# FULL RECONFIGURABLE INTERLEAVER ARCHITECTURE FOR HIGH-PERFORMANCE SDR APPLICATIONS

Renaud Pacalet (Telecom-Paristech, LabSoC, Sophia-Antipolis, France; renaud.pacalet@telecom-paristech.fr);
Jair Gonzalez-Pina (Telecom-Paristech, LabSoC, Sophia-Antipolis, France; jair@telecom-paristech.fr).

## ABSTRACT

This paper presents an interleaver / deinterleaver architecture that meets all the requirements for complex SDR applications, basically, it offers enough flexibility to implement about any interleaving method. This architecture allows to run several interleaving processes concurrently, either with the same method or with different ones, enabling high performance multi-radio applications with the same hardware. This architecture integrates the interleaving altogether with the rate matching and the frame equalization – two closely related processing –, improving the overall system's performance.

While the reconfigurable interleaver can be shared among several steps of the communications process, it would be very inefficient to use it for the internal interleaving of turbo coding or decoding. It would require a large communication bandwidth between the interleaver and the channel coder / decoder. This paper also discusses the implementation of two highly optimized interleaver architectures for the specific requirements of turbo coding modules of LTE and UMTS standards.

## 1. INTRODUCTION

It is widely accepted that future wireless communications devices will be based in Software Defined Radio Platforms (6),(8), *Radio Platforms* composed of shared hardware and software modules, all of them being configurable enough to support different standards.

This paper presents three interleaver architectures: a reconfigurable one that is capable of implementing any interleaving method, and two other which are highly optimized for the internal interleaving of the channel turbo coding / decoding modules of LTE and UMTS standards.

As the interleaving process has a direct impact on the performance of the system, many papers have been written that evaluate performance of different interleaving methods (5), (9), (10), (12). In contrast, few papers are focused on interleaver architectures for SDR applications. (4), (11) and (14) are examples of dual-mode interleaver architectures, which are sufficient for systems targeting standards with similar interleaving methods, but not for real SDR and CR applications.

Our reconfigurable interleaver / deinterleaver module also implements rate-matching and frame-equalization within the same hardware. To our knowledge, no other previous architecture proposed this, which in fact increases not only the performance of the interleaver itself, but also the overall system performance.

The paper is organized as follows: Section II describes the functionality and architecture of our full reconfigurable interleaver. Section III presents the optimized core of LTE's inner interleaver. The section IV presents the optimized core for UMTS inner interleaver. The cores are compared to other implementations in section V. Section VI concludes.

## 2. FULL RECONFIGURABLE INTERLEAVER

As illustrated by figure 1, our fully reconfigurable interleaving core uses three internal multi-port memories: an input buffer $X$, an output buffer $Y$ and a permutation buffer $P$, which holds the table that defines the permutation function $\pi(i)$. The basic operation mode is straightforward: entries of the permutation table are pointers to input buffer entries; during execution the permutation table stored in $P$ is parsed, one entry at a time, and the pointed input sample $X[P[i]]$ is copied in the output buffer: $Y[i] = X[P[i]]$.

In the presented implementation, the input and output buffers are 8 bits wide with length of $2^{16} = 65536$. The $P$ buffer is 16 bits wide, allowing to address the whole 64k samples from input buffer $X$. This memory configuration exemplifies the requirements of a worst case scenario, and can be changed according to the particular requirements of a given application, without altering the functionality of the core.

For systems targeting only one or two communications standards, and assuming their interleaving functions are structured ones, it is likely that more efficient architectures exist, both in terms of silicon area and power consumption: while input and output buffers can usually not be avoided in

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement SACRA n° 249060.

any implementation, the permutation buffer can sometimes be replaced by a more cost effective address generator. But as we target complex SDR with a full range of flexibility and as we want to adapt even to not yet defined interleaving methods, it was chosen using a permutation buffer as the one option that offers the level of flexibility and performance that is required. In order to mitigate the overhead, we take benefit of the offered flexibility and implement extra features as rate matching, frame equalization, handling of soft bits and hard bits, and mapping in multi-carrier waveforms.

