first attempt to embed mapping algorithms on chip and experimental results show that a significant amount of memory can be saved by using on-chip execution of mapping algorithms. Results also highlight that improvement in design and implementation of mapping algorithms are still needed to embed mapping algorithms on-chip to implement flexible decoder architectures.**

1. INTRODUCTION

The development of broadband devices such as smart-phones or notebooks requires wireless telecommunications standards that support high data rate applications. To cope with this high data throughput requirement, current telecommunication standards use OFDM, MIMO and advance error correction techniques to reliably transfer data on different wireless networks. Moreover, each standard includes different block lengths in order to support different applications running on these broadband devices.

Turbo-like codes [1] [2] are used in current telecommunication standards [15]-[17] for channel coding, equalization, demodulation and synchronization due to their excellent error correction properties. However, iterative algorithms that are used to decode these codes result in huge latency. Parallel decoder architectures are thus employed at receiver to speed up the decoding and to support application timing requirements. In parallel architectures, several processing elements *PEs* (decoders) are concurrently used to decode the received information. To increase memory bandwidth, several memory banks *BKs* are connected with these *PEs* through interconnection network. The function of the network is to exchange the data between PEs and *BKs* according to predefined interleaving or permutation law. Interleaving laws are parameterized by block lengths. Typical parallel decoder architecture is shown in Figure 1.

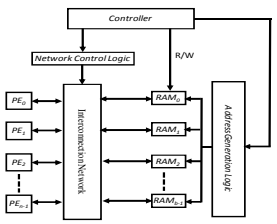


Figure 1 : Parallel decoder architecture

Unfortunately, parallel decoder architectures suffer from memory access conflict problem which results in increased latency and hardware cost. To manage this problem, different conflict free interleaving laws are used in current telecommunications standards. For example, 3GPP-LTE [15] uses Quadratic Permutation Polynomial (QPP) [4] interleaver whereas WiMAX [17] uses ARP [3] interleaver to permute the data. These interleavers often simplify the parallel decoder architecture Indeed, these interleavers are conflict free for particular types or degrees of parallelism used in turbo decoding. For example, QPP interlever is conflict free for SISO Decoder level and Radix-4 level Parallelism whereas ARP supports only SISO Decoder level Parallelism. Hence, to fulfill high throughput requirement of current and future standards, decoding architecture must support all types of parallelisms that can be employed in turbo decoding. Currently two classes of approaches exist to tackle memory conflict problem when designing parallel decoder architectures:

- *Run time approaches* use extra memory elements and control logic in the communication network in order to remove conflicts [5][6][7][8][9].
- *Design time approaches* find a memory mapping to provide conflict free concurrent access to all the memory banks [11][12][13]

In first family of approaches, collision problem is tackled either through the addition of extra memory elements and/or complex interconnection network. In [5][6][7], dedicated interconnection network called LLR distributor is designed to tackle conflict problem for turbo codes. Different structures such as *Tree Interleaver Bottleneck Breaker* (TIBB), *Ring Interleaver Bottleneck Breaker* (RIBB) and *General Interleaver Bottleneck Breaker* (GIBB) are proposed to handle conflict and to connect LLR distributor to *PEs* and *BKs*. To increase the scalability and to meet higher throughput requirement on flexible communication network, two heterogeneous multistage networks, butterfly and Benes, are investigated in [8]. These networks exhibit huge scalability and very simple packet routing algorithms but requires pre-computation of routing paths and packet scheduling which is not a feasible solution for implementing different standards on the flexible decoder architecture. In [9], Binary de Bruijn interconnection network is presented to provide scalability and allow any permutation to be routed efficiently. Due to its path diversity, communication conflicts are managed 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 and cost due to increased buffer control architecture necessary to manage conflicting packets. Also, delay introduced due to conflict management mechanisms degrades the maximum throughput and makes these approaches inefficient for high data rate and low power architectures.

