Hindawi Publishing Corporation
Journal of Computer Systems, Networks, and Communications
Volume 2010, Article ID 793807, 16 pages
doi:10.1155/2010/793807

Research Article

Multimode Flex-Interleaver Core for Baseband Processor Platform

Rizwan Asghar and Dake Liu

Department of Electrical Engineering, Linköping University, 581 83 Linköping, Sweden

Correspondence should be addressed to Rizwan Asghar, rizwan@isy.liu.se

Received 25 August 2009; Accepted 12 October 2009

Academic Editor: Rashid Saeed

This paper presents a flexible interleaver architecture supporting multiple standards like WLAN, WiMAX, HSPA+, 3GPP-LTE, and DVB. Algorithmic level optimizations like 2D transformation and realization of recursive computation are applied, which appear to be the key to reach to an efficient hardware multiplexing among different interleaver implementations. The presented hardware enables the mapping of vital types of interleavers including multiple block interleavers and convolutional interleaver onto a single architecture. By exploiting the hardware reuse methodology the silicon cost is reduced, and it consumes 0.126 mm^2 area in total in 65 nm CMOS process for a fully reconfigurable architecture. It can operate at a frequency of 166 MHz, providing a maximum throughput up to 664 Mbps for a multistream system and 166 Mbps for single stream communication systems, respectively. One of the vital requirements for multimode operation is the fast switching between different standards, which is supported by this hardware with minimal cycle cost overheads. Maximum flexibility and fast switchability among multiple standards during run time makes the proposed architecture a right choice for the radio baseband processing platform.

1. Introduction

Growth of high-performance wireless communication systems has been drastically increased over the last few years. Due to rapid advancements and changes in radio communication systems, there is always a need of flexible and general purpose solutions for processing the data. The solution not only requires adopting the variances within a particular standard but also needs to cover a range of standards to enable a true multimode environment. The symbol processing is usually done in baseband processors. A fully flexible and programmable baseband processor [1–3] provides a platform for true multimode communication. To handle the fast transition between different standards, such type of platform is needed in both mobile devices and especially in base stations. Other than symbol processing, one of the challenging area is the provision of flexible subsystems for forward error correction (FEC). FEC subsystems can further be divided in two categories, channel coding/decoding and interleaving/deinterleaving. Among these categories, interleavers and deinterleavers appeared to be more silicon consuming due to the silicon cost of the permutation tables used in conventional approaches. For multistandard support devices the silicon cost of the permutation tables can grow much higher, resulting in an unefficient solution. Therefore, the hardware reuse among different interleaver modules to support multimode processing platform is of significance. This paper presents a flexible and low-cost hardware interleaver architecture which covers a range of interleavers adopted in different communication standards like HSPA Evolution (HSPA+) [4], 3GPP-LTE [5], WiMAX; IEEE 802.16e [6], WLAN; IEEE 802.11a/b/g [7], IEEE 802.11n [8], and DVB-T/H [9].

Interleaving plays a vital role in improving the performance of FEC in terms of bit error rate. The primary function of the interleaver is to improve the distance properties of the coding schemes and to disperse the sequence of bits in a bit stream so as to minimize the effect of burst errors introduced in transmission [10, 11]. The main categories of interleavers are block interleavers and convolutional interleavers. In block interleavers the data are written row wise in a memory configured as a row-column matrix and then read column-wise after applying certain intra-row and inter-row permutations. They are usually specified in the

form of a row-column matrix with row and/or column permutations given in tabular form, however; they can also be specified by a modulo function having more complex functions involved to define the permutation patterns. On the other hand, convolutional interleavers use multiple first-in-first-out (FIFO) cells with different width and depth. They are defined mainly by two parameters, the depth of memory cells and number of branches.

Looking at the range of interleavers used in different standards (Table 1) it seems difficult to converge to a single architecture; however, the fact that multimode coverage does not require multiple interleavers to work at the same time provides opportunities to use hardware multiplexing. The multimode functionality is then achieved by fast switching between standards. This research is to merge the functionality of different types of interleavers into a single architecture to demonstrate a way to reuse the hardware for a variety of interleavers having different structural properties. The method in general is the so-called hardware multiplexing technique well presented in [12]. It starts at analyzing and profiling multiple implementation flows, identifying opportunities of hardware multiplexing, and eventually fine tuning the microarchitecture, using minimal hardware, and maximal reuse of multifunctions.

This paper is organized as follows. Section 2 presents the previous work done for the interleaver algorithm implementations. The challenges involved to cover the wide range of standards are mentioned in Section 3. It also presents a shared data flow and hardware cost associated with different implementations. Section 4 provides the detailed explanation of the unified interleaver architecture and its subblocks. A brief explanation of the algorithmic transformations and optimizations used for efficient mapping onto single architecture is given in Section 5 with selected example cases. The usage of the proposed architecture while integrating into baseband system is explained in Section 6. Section 7 provided the VLSI implementation results and comparison to others followed by a conclusion in Section 8.

2. Previous Work

A variety of interleaver implementations having different structural properties have been addressed in literature. The main area of focus has been low cost and throughput. Most of the work covers a single or a couple of interleaver implementations which is not sufficient for a true multimode operation. The design of interleaver architecture for turbo code internal interleaver has been addressed in [13–17]. Some of these designs targeted very low-cost solutions. A recent work in [18] provides a good unified design for different standards; however, it covers only the turbo code interleavers and does not meet the complete baseband processing requirements demanding an all-in-one solution. The work in [19–22] covers the DVB-related interleaver implementations. Literature [23–27] focuses on more than one interleaver implementations with reconfigurability for multiple variants of wireless LAN and DVB. High-throughput interleaver architectures for emerging wireless communications based on MIMO-OFDM techniques have been addressed in [25, 27]. These techniques require multiple-stream processing in parallel, thus requiring parallel addresses generation and memory architecture as shown in Figure 1.

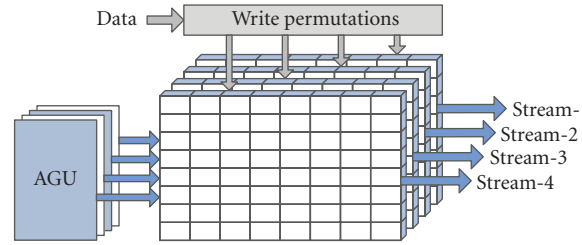

Figure 1: 3D view of interleaver configuration for a multistream communication system.