An important characteristic of our architecture is its capability to handle concurrent instances of the same or different standards (for multi-radio applications). Switching from one standard to another is just a matter of loading a new permutation table in memory or, when several tables fit in the $P$ buffer, pointing to a different table(see figure 1 ). In other approaches based on address generators this can be achieved by duplicating the hardware and / or adding different generators for different standards (assuming all of them are known at design time)...up to the point where the number of generators becomes intractable or where their cumulated cost exceeds the cost of our table-based solution. Our figures account already for this capability.

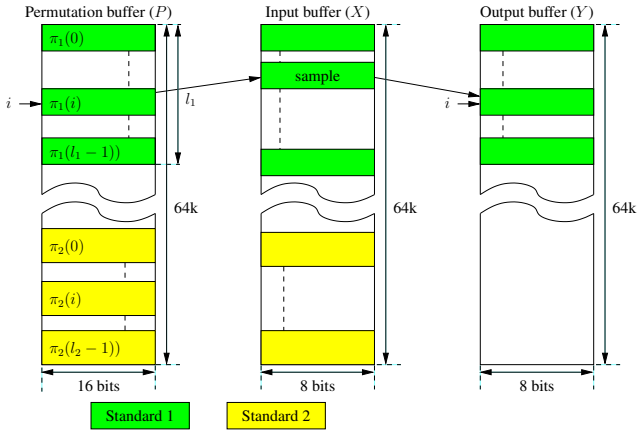


**Fig. 1**. Memory based interleaver architecture

## 2.1. Advanced Interleaving modes

As already mentioned, our reconfigurable architecture offers several extra features, configured through a set of parameters. It is important to note that these extra features do not require extra computation time, they are performed while interleaving and with the same hardware.

### *2.1.1. Hard-bits and soft-bits modes*

The module handles soft bit samples of 1 to 8 bits, stored on a one byte per sample basis, as shown on figure 2, example (a). The unused bits on a byte are ignored in the input buffer and forced to zero in the output buffer.

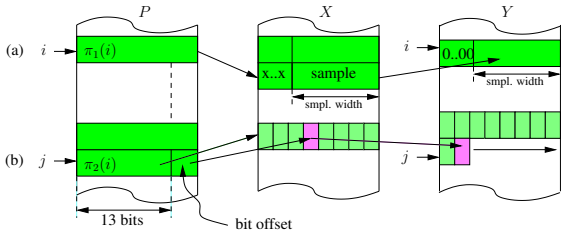


**Fig. 2**. Softbits (a) and Hardbits (b) operating mode

In the hard-bits mode, each byte of the input or output buffer holds 8 different one-bit samples. The meaning of the entries in the permutation table $P$ changes: the 13 MSBs (Most Significant Bits) are the byte offset in the input buffer $X$ and the 3 LSBs (Least Significant Bits) are the bit offset of the input sample in the selected byte, as depicted in figure 2, example (b), where the input and output data are accessed bitwise. Hard bits are frequently used on transmission and soft bits on reception.

### *2.1.2. Rate-matching mode*

The rate matching process is carried out so that the size of a block of samples matches the size of radio frames. It will either repeat bits to increase the rate or puncture bits to decrease the rate.

At transmission the rate-matching is specified in the permutation table, either by repeating entries in the $P$ Buffer, or by omitting them, accessing several times the same sample from the $X$ buffer, or simply not accessing it at all, according to the rate-matching specification.

At reception, repeated samples will be handled differently from the others. When rate-matching mode is enabled, the MSB of each $P$ entry is considered as a repeat flag. The 15 LSBs form the pointer to the input sample, reducing the range to 32768 instead of 65536. The repeat flag defines a *repeat sequences* in the $P$ buffer (figure 3). The pointers in a repeat sequence point to the different copies of the same repeated input sample. The different copies can have different values because of transmission errors. Depending on the selected rate-matching mode, the corresponding, single, output sample will be the average of the copies or the last one. If rate-matching mode is *average*, a repeat sequence is at most 8 entries long. Unless the last of a repeat sequence, an entry

with the repeat flag unset points to a regular, non-repeated, sample.