In second kind of approach, different algorithms are proposed to provide conflict free concurrent accesses to all processing elements. For that, pre-processing is realized to determine the memory locations for each data element used in the computation. The most common approach is to prepare conflict graph in which a node represents a data and two nodes are connected if and only if the associated data are accessed at the same time. Node coloring approach can then be used to solve the mapping problem: each color corresponds to one memory bank. However, node coloring is NP-complete problem as shown in [10]. In [11][12], the authors proposed a heuristics to find conflict free memory mapping in turbo decoder. Contrary to the literature belief, the authors have proven that for every code, conflict free memory mapping always

exists to tackle collision problem. However, the proposed approach is based on a simulated-annealing algorithm, so the user cannot predict when the algorithm will end. In [13], another heuristic is proposed which finds conflict free memory mapping while optimizing the storage element and interconnection network. However, all these heuristics fail to remove the computational complexity of the problem. The benefit of these approaches is that decoder implementation does not need any specific network and extra memory elements to support particular interleaving law. Rather any network which supports all the permutation patterns between inputs and outputs can be used to implement any interleaving law. However, the approach requires preprocessing to map data in different memory banks for different block lengths and parallelism degree.

In this article, to the best of our knowledge, we have presented the first attempt to embed memory mapping algorithms on-chip in order to execute them at runtime to solve conflict problem. The purpose of this article is to show that both runtime and design time approaches could be merged in the future to design flexible decoder.

The rest of the paper is organized as follows. Section 2 explains how memory mapping approaches are used to solve memory conflict problem and what are the inconveniencies in executing design time approaches offchip. Section 3 describes the architecture that is used in this article to execute memory mapping algorithms. Section 4 uses different embedded processors to measure the computational complexity of different memory mapping approaches. Finally, in section 5, we conclude our paper.

2. MEMORY MAPPING APPROACHES

As explained in previous section, memory mapping approaches are used to solve memory access conflict problem when designing parallel decoder architectures. However, before presenting detailed mechanism of how these approaches work, we briefly explains memory conflict problem. The problem can best be explained through simple example of turbo codes. In this example, natural and interleaved orders are:

Natural order = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
Interleaved order = 0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11

For parallel processing, this codeword is divided into four windows in both natural and interleaved order and arranged in data access matrices of Figure 2. In this figure, each row (or window) is processed by one processing elements whereas data in each column (or time instance) are need to be accessed concurrently in parallel.

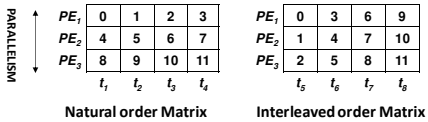
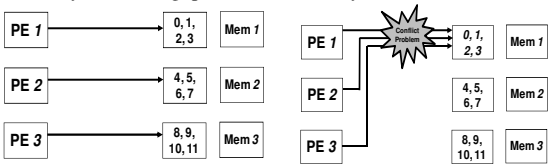


Figure 2: Data Access Matrices

To increase memory bandwidth, three memory banks are used so that each processing element can concurrently get data elements in parallel. Data elements 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 as shown in Figure 3.a. However, using this memory mapping, all processing elements always access the same memory bank at each time instance in interleaved order as shown in Figure 3.b. This results in memory conflict problem and increases latency (by three in this example) in data fetching from memory due to the presence of conflict management mechanism in communication network. Furthermore, this problem reduces system throughput and increases system cost.



(a) Conflict free natural order Access (b) Conflict full interleaved order Access

Figure 3: Memory Conflict Problem in Parallel Turbo Decoder

To solve this memory conflict problem, mapping approaches uses different heuristics to find conflict free memory mapping in which all the *PEs* always access the different memory banks at each time instance. These heuristics first transform natural and interleaved orders in matrix format as shown in Figure 2. Afterwards, heuristics use additional matrices in which they store current mapping information. This mapping information is modified after each iteration of the mapping algorithm until conflict free memory mapping is obtained. To explain execution flow of mapping approaches, we use a simple approach in which two additional matrices called *Mapping Matrices* are used to store mapping information. These matrices (MAP_{Nat}, MAP_{Int}) have the same order as the natural or interleaved order matrices as shown in Figure 4. To find conflict free memory mapping, each column of the mapping matrices should contain different memory banks and each data must be mapped in one and only one memory bank.