Some commercial solutions [28–30] from major FPGA vendors are also available for general purpose use. The available literature reveals that they do not compute the row or column permutations on the fly; instead they take row or column permutation tables in the form of a configuration file as input and use them to generate the final interleaved address. In this way, the complexity for on-the-fly computation of permutation patterns is avoided. This approach needs extra memory to store the permutation patterns. As these implementations are targeted for FPGA use only, they also enjoy the availability of dual port block RAM, which is not a good choice for chip implementations.

3. Shared Data Flow and Algorithm Analysis

The motivation of the research is to explore an all-in-one reconfigurable architecture which can help to meet fast time-to-market requirements from industry and customers. A summary of targeted interleaver implementations which are being widely used is provided in Table 1. The broadness of the interleaving algorithms gives rise to many challenges when considering a true multimode interleaver implementation. The main challenges are as follows:

(i) on the fly computation of permutation patterns,

(ii) wide range of interleaving block sizes,

(iii) wide range of algorithms,

(iv) fast switching between different standards,

(v) sufficient throughput for high-speed communications,

(vi) maximum standard coverage,

(vii) acceptable silicon cost and power consumption.

Exploring the similarities between different interleaving algorithms a shared data flow in general is shown in Figure 2. This data flow is shared by different interleaver types summarized in Table 1. Many of the interleaver algorithms, for example, [4, 6–9] need some preprocessing before starting actual interleaving process. Therefore the whole data flow has been divided into two phases named as precomputation phase as shown in Figure 2(a) and the execution phase as shown in Figure 2(b). There are many

Table 1: List of algorithms and permutations in different interleaver implementations and the cost comparison.

Standard	Interleaver type	Algorithm/permutation methodology	HW cost	
			Addr. Gen. @65 nm (μm^2)	Data memory @6 soft bits (kbits)
HSPA+	BTC	Multistep computation including intra-row permutation computation $S(j) = (v \times S(j-1))\%p$; $r(i) = T(q(i))$; $U(i,j) = S((j \times r(i))\%(p-1))$; $q\text{mod}(i) = r(i)\%(p-1)$; $RA(i,j) = \{RA(i,j-1) + q\text{mod}(i)\}\%(p-1)$; $I_{i,j} = \{C \times r(i)\} + U(i,j)$	12816	59.92
	1st, 2nd, and HS-DSCH int.	Standard block interleaving with given column permutations. $\pi(k) = \left(P\left\lfloor \frac{k}{R} \right\rfloor + C \times (k\%R)\right)\%K_\pi$	2288	29.96
LTE	QPP for BTC	$I_{(x)} = (f_1 \,.\, x + f_2 \,.\, x^2)\%N$	3744	72.0
	Sub-Blk. int.	Standard block interleaving with given column permutations.	2080	36.0
WiMAX	Channel interleaver	Two step permutation $M_k = \left(\frac{N}{d}\right) \times (k\%d) + \left\lfloor \frac{k}{d} \right\rfloor$; $J_k = s \times \left\lfloor \frac{M_k}{s} \right\rfloor + \left(\left(M_k + N - \left\lfloor d \times \frac{M_k}{N} \right\rfloor\right)\%s\right)$	8944	9.0
	Blk. int. b/w RS & CC	Standard block interleaver without any permutations	2080	19.92
	CTC interleaver	$I_{(x\%4=0)} = (P_0 \cdot x + 1)\%N$; $I_{(x\%4=1)} = \left(P_0 \cdot x + 1 + \frac{N}{2} + P1\right)\%N$; $I_{(x\%4=2)} = (P_0 \cdot x + 1 + P1)\%N$; $I_{(x\%4=3)} = \left(P_0 \cdot x + 1 + \frac{N}{2} + P3\right)\%N$	7280	56.25
WLAN	Channel interleaver	Two step permutation $M_k = \left(\frac{N}{d}\right) \times (k\%d) + \left\lfloor \frac{k}{d} \right\rfloor$; $J_k = s \times \left\lfloor \frac{M_k}{s} \right\rfloor + \left(\left(M_k + N - \left\lfloor d \times \frac{M_k}{N} \right\rfloor\right)\%s\right)$	8944	1.68
802.11n	Ch. Interleaver with frequency rotation	Two step permutation as above, with extra frequency interleaving, that is, $R_k = \left[J_k - \left\{\left(((i_{ss}-1) \times 2)\%3 + 3\left\lfloor \frac{i_{ss}-1}{3} \right\rfloor\right) \times N_{ROT} \times N_{BPSC}\right\}\right]\%N$	11563	24.54
DVB-H	Outer conv. interleaver	Permutation defined by depth of first FIFO branch (M) and number of total braches.	12272	8.76
	Inner bit interleaver	Six parallel interleavers with different cyclic shift $H_e(w) = (w + \Delta)\%126$; where $\Delta = 0, 63, 105, 42, 21$ and 84	3120	0.738
	Inner symbol interleaver	$y_{H(q)} = x_q$ for even symbols; $y_q = x_{H(q)}$ for odd symbols; where $H(q) = (i\%2) \times 2^{N_r-1} + \sum_{j=0}^{N_r-2} R_i(j) \times 2^j$;	3536	35.4
General purpose use	Row or/and Col. Perm. Given	Standard block interleaver with or without row or/and column permutation.	3952	24.0
Total cost	$\sum$ (all)	Independent implementations	$\sim$ 82619	$\sim$ 378.0
This work	Reconfigurable Solution	HW Multiplexed Design	27757	72.0

minor differences in both the phases when we consider different types of interleavers; however, one of the main differences might be due to the type of interleaver, that is, block interleaver or convolutional interleaver. Other than the differences in address calculation for the two categories, a major difference is the memory access mechanism. In case of block interleaver the memory read and write is explicit but a convolutional interleaver needs to write and

Table 2: Architecture exploration for different standards.