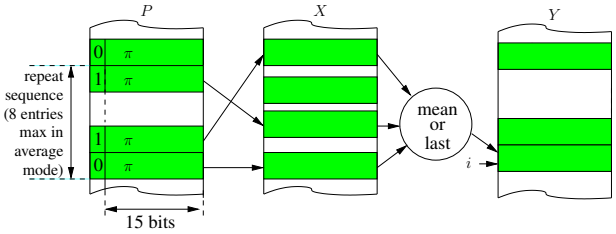


**Fig. 3**. Rate-matching operating mode

### *2.1.3. Frame equalization mode*

Frame equalization consists in adding stuffing bits to the data stream, either `zeros` or `ones`, with the purpose of ensuring that the output can be segmented into equal size segments to be transmitted in a TTI (Transmission Time Interval).

When the frame equalization option is enabled, two special values of permutation entry are defined: `ForceZero` and `ForceOne`. When a `ForceZero` / `ForceOne` entry is encountered in the $P$ buffer, the interleaver forces a `zero` / `one` in the next ($i^{th}$) position of the $Y$ buffer. The `ForceZero` and `ForceOne` special values can be specified as an operation parameter, and should of course be different. Figure 4 illustrates the frame equalization.

### *2.1.4. Very large permutation functions*

Although the proposed buffer sizes are large enough for most today's standards, it is possible that future interleaving methods require larger permutation tables. Still, these large permutations can be implemented in this architecture in several passes.

When interleaving in several passes, it may be required to skip output entries that were updated in previous passes or will be updated in subsequent passes. In order to allow this, a third special value is defined for the permutation entries: the `SkipValue`. When the `SkipValue` entry is encountered and the skip mode is enabled, the pointer to the output buffer is incremented without writing any output sample, thus preserving the value of the skipped sample. The `SkipValue` entry can also be defined as an operating parameter and should be different from `ForceZero` and `ForceOne` if both skip and frame equalization modes are enabled.

## 2.2. Implementation

The interleaver was implemented as a pipeline with stationary control as depicted in figure 5. The pipeline is divided in nine

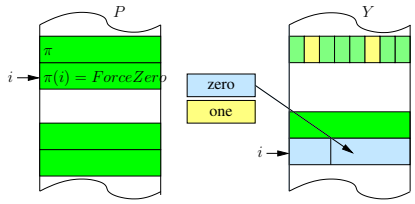


**Fig. 4**. Frame equalization mode

stages. Flushing or stalling it is never necessary. It runs at full speed (one permutation entry per cycle) in all configurations. Three out of the nine stages are dedicated to memory access, the others are used to generate read and write addresses for the three buffers, and to make calculations for Rate-matching and Frame-equalization procedures.

The address generation logic of this module was synthesized for a 130 nm CMOS standard cells library, its maximum speed is 350Mhz, although we require only 200Mhz. At 200Mhz the area consumption is of just $0.02mm^2$.

## 3. OPTIMIZED INTERLEAVER CORES

A generic, table-based, interleaver such as the one described in the previous section is not always the best solution, even in the SDR context. The internal interleaver of turbo coders / decoders is tightly coupled with the channel coding / decoding. It would be rather inefficient to delegate this specific interleaving to an external generic interleaver. This is especially true for the decoder because the interleaver is used several times per iteration of the decoding process and the number of iterations is frequently 5 to 10 per code word. A dedicated, compact and energy efficient address generator, embedded directly inside the channel coder / decoder seams a much better solution. The two address generators presented below are dedicated to the 3GPP UMTS and 3GPP LTE turbo interleavers. They implement all variants of these two standards at a much lower hardware cost than the generic interleaver. Their major drawback, of course, is that they would not handle a possible new standard, with a totally different turbo interleaving scheme.