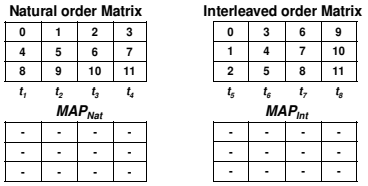


Figure 4: Matrices used in SAGE

Algorithm initializes by assigning memory banks to the first column of MAP_{Nat}. Next, algorithm updates the entries corresponding to the data in MAP_{Int} with this mapping information. After that, at each iteration, the algorithms select the most constraint column (column which has minimum number of filled entries), fills that column with mapping information respecting the constraints and update that mapping information into other matrix until all the columns of the mapping matrices are filled with mapping information. If memory banks are represented by *A, B, C* then mapping matrices at the end of the algorithm is shown in Figure 5.

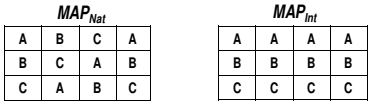


Figure 5: Final Mapping Matrices

The resultant memory mapping is,

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

Based on this mapping, addressing and network control logic are generated. Currently, memory mapping algorithms are using memory to store addressing and control logic. So, if we change the interleaving law then new interleaved order is obtained and using memory mapping approaches, we get a new mapping that is different from the previous one. For example, new interleaved order and memory mapping are:

I*nterleaved 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 these approaches is the requirement of multiple memory elements to support different block lengths within a standard or multiple standards as shown in Figure 6. 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 optimize memory necessary to store control information. So, the only 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. However, computational complexity of current memory mapping approaches makes them difficult to be embedded on chip. In this article, we explore the possibility of executing these algorithms on FPGA using different embedded processors to show the advantages and disadvantages of embedding mapping approaches on chip.

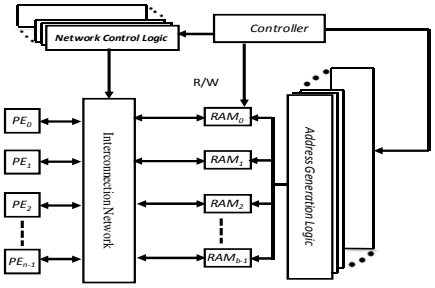


Figure 6 : Parallel decoder architecture supporting multiple block lengths

3. PROPOSED ARCHITECTURE

We proposed a dedicated hardware architecture (see Figure 7) to allow for embedding memory mapping algorithms on chip. 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. Multiple network and addressing ROMs (as shown in Figure 6) are replaced by a two RAM i.e. *Network RAM* and *addressing RAM*. Control Unit executes mapping algorithm and updates these RAMs each time block length changes.

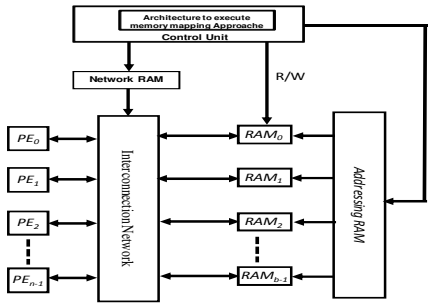


Figure 7 : Parallel decoder architecture to embed memory mapping algorithms on chip

Sizes of *Network* and *addressing* RAMs depend on maximum block length and on the parallelism supported by decoding architecture. To determine size of different components of the architecture to support complete telecommunication standard, following parameters are considered:

N = Total Number of processing elements
B = Maximum number of memory banks
T = Maximum number of access to the memory
R = Maximum number of data in each bank

For example, if we consider four processing elements (N = 4) implemented by using forward backward, butterfly, radix-4 and butterfly with radix-4 parallelisms (B = 16). However, memory mapping approaches would be able to solve mapping problem for any type of parallelism used in turbo decoding. Size of addressing RAM = $B * T * \lceil \log_2(R) \rceil$ where size of each word is $B*\lceil \log_2(R) \rceil$ bits. Similarly, if the network is Benes then the size of network RAM = T * (N/2*((2*$\log_2$N)-1)). Also the size of bus from network RAM to network is N/2*((2*$\log_2$N)-1) bits and the size of each bus from addressing RAM to bank is $B*\lceil \log_2(R) \rceil$ bits.