Standard	Interleaver type	Block size	Adders/ comparator	Multiplier	HW LUT	Configurable LUT/registers	Memory size (SB: soft bits)
HSPA+	Prime interleaver for BTC	5114	7	1	20×5b 440×7b 52×14b	20×8b 256×8b	$2 \times 5114 \times$ SB
	1st, 2nd, and HS-DSCH interleaving	5114	2	1	15×3b 32×5b	—	$5114 \times$ SB
3GPP-LTE	QPP interleaver for BTC	6144	5	—	188×19b	2×13b	$2 \times 6144 \times$ SB
	Sub-Block interleaver	6144	2	1	32×5b	—	$6144 \times$ SB
WiMAX (802.16e)	Channel interleaver	1536	5	1	15×4b	2×2b 1×11b	$1536 \times$ SB
	Block interleaver b/w RS and CC	2550	2	1	—	—	2550×8b
	CTC interleaver	2400	4	—	32×27b	1×12b	$4 \times 2400 \times$ SB
WLAN (802.11 a/b/g)	Channel interleaver	288	5	1	15×4b	2×2b 1×9b	$288 \times$ SB
802.11n Enhanced WLAN	Channel interleaver with frequency rotation	2592	9	1	30×4b 24×9b	2×2b 2×10b	$4 \times 648 \times$ SB
DVB ETSI EN 300-744	Outer convolutional interleaver	1122	4	1	—	11×11b	357×8b 765×8b
	Inner bit interleaver	126	8	—	—	21×1b 126×1b	$2 \times 126 \times 1$b $2 \times 126 \times 2$b
	Inner symbol interleaver	6048	1	—	30×1b	—	6048×6b
General purpose use	Row or/and Column permutation given as a table	4096	2	1	—	256×8b 64×6b	$4096 \times$ SB

read at the same time. This demands a dual port memory; however, it has been dealt by dividing the memories and introducing a delay in the read path. To get the general idea of cost saving by using hardware multiplexed architecture with shared data flow, each of the algorithms is implemented separately after applying appropriate algorithmic transformations. Comparing the hardware cost for different implementations as given in Table 1, the proposed hardware multiplexed architecture based on shared data flow provides 3 times lower silicon cost for address generation and about 5 times lower silicon cost for data memory in shared mode. Going through all the interleaver implementations given in Table 1, different hardware requirements for computing elements and memory are summarized in Table 2. Looking at the modulo computation requirements, the use of adder appears to be the common computing element for all kinds of implementations. Further observation reveals that adder is mostly followed by a selection logic. Therefore, a common computing cell named *acc_sel* as shown in Figure 3 is used to cover all the cases. Table 2 shows that the computational part of the reconfigurable implementation can be restricted to have 8 additions, 1 multiplication, and a comparator.

The memory requirements for different implementations are also very wide, due to different sizes, width, memory banks and ports. The memory organization and address computation is explained in detail in the next section.

4. Multimode Interleaver Architecture

The study from algorithm analysis provides the basis to multiplex the hardware intensive components and combine the functionality of multiple types of interleavers. The architecture for the multimode interleaver is given in Figure 4. The hardware partitioning is done in such a way that all computation intensive components are included in the address generation block. The other partitioned blocks are register file, control-FSM, and memory organization block. These blocks are briefly described in the following subsections.

4.1. Address Generation (ADG) Block. Address generation is the main concern for any kind of interleaving. Unified address generation is achieved by multiplexing the computation intensive blocks mentioned in Table 2. The address generation hardware is shown in detail in Figure 4. It is surrounded by other blocks like control FSM, register file, and some lookup tables. It utilizes 8 *acc_sel* units with a multiplier and a comparator. The reconfigurability is mainly achieved through changing the behavior of *acc_sel*

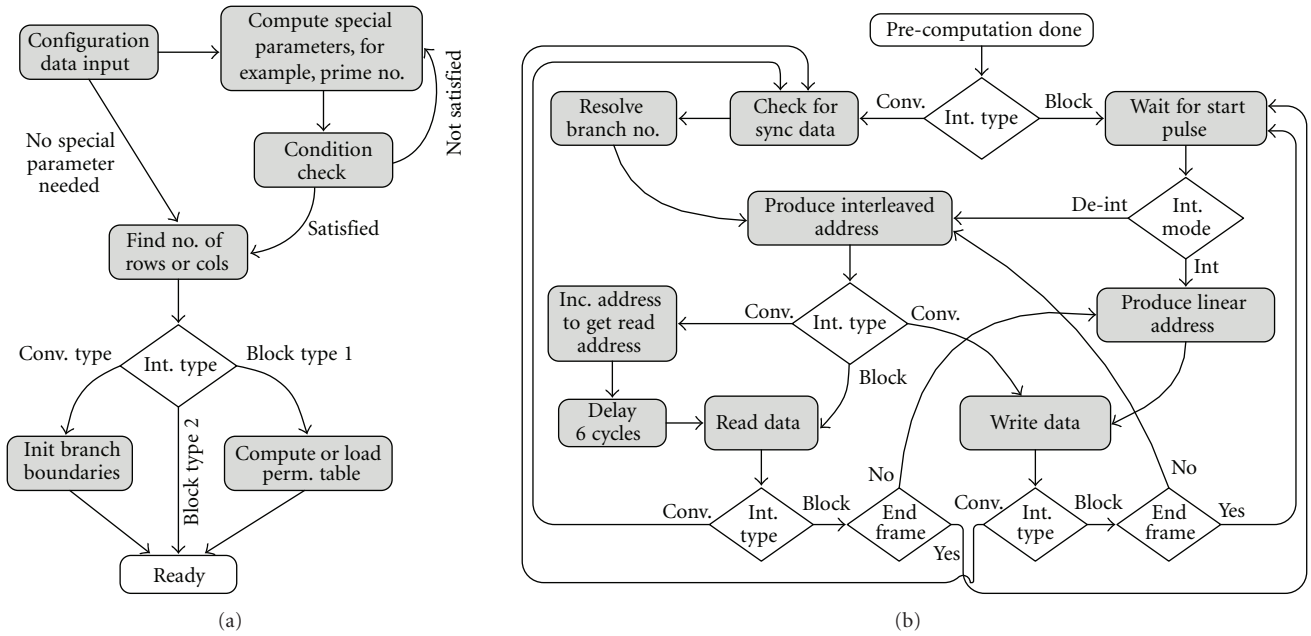

Figure 2: Data flow graph for (a) precomputation phase (b) execution phase.