Contrary to other implementations, like (3), these interleavers are already designed to be embedded in a complete SoC in which a CPU is in charge of controlling the whole digital baseband processing. Thanks to this CPU, some block-length specific parameters are computed in software and directly used by the hardware address generators. This HW / SW partitioning allows for a more efficient hardware utilization.

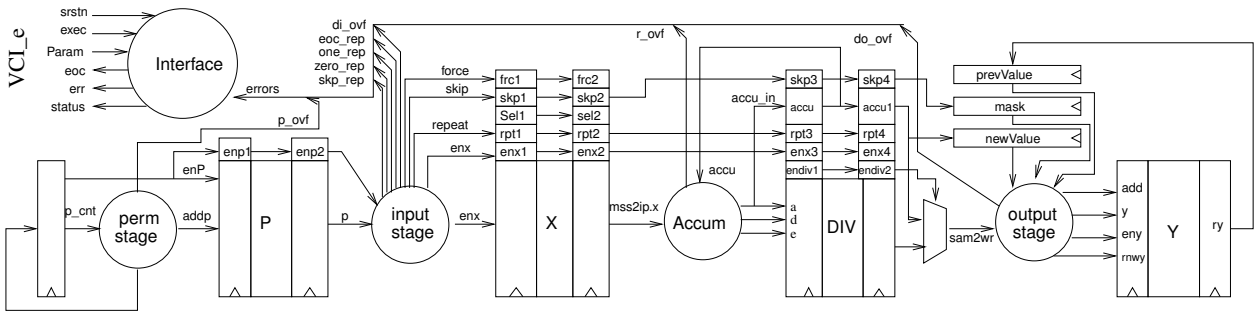


**Fig. 5**. Full reconfigurable Interleaver controller

### 3.1. LTE turbo encoder interleaver

This module implements an address generator for the LTE turbo encoder and decoder as defined in (2). The block of samples to interleave is first written in a buffer in the natural order. It is then read out according to the sequence of addresses produced by the address generator. The inverse permutation is applied by writing the input block with the address generator and reading out in natural order.

The 3GPP LTE turbo internal interleaver is defined in (2) as $\pi(i) = P(i) = (f_1 \times i + f_2 \times i^2) \bmod k$, where $f_1$ and $f_2$ are two parameters depending on the block size $k$. The module generates a sequence of indexes $P(0), P(1), \ldots P(k-1)$. $P(i)$ is the position in the input block of the $i^{th}$ sample of the output block. The first output index $P(0)$ is thus the index in the input block of the first output sample. The parameters used by the module $(\alpha, \beta)$ are calculated in advance from the $f_1, f_2$ parameters of the 3GPP specification: $\alpha = (f_1 + f_2) \bmod k$ and $\beta = (2 \times f_2) \bmod k$.

In the course of an interleaving / deinterleaving operation, the $i$ index increments from 0 to $k-1$. The module takes benefit of this simple scheme to compute the polynomial $P(i)$ incrementally:

$$\begin{aligned} P(i+1) &= f_1 \times (i+1) + f_2 \times (i+1)^2 \\ &= P(i) + f_1 + 2 \times i \times f_2 + f_2 \\ &= P(i) + Q(i) \\ Q(i+1) &= Q(i) + 2 \times f_2 = Q(i) + \beta \end{aligned}$$

With $P(0) = 0$ and $Q(0) = f_1 + f_2 = \alpha$. The sequence wraps around $k$ naturally. There is no need to detect the last value of a sequence or to reload $Q(0)$ and $P(0)$ in their corresponding registers. Indeed $P(k) = 0 = P(0)$ and $Q(k) = \alpha = Q(0)$. Because $P(i), Q(i), \alpha, \beta$ are naturals less than $k$, the modulus $k$ reduction when computing $P(i+1)$ and $Q(i+1)$ is a simple comparison with $k$ and an optional subtraction by $k$. The implementation is depicted in figure6.