Memory mapping architecture starts by receiving information about the block length decoder wants to decode. The architecture first calculates interleaved order related to this block length. Based on this information, memory mapping algorithm is executed to calculate conflict free memory mapping and updates each addressing and control memory to start decoding new block.

4. EXPERIMENTS

In this section, different experiments have been performed using different embedded processors to measure the computational complexity of memory mapping approaches based on three aspects: block size, parallelism and processor. Moreover, the memory required to store command words both in case of on chip and off chip execution of memory mapping approaches is also compared in this section. In this regard, two soft processors (micro blaze and NIOS II) and one hard processor (PowerPC) embedded in Xilinx and Altera FPGAs are used to execute [11][13]. For simplicity, we divide these experiments into three sets based on the embedded processor that executes these algorithms. For each processor, execution time to compute [11][13] is calculated and compared in this section. In this article, we implement HSPA interleaver used in 3GPP-WCDMA [16] on parallel architecture. This interleaver is not conflict free to support parallel implementation of turbo decoder and memory mapping approaches are required to find mapping for wide range of block sizes. The architecture is designed to support all f the block sizes.

The first processor we considered is Microblaze which is a soft processor used in Xilinx FPGAs. This embedded processor has been implemented on Virtex-5 ML507 Evaluation Platform with Processor clock frequency of 125MHz and System Clock frequency of 100 MHz. Due to limited resources of FPGA, we use off-chip memory to execute our mapping algorithms. 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. Off-chip memory was used to execute mapping algorithms due to limited on-chip memory of FPGA. The third processor we considered in our experiments 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. On-chip memory was used in this set of experiments to execute mapping algorithms. To measure the impact of architecture of embedded processors on execution time, normalized time values are used. For normalized time, PowerPC execution time is used as a reference and execution times of Microblaze and NIOS II are normalized with respect to the PowerPC clock frequencies.

The normalized times to execute [11][13] on Microblaze, NIOS II and PowerPC for different block lengths and PE = 4 are shown in Figure 8. From this figure, it is clear that execution time of [11][13] increases with the increase of block lengths. Moreover, [13] is always able to finds memory mapping in less time than [11]. From processor prospective, Microblaze takes the highest time to find memory mapping whereas NIOS II executes the mapping algorithm in the least time.

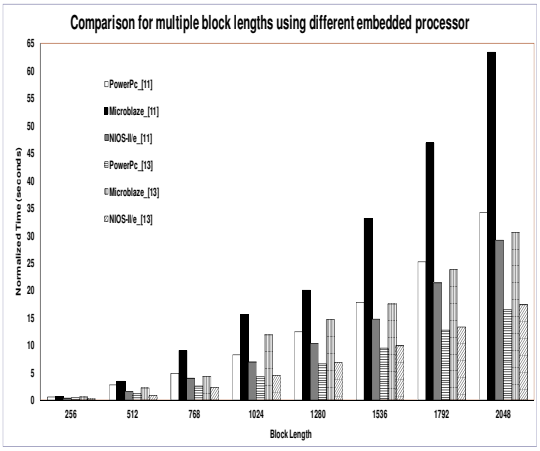


Figure 8 : Normalized Run time Values of [11] and [13] for different embedded processors using forward backward Parallelism

We measured the performance of [11][13] for different types of parallelism used in turbo decoding. Figure 9 gives the normalized execution time of different processors for forward backward, butterfly and butterfly with radix-4 parallelism. From this comparison, it is clear that execution time increases almost 7 times while moving from forward backward parallelism to butterfly with radix-4 parallelism.