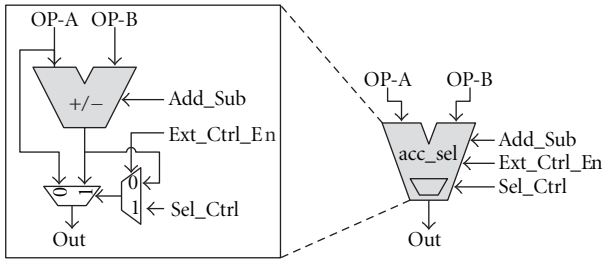

Figure 3: An accumulation and selection cell (acc_sel).

and appropriate multiplexer selection. The control signals *Add_Sub*, *Ext_Ctrl_En* and, *Sel_Ctrl* are used to define the behavior of *acc_sel* block. Using these signals in an appropriate way this block can be configured as an adder, a subtractor, a modulo operation with MSB of output as select line, or just a bypass. All the combinations are fully utilized and make it a very useful common computing element. The address generation block takes the configuration vector and configures itself with the help of a decoder block and part of the LUT. The configuration vector is 32 bit wide, which defines block size, interleaver depth, interleaving modes, and modulation schemes.

The ADG block generates the interleaved address based on all the permutations involved for implementing a block interleaver, whereas it generates memory read and write addresses concurrently while implementing a convolutional interleaver. The role of ADG block to be used as an interleaver or deinterleaver is mainly controlled by the controller after employing an addressing combination (permuted or sequential addressing) for writes and reads from the memory.

4.2. Control FSM. Two modes of operation for the hardware are defined as precomputation mode and execution mode. In order to handle the sequence of operations in the two modes a multistate control-FSM is used. The flow graph of the control-FSM is shown in Figure 5. During precomputation phase, the FSM may perform two main functions: (1) computation of necessary parameters required for interleaver address computation and (2) initialization of registers to become ready for execution phase. Other than IDLE state, 5 states (S1∼S4, S8) are assigned for precomputation. The common parameter to be computed in the precomputation phase is number of rows or columns; however, some specific parameters like prime number p; and intra-row permutation sequence $S(j)$ in WCDMA turbo code interleaver are also computed during this phase. For the interleaver functions which do not require precomputation, the initialization steps for precomputation are bypassed, and the control FSM directly jumps to the execution phase. The extra cycle cost associated with the precomputation has been investigated for the current implementation and the results are presented in a later section. In the execution phase, the control-FSM helps in sequencing the loading of data frames into memory or reading data frames from memory. In total 4 states (S5∼S7, S9) are assigned for execution phase. S9 is used for convolutional interleaver case only, whereas states S5∼S7 are reused for all types of interleavers. During the execution phase the control-FSM keeps track of block size also by employing row and column counter, thus providing the block synchronization required for each type of interleaver implementation.

4.3. Register File. The requirement of temporary storage of parameters arises with many types of interleaver implementations. Register requirements from different implementations are listed in Table 2. Some special usage configuration is also required for different cases; for example, WCDMA turbo

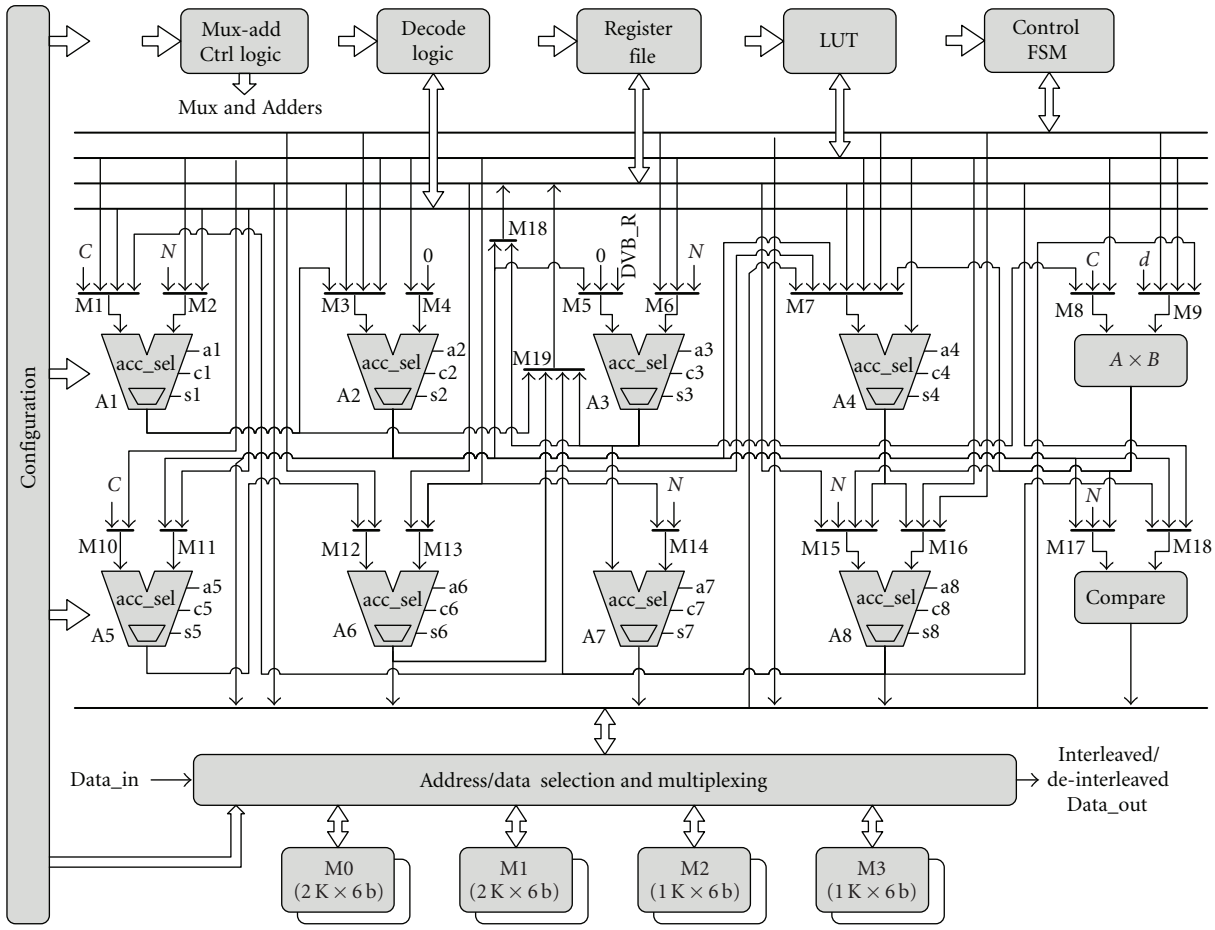

Figure 4: Address generation schematic in detail.

code interleaver needs 20 registers to form a circular buffer, convolutional interleaver in DVB requires 11 registers to be used as a general purpose register file, and the bit interleaver in DVB requires a long chain of single bit registers. Due to small size and special configuration requirements, a general purpose register file is not feasible here, and a fully customized register file is used. The width of registers is not the same and it is optimized as per requirement from different implementations. The registers can also be connected to form a chain, thus the single bit buffer for a bit interleaver is managed by circulating the shifted output inside register file. The two data input ports of the register file are fed through multiplexers M18 and M19 as shown in Figure 4.