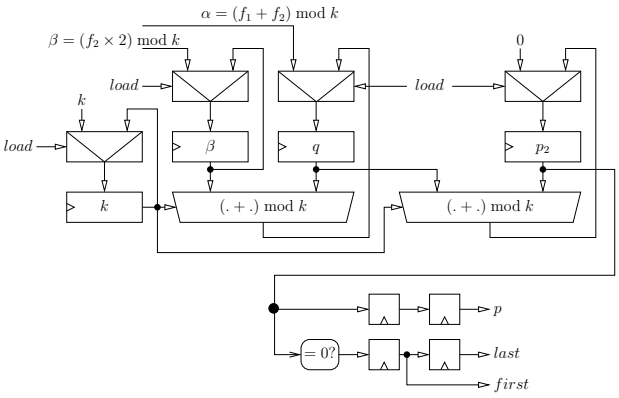


**Fig. 6**. Optimized architecture for LTE turbo interleaver

### 3.2. UMTS turbo encoder interleaver

This module implements an address generator for the UMTS turbo encoder and decoder as defined in (1). As in the LTE case, the interleaving / deinterleaving is implemented by writing in natural order and reading in permuted order or the opposite.

The module generates a sequence of indexes $n_0, \ldots n_{k-1}$. The $n_i$ index is the position in the input block of the $i^{th}$ output sample. The first output value $n_0$ is thus the position in the input block of the first output sample.

After hardware / software partitioning, it was decided to pre-compute several parameters in software. From the UTMS standard, the parameters that are precomputed are: number of rows `rp`, number of columns `c`, prime number `p`. We also precompute the serie `x`, which can be of 5, 10 or 20 elements, where $x_i = (v^{q_i} \times 256) \bmod p$, all this parameters are passed to the module before starting operation.

Once having the precomputed parameters, the interleaver implements the following steps, according to the UMTS standard for turbo code internal interleaver:

1. Find the $\langle S(j)\rangle_{j\in\{0,1,\ldots,p-1\}}$ for intra-row permutation, where $S(j) = s(\nu \times s(j-1)) \bmod p$
2. Find the $\langle q_i\rangle_{i\in\{0,1,\ldots,rp-1\}}$, where $q_0$ is equal to 1; and $q_i$ is to be a least primer integer number such that gdc($q_i$,$p-q$)=1; and $q_i > 6$ and $> q_{i-1}$ for each i=1,2,3...,R-1.
3. Find the $\langle r_i\rangle_{i\in\{0,1,\ldots,rp-1\}}$ sequence, where $r_T(i)$=$q_i$, $i = 0, 1, 2 \ldots, rp$ and $\langle T(I)\rangle_{I\in\{0,1,\ldots,rp-1\}}$ is selected from the standard according to the block size `k`.
4. Perform intra-row permutation as: $U_i(j) = s((j \times r_i) \bmod (p-1))$; where $U_i(j)$ is the original bit position of j-th permuted bit of i-th row.
5. Perform inter-row permutation as $U_{T(i)}(j)$
6. Read the matrix column by column.

In our algorithm, $U$ sequences of 5, 10 or 20 elements are constructed incrementally. The current value of each $U$ sequence is stored in a shift register ($u^{(i)}$ in figure 7). The $j^{th}$ value of the $T(i)^{th}$ $U$ sequence is:

$$\begin{aligned} U_{T(i)}(j) &= S((j \times r_{T(i)}) \bmod (p-1)) \\ &= (v^{(j\times r_{T(i)}) \bmod (p-1)}) \bmod p \end{aligned}$$

As by definition $r_{t(i)} = q_i$, it can be written:

$$U_{T(i)}(j) = (v^{(j\times q_i) \bmod (p-1)}) \bmod p$$

Fermat's little theorem allows to rewrite this expression as:

$$U_{T(i)}(j) = (v^{j\times q_i}) \bmod p = \left(\frac{x_i}{256}\right)^j \bmod p$$