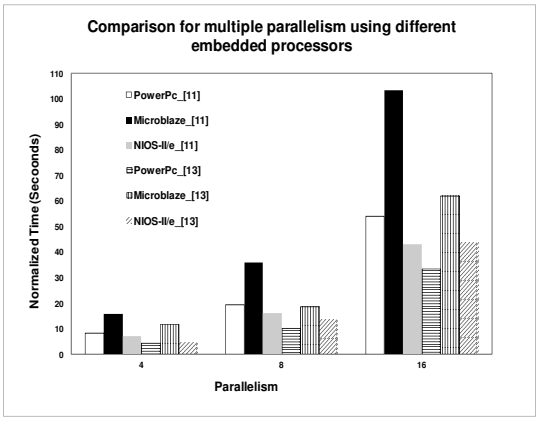


Figure 9 : Normalized Run time Values of [11] and [13] for different types of parallelism using block length 1024

From architecture perspectives, cost of our architecture always remains constant for different parallelisms supporting several block lengths for each processor. However, the cost of ROM memory required to store command words in case of off-chip execution of mapping approaches is extremely high to be implemented on practical systems. To implement all the block sizes used in 3GPP-WCDMA, off-chip approach requires 62Mbits memory to store command words whereas thanks to the extensive reuse of RAM only 128Kbits of memory is required in case of on-chip execution of mapping algorithms using forward backward (N=4) parallelism. Figure 10 shows the comparison between the memory required to store command words in case of forward backward (N=4), butterfly (N=8) and butterfly with radix-4 (N=16) parallelisms. Moreover, by changing the parallelism, same memory is used to store command words to support this parallelism in case of on-chip execution of mapping approaches. However, off-chip approach requires extra memory to store new set of command words with each type of parallelism. In Figure 10 the comparison of the memory required in decoder architecture is given.

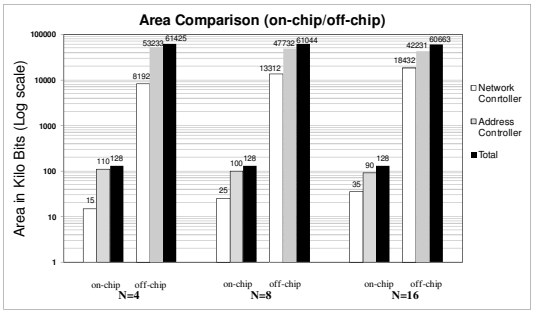


Figure 10 : Area Comparison (Log Scale) using on-chip and off-chip approaches for different types of parallelism

From these experiments, it is clear that significant reduction in architectural cost can be obtained by executing mapping algorithms on-chip. Moreover, execution times of these algorithms are affected by size of block lengths, parallelisms, architecture of embedded processor and mapping algorithm. This time can thus be reduced by using ASIP or non programmable hardware accelerator architecture to execute mapping algorithms. In addition, improvement in algorithmic development would also significantly improve the timing performances.

5. CONCLUSION

In this article, we proposed the first attempt to embed memory mapping approaches on-chip to solve memory conflict problem in parallel hardware decoders. Dedicated architecture composed of an embedded processor and RAM memory banks to store command words has been proposed. Experiments have been done by using different memory mapping approaches using several embedded processors to compare the computational and architectural complexity of executing these approaches on-chip and off-chip. Experimental results shows that computational complexity of memory mapping approaches depends on mapping algorithms, embedded processor architectures, block lengths and parallelisms. From architectural prospective, cost of the system to implement mapping algorithms off-chip is 480 times greater than the cost to implement these algorithm on-chip. This motivates to implement mapping algorithm on-chip to support multiple standards on a single chip. However, more efficient algorithms are required to find conflict free memory mapping on-chip and to design flexible decoder that supports multiple standards and parallelisms. Moreover, to improve the computational time and to reduce area of the architecture that executes mapping algorithms, ASIP or hardware accelerator need to be used. This will enable to implement flexible decoder architecture with reduced architectural and computational complexity.

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.