4.4. Memory Organization. Memory requirements for different types of interleaver implementations are very much different as listed in Table 2. Also, soft bit processing in the decoder implies different requirements of bit width for different conditions and decoding architectures. The maximum width requirement is 6 bits for symbol interleaving and 8 bits for part of the memory in WCDMA. Multistream transmission requires multiple banks of memories in parallel. The size of the memory is taken as $2 \times 6144 \times SB$, which is due to large block size requirements for 3GPP-LTE, 3GPP-WCDMA, and DVB.

Memory partitioning is mainly motivated by the high-throughput requirements from the multistream system, for example, 802.11n. It requires four memory banks in parallel which appears to be a good choice to meet other requirements as well. Parallel memory banks can also be used in series to form a big memory. Partial parallelism can also be used where larger memory width is needed. Another worth full benefit of using multiple memory banks is avoiding the use of dual port RAM, which is not silicon efficient. Thus all the memories in the design are single port memories. The interleaved addresses for block and convolution interleavers computed by address generation block are combined according to the configuration requirement to make the final memory address. Figure 6 shows the memory organization with address selection logic. Particularly for convolutional interleaving, a small delay line with depth of 6 in the path of read addresses and control signals is used to avoid the data write and read for the same memory in a single clock cycle.

5. Algorithm Transformation for Efficient Mapping

The main objective is to use single architecture for interleaver implementation with maximum hardware sharing among

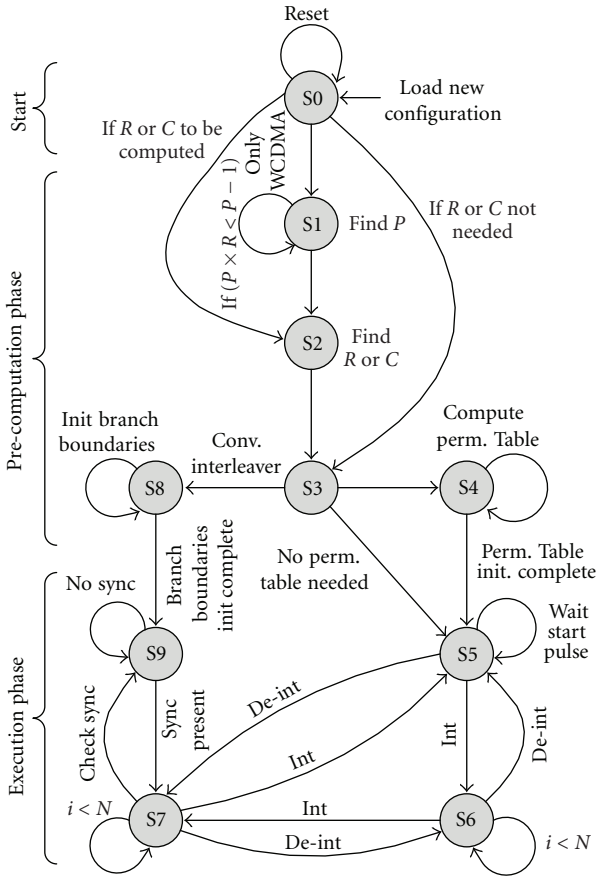

Figure 5: FSM state graph.

different algorithms. The versatility of interleaving algorithms makes it an in-efficient implementation when original algorithms are directly mapped to same architecture. On the other hand some transformations based on modular algebra can be applied on the original algorithms to make them hardware efficient. Same algorithmic transformations can be used to reach to an efficient hardware multiplexing among different standards. The following subsections present some transformation examples for selected algorithms which are very much versatile in the implementation point of view. These subsections cover channel interleaving for WiMAX and WLAN including 802.11n with frequency rotation, turbo code block interleaving for LTE, WiMAX, and HSPA Evolution, and convolutional interleaving used in DVB.

5.1. Channel Interleaving in WiMAX and WLAN. The channel interleaving in 802.11a/b/g (WLAN) and 802.16e (WiMAX) is of the same type. The interleaver function defined by a set of two equations for two steps of permutations, provides spatial interleaving, whereas the newly evolved standard 802.11n [8] based on MIMO-OFDM employs frequency interleaving in addition to spatial interleaving. Most of literature available [31–36] covers the performance and evaluation of WLAN interleaver design for a high-speed communication system; however, some recent work [23–27] focuses on interleaver architecture design including some complexity reduction techniques along with feasibility to gain higher throughput. The 2D realization of interleaver functions is exploited to enable efficient hardware implementation. The two steps of permutations for index k for interleaver data are expressed by the following equations:

$$M_k = \left(\frac{N}{d}\right) \times (k\%d) + \left\lfloor \frac{k}{d} \right\rfloor, \tag{1}$$

$$J_k = s \times \left\lfloor \frac{M_k}{s} \right\rfloor + \left(\left(M_k + N - \left\lfloor d \times \frac{M_k}{N} \right\rfloor \right) \% s \right). \tag{2}$$

Here N is the block size corresponding to number of coded bits per allocated subchannels and the parameter s is defined as $s = \max\{1, N_{\mathrm{BPSC}}/2\}$ where N_{BPSC} is the number of coded bits per subcarrier, (i.e., 1, 2, 4 or 6 for BPSK, QPSK, 16-QAM, or 64-QAM, resp.). The operator % is the modulo function computing the remainder and the operator $\lfloor x \rfloor$ is the floor function, that is, rounding x towards zero. The range of n and k is defined as $0, 1, 2, \ldots (N-1)$. The direct implementation of the above mentioned equations is very much hardware in-efficient and also the mapping onto the proposed unified interleaver architecture is not possible. Therefore, realization of two 1D equations into 2D space and computation of interleaved address in recursive way is adopted to reduce the hardware complexity as explained in the following subsections.