From which it comes:

$$U_{T(i)}(j \neq 0) = \left(U_{T(i)}(j-1) \times \frac{x_i}{256}\right) \bmod p$$

With $U_{T(i)}(0) = 1$. Then, using $\overline{\times}(a,b,p)$ 8 bits modular Montgomery (7) multiplication:

$$\overline{\times}(a,b,p) = \left(\frac{a\times b}{256}\right) \bmod p$$

We end up with $U_{T(i)}(j \neq 0) = \overline{\times}(U_{T(i)}(j-1), x_i, p)$

### 3.3. Architecture

This core is based on a Montgomery multiplication implementation, with two shift registers as inputs, $x^{(i)}$ and $u^{(i)}$ for the two sequences $U_{T(i)}(j-1)$ and $x_i$ respectively. The Montgomery multiplication is implemented in 5 stages pipeline (corresponding to the minimum number of rows). The first 4 stages of the pipeline implement the 8 iterations of the 8 bits modular Montgomery multiplier, 2 bits of $x_i$ at a time, LSBs first. The fifth stage implements a final reduction modulus $p$.

As depicted in figure 7, for a new interleaving procedure, the current value of each $U$ sequence is initialized to 1. Then, these values enter a five stages pipeline, one after the other. The output of the pipeline is fed back to the first position in the $u^{(i)}$ pipeline (which is different for 5, 10 and 20 rows schemes). A 5, 10 or 20 stages pipeline ($x^{(i)}$) contains the $x_i$. The $x^{(i)}$ and $u^{(i)}$ shift registers move together so that the right $U$ and $x$ values are always input to the four first stages of the modular Montgomery multiplier pipeline. The modular Montgomery multiplier also uses a 5 stages $w^{(i)}$ shift register to store the intermediate results of the modular Montgomery multiplication. Once computed, the current value of each $U$ sequence enters a 3 stages post-processing unit. This unit computes the actual $n_i$ indexes from the $U$ sequences as specified in the 3GPP standards. The first stage computes the input row and column indexes. The second stage computes the input bit position. The $U$ sequences naturally wrap so there is no need for an initialization between two sequences with the same parameter set.

**Table 1**. Design Results comparison

| Design | Process | Standards | Area ($mm^2$) | Freq (MHz) |
|---|---|---|---|---|
| Design(13) | 180 nm | UMTS | 0.24 | 130 |
| Design(3) | 90 nm | UMTS, LTE | ND | ND |
| This work | 130 nm | UMTS, LTE | 0.044 | 350 |

### 3.4. implementation Results

The two address generators modules were synthesized for a 130 nm CMOS standard cells library, with 350 MHz a target clock frequency. Table 1 summarizes the results. Silicon area is used instead of the more classical (and more technology-independent) gate count because some previous works count instances of standard cells while others count *gate equivalents*, that is, the less buffered, 2-inputs, NAND gate. Taking into account the technology shift our design outperforms (13) in terms of silicon area, maximum frequency and flexibility. It is probably very close from the one of (3), which is very similar. The main differences are the utilization of a Montgomery modular multiplier, the absence of internal RAM (2 K-bits in (3)) and the pre-computation of parameters in our case, while they require extra hardware to do this.