5.1.1. BPSK-QPSK. As N_{BPSC} is 1 and 2 for BPSK and QPSK, respectively; thus $s = 1$ for both cases and (2) simplifies to the following form:

$$J_k = \left(\frac{N}{d}\right) \times (k\%d) + \left\lfloor \frac{k}{d} \right\rfloor. \tag{3}$$

Considering the interleaver as a block interleaver, the parameter d is usually considered as total number of columns N_{COL}, and parameter N/d is taken as total number of rows N_{ROW}, but the column and row definition are swapped hereafter. The parameter d is taken as total number of rows and parameter N/d is taken as total number of columns. The functionality still remains the same, with the benefit that it ends up with the recursive expression for all the modulation schemes. According to new definitions, the term $(k\%d)$ provides the behavior of row counter and the term $\lfloor k/d \rfloor$ provides the behavior of column counter. Thus introducing two new variables i and j as two dimensions, such that j increments when i expires, the ranges for i and j are mentioned as follows:

$$i = 0, 1, \ldots (d-1), \qquad j = 0, 1, \ldots \left(\frac{N}{d} - 1\right), \tag{4}$$

which satisfies against k when $i = (k\%d)$ and $j = \lfloor k/d \rfloor$. Defining total number of columns as $C = N/d$, (3) can be written as

$$J_{i,j} = C \times i + j. \tag{5}$$

The recursive form after handling the exception against $i = 0$ can be written as

$$J_{i,j} = \begin{cases} j, & \text{if } (i = 0), \\ J_{(i-1),j} + C, & \text{otherwise.} \end{cases} \tag{6}$$

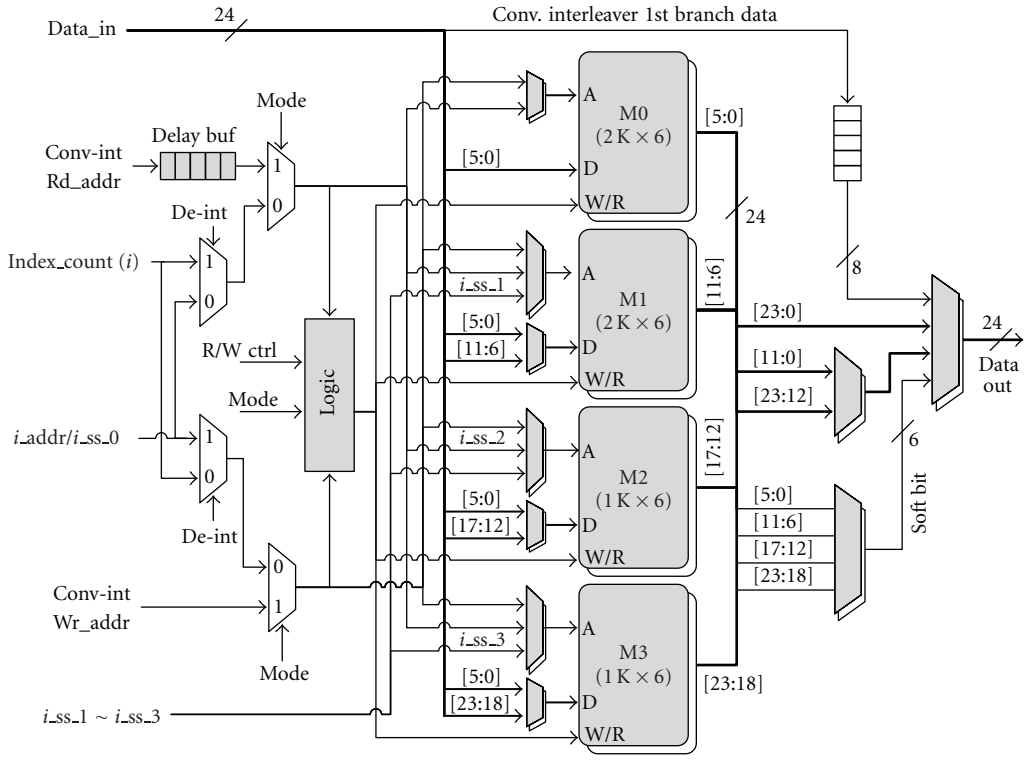

Figure 6: Memory address selection and data handle.

Defining row counter i as $i = R_c$ and column counter j as $j = C_c$, the hardware for (6) is shown in Figure 7(a). The case of BPSK and QPSK do not carry any specific inter-row or inter-column permutation pattern; thus it ends up with relatively simple hardware, but it provides the basis for analysis for 16-QAM and 64-QAM cases, which are more complicated.

5.1.2. 16-QAM. 16-QAM scheme has 4 code bits per subcarrier; thus parameter s is 2 and (2) becomes

$$J_k = 2 \times \left\lfloor \frac{M_k}{2} \right\rfloor + \left(\left(M_k + N + \left\lfloor \frac{d \times M_k}{N} \right\rfloor \right) \%2 \right). \tag{7}$$

Like BPSK/QPSK case, algebraic only steps cannot be used here to proceed due to the presence of floor and modulo functions. Instead, all the possible block sizes for 16-QAM are analyzed to restructure the above equation. The following structure appears to be equivalent to (7) and at the same time resembles the structure of (3); thus it suits well for hardware multiplexing:

$$J_k = \left(\frac{N}{d} \right) \times (k\%d) + \left\lfloor \frac{k}{d} \right\rfloor + r_k^2. \tag{8}$$

The extra term r_k^2 is defined by the following expression:

$$\begin{aligned} r_k^2 = {} & [(1 - (k\%2)) - (k\%2)] \left\{ 1 - \left(\left\lfloor \frac{k}{d} \right\rfloor \%2 \right) \right\} \\ & + [((k\%2) - 1) + (k\%2)] \left\{ \left\lfloor \frac{k}{d} \right\rfloor \%2 \right\}. \end{aligned} \tag{9}$$

This term appears due to the reason that the interleaver for 16-QAM carries specific permutation patterns, making the structure more complicated. Considering the 2-dimensions i and j having range as mentioned in (4), the behavior of the term $k\%2$ is the same as that of $i\%2$, when i is the row counter. Thus (8) can be written in 2D representation as follows:

$$J_{i,j} = \begin{cases} j, & \text{if } (i = 0), \\ J_{(i-1),j} + C + r_{i,j}^2, & \text{otherwise}, \end{cases} \tag{10}$$

where

$$\begin{aligned} r_{i,j}^2 = {} & [(1 - (i\%2)) - (i\%2)] \{1 - (j\%2)\} \\ & + [(i\%2) + (1 - (i\%2))] \{j\%2\}. \end{aligned} \tag{11}$$

The term can further be simplified to a smaller expression but it is easy to realize the hardware from its current form. The modulo terms can be implemented by using the LSB of row counter R_c and column counter C_c, and the required sequence can be generated with the help of an XOR gate and an adder as shown in Figure 7(b).

5.1.3. 64-QAM. The parameter s is 3 for 64-QAM; thus (2) becomes

$$J_k = 3 \times \left\lfloor \frac{M_k}{3} \right\rfloor + \left(\left(M_k + N + \left\lfloor \frac{d \times M_k}{N} \right\rfloor \right) \%3 \right). \tag{12}$$

The presence of modulo function $x\%3$ makes it much harder to reach some valid mathematical expression algebraically. Different structures for all possible block sizes for

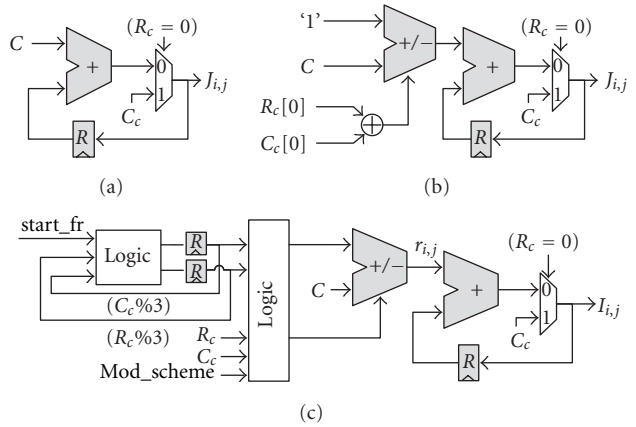

Figure 7: Interleaver address generation for (a) BPSK-QPSK, (b) 16-QAM, and (c) combined for all modulation schemes.

64-QAM are analyzed and the structure similar to (6) and (10) and equivalent to (12) is given as follows:

$$J_{i,j} = \begin{cases} j, & \text{if } (i = 0), \\ J_{(i-1),j} + C + r^3_{i,j}, & \text{otherwise,} \end{cases} \tag{13}$$

where i and j represent two dimensions and their range is given by (4). Defining $i' = (i\%3)$ and $j' = (j\%3)$, $r^3_{i,j}$ is given as

$$\begin{aligned} r^3_{i,j} = {} & \left((1 - j') + \frac{j'(j'-1)}{2}\right)\left\{2\left((1 - i') + \frac{i'(i'-1)}{2}\right) - \left(i' - \frac{i'(i'-1)}{2}\right)\right\} \\ & + (j' - j'(j'-1))\{2(i' - i'(i'-1)) - ((1 - i') + i'(i'-1))\} \\ & + \left(\frac{j'(j'-1)}{2}\right)\left\{2\left(\frac{i'(i'-1)}{2}\right) - \left(1 - \frac{i'(i'-1)}{2}\right)\right\}. \end{aligned} \tag{14}$$

The term $r^3_{i,j}$ provides the inter-row and inter-column permutation for $s = 3$ against row counter i and column counter j. The expression for $r^3_{i,j}$ looks very long and complicated, but eventually, it gives a hardware efficient solution as the terms inside braces are easier to generate through a very small lookup table. The generic form for (6), (10), and (13) to compute the interleaved address $I_{i,j}$ can be written as

$$I_{i,j} = \begin{cases} j, & \text{if } (i = 0), \\ I_{(i-1),j} + C + r^s_{i,j}, & \text{otherwise.} \end{cases} \tag{15}$$

Here parameter s distinguishes different modulation schemes. For BPSK/QPSK $r^1_{i,j} = 0$, and for 16-QAM and 64-QAM, $r^2_{i,j}$ and $r^3_{i,j}$ are given by (9) and (14), respectively. The hardware realization supporting all modulation schemes is shown in Figure 7(c). It appears to be a much optimized implementation as it involves only two additions, some registers, and a very small lookup table.

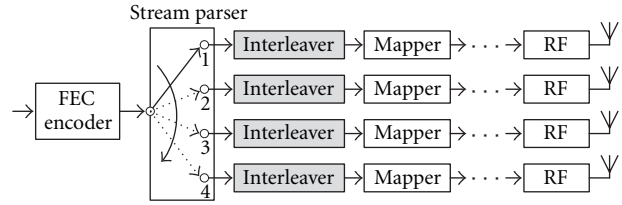

Figure 8: Use of interleaver in multiple spatial streams (802.11n).

5.2. Frequency Interleaving in 802.11n. The transmission in 802.11n can be distributed among four spatial streams as shown in Figure 8. The interleaving requires frequency rotation in case more than one spatial streams are being transmitted. The frequency rotation is applied to the output of the second permutation J_k. The expression for frequency rotation for spatial stream i_{ss} is given as follows:

$$R_k = \left[J_k - \left\{\left(((i_{\text{ss}} - 1) \times 2)\%3 + 3\left\lfloor \frac{i_{\text{ss}} - 1}{3} \right\rfloor\right) \times N_{\text{ROT}} \times N_{\text{BPSC}}\right\}\right]\%N. \tag{16}$$

Here N_{ROT} is the parameter which defines different frequency rotations for 20 MHz and 40 MHz case in 802.11n. The frequency rotation also depends on the index of the spatial stream i_{ss}, thus each spatial stream faces different frequency rotations. Defining the rotation term as J_{ROT}, that is,

$$J_{\text{ROT}} = \left\{\left(((i_{\text{ss}} - 1) \times 2)\%3 + 3\left\lfloor \frac{i_{\text{ss}} - 1}{3} \right\rfloor\right) \times N_{\text{ROT}} \times N_{\text{BPSC}}\right\}, \tag{17}$$

we have

$$R_k = (J_k - J_{\text{ROT}})\%N. \tag{18}$$

The range for the term $(J_k - J_{\text{ROT}})$ is not bounded and it can have value greater than $2N$; thus direct implementation cannot be low cost. Analyzing the two terms $[J_k\%N]$ and $(-J_{\text{ROT}})\%N$ separately, it is observed that the second term provides the starting point for computing the rotation R_k. As the rotation is fixed for a specific spatial stream, thus the starting value $r_{ks} = (-J_{\text{ROT}})\%N$ also holds for all run time computations. Equation (18) in combination with (10) can be written as

$$J^{i_{\text{ss}}}_{i,j} \equiv R_k = \left(J_{i,j} + r_{ks}\right)\%N. \tag{19}$$