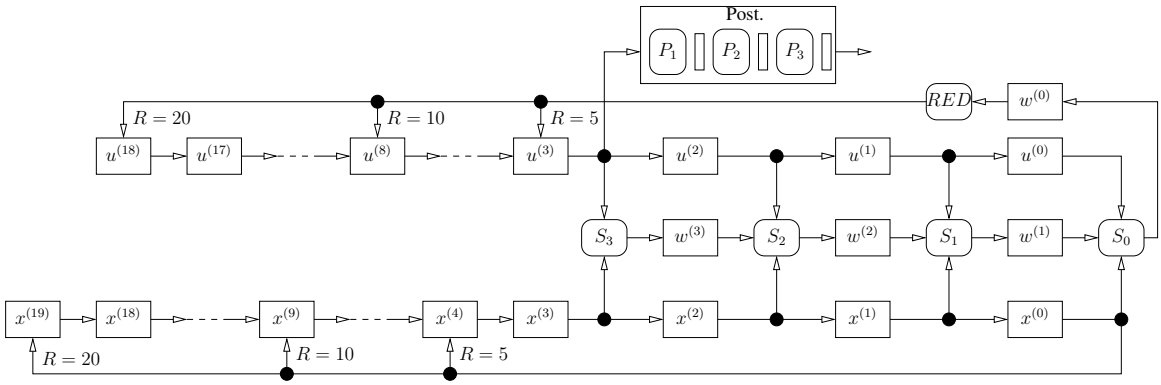


**Fig. 7**. Optimized architecture for UMTS turbo interleaver

### 4. CONCLUSIONS

This paper presents three architectures of interleavers targeting complex SDR applications. The first is a full reconfigurable design that can be used for practically all known standards of communications, and providing more flexibility than any other architecture of our knowledge. The two others are optimized for the LTE and UMTS turbo interleavers respectively. They are extremely compact and fast but their flexibility is limited to the variants of their target standard.

## References


[1] 3GPP. Multiplexing and channel coding (fdd), 25-212 rel 920.

[2] 3GPP. Evolved universal terrestrial radio access (e-utra); multiplexing and channel coding, 36-212 rel 920.

[3] ASGHAR, R., AND LIU, D. Dual standard reconfigurable hardware interleaver for turbo decoding. In *Wireless Pervasive Computing, 2008. ISWPC 2008. 3rd International Symposium on* (7-9 2008), pp. 768 –772.

[4] CHANG, Y.-N. A low-cost dual-mode deinterleaver design. *Consumer Electronics, IEEE Transactions on 54*, 2 (May 2008), 326–332.

[5] HAO, D., YAO, P., AND HOEHER, P. Analysis and design of interleaver sets for interleave-division multiplexing and related techniques. In *Turbo Codes and Related Topics, 2008 5th International Symposium on* (Sept. 2008), pp. 432–437.

[6] MITOLA, J. The software radio architecture. *Communications Magazine, IEEE 33*, 5 (May 1995), 26–38.

[7] MONTGOMERY, P. L. Modular multiplication without trial division. *Mathematics of Computation 44*, 170 (1985), 519–521.

[8] MUHAMMAD, N.-U.-I., RASHEED, R., PACALET, R., KNOPP, R., AND KHALFALLAH, K. Flexible baseband architectures for future wireless systems. pp. 39 –46.

[9] PARK, S.-J., AND JEON, J.-H. Interleaver optimization of convolutional turbo code for 802.16 systems. *Communications Letters, IEEE 13*, 5 (May 2009), 339–341.

[10] SADJADPOUR, H., SLOANE, N., SALEHI, M., AND NEBE, G. Interleaver design for turbo codes. *Selected Areas in Communications, IEEE Journal on 19*, 5 (May 2001), 831–837.

[11] SANCHEZ-ORTIZ, C., PARRA-MICHEL, R., AND GUZMAN-RENTERIA, M. Design and implementation of a multi-standard interleaver for 802.11a, 802.11n, 802.16e & dvb standards. In *Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on* (Dec. 2008), pp. 379–384.

[12] WANG, M., SHEIKH, A., AND QI, F. Interleaver design for short turbo codes. In *Global Telecommunications Conference, 1999. GLOBECOM '99* (1999), vol. 1B, pp. 894–898 vol. 1b.

[13] WANG, Z., AND LI, Q. Very low-complexity hardware interleaver for turbo decoding. *Circuits and Systems II: Express Briefs, IEEE Transactions on 54*, 7 (july 2007), 636 –640.

[14] WU, Y.-W., TING, P., AND MA, H.-P. A high speed interleaver for emerging wireless communications. In *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on* (June 2005), vol. 2, pp. 1192–1197.