Here $J^{i_{\text{ss}}}_{i,j}$ is the joint address after applying both, spatial interleaving and frequency interleaving against row index i, column index j and spatial stream index i_{ss}. A lookup can be used for the starting values for r_{ks} against different spatial

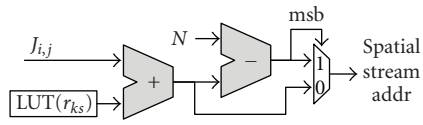

Figure 9: HW for frequency rotation in 802.11n.

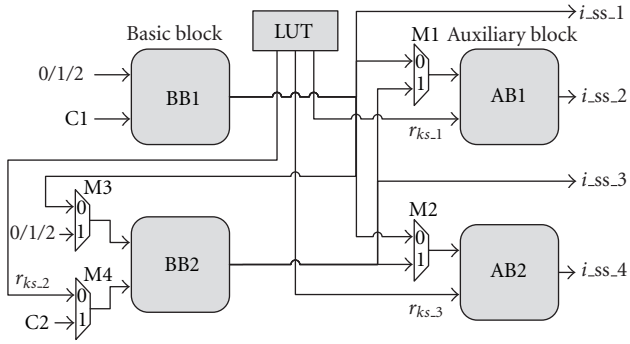

Figure 10: HW for quad stream interleaver.

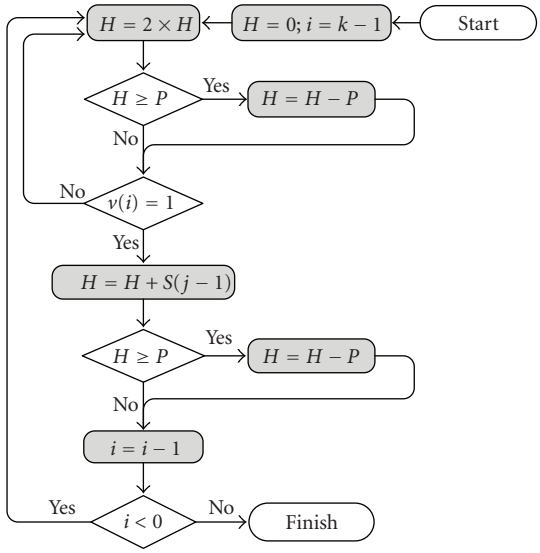

Figure 11: Flow graph for interleaved modulo multiplication algorithm.

streams. The r_{ks} values for all the cases follow the condition, that is, $(r_{ks} < N)$ which depicts that the term $(J_k + r_{ks})$ cannot be larger than $2N$. Therefore, the frequency rotation can be computed with a very small hardware as shown in Figure 9.

5.3. Multistream Interleaver Support in 802.11n. The spatial interleaver address generation block shown in Figure 7(c) is denoted as Basic Block (BB) and the frequency rotation block as shown in Figure 9 is denoted as Auxiliary Block (AB). Both these blocks combine to form a complete address generation circuit for one spatial stream. In order to provide support for four streams in parallel, one may consider replicating the two blocks four times. However, an optimized solution would be to use 2 basic blocks and 2 auxiliary blocks, still providing support for 4 spatial streams. The hardware block diagram to generate the interleaver addresses for multiple streams in parallel is shown in Figure 10. This hardware supports quick configuration changes thus providing full support to any multitasking environment. If some new combination of modulation schemes is needed to be implemented, which is not supported already, the interfacing processor can do task scheduling for different types of modulation schemes.

5.4. Turbo Code Interleaver for HSPA+. The channel coding block in HAPA+ including WCDMA uses turbo coding [37] for forward error correction. 3GPP standard [4] proposes the algorithm for block interleaving in turbo encoding/decoding as mentioned below. Here N is the block size, R is the row size, and C is the column size in bits.

(i) Find appropriate number of rows "R", prime number "p", and primitive root "v" for particular block size as given in the standard.

(ii) Col Size:

$$\begin{aligned} &C = p - 1, \quad \text{if } (N \leq R \times (p-1)), \\ &C = p, \quad \text{if } (R \times (p-1) < N \leq (R \times p)), \\ &C = p + 1, \quad \text{if } (R \times p < N). \end{aligned} \tag{20}$$

(iii) Construct intra-row permutation sequence S(j) by

$$S(j) = [v \times S(j-1)]\%p; \quad j = 1, 2, \ldots p-2. \tag{21}$$

(iv) Determine the least prime integer sequence $q(i)$ for $i = 1, 2, \ldots R-1$, by taking $q(0) = 1$, such that $g.c.d(q(i), p-1) = 1$, $q(i) > 6$ and $q(i) > q(i-1)$.

(v) Apply inter-row permutations to $q(i)$ to find $r(i)$:

$$r(i) = T(q(i)). \tag{22}$$

(vi) Perform the intra-row permutations Ui, j, for $i = 0, 1, \ldots R-1$ and $j = 0, 1, \ldots p-2$.

If $(C = p)$: $Ui, j = S[(j \times r(i)) \bmod (p-1)]$ and $Ui, (p-1) = 0$.

If $(C = p+1)$: $Ui, j = S[(j \times r(i)) \bmod (p-1)]$, and $Ui, (p-1) = 0$, $Ui, p = p$, and if $(N = R \times C)$ then exchange $U(R-1, 0)$ with $U(R-1, p)$.

If $(C = p-1)$: $Ui, j = S[(j \times r(i)) \bmod (p-1)] - 1$.

(vii) Perform the inter-row permutations.

(viii) Read the address columns wise.

The presence of complex functions like modulo computation, intra-row and inter-row permutations, multiplications, finding least prime integers, and computing greatest common divisor makes it in-efficient while implementing it in its original form. Further, to get one interleaving address in each cycle, some preprocessing is also required where parameters like total number of rows or columns, least prime number sequence $q(i)$, inter-row permutation patterns $T(i)$, intra-row permutations $S(j)$, prime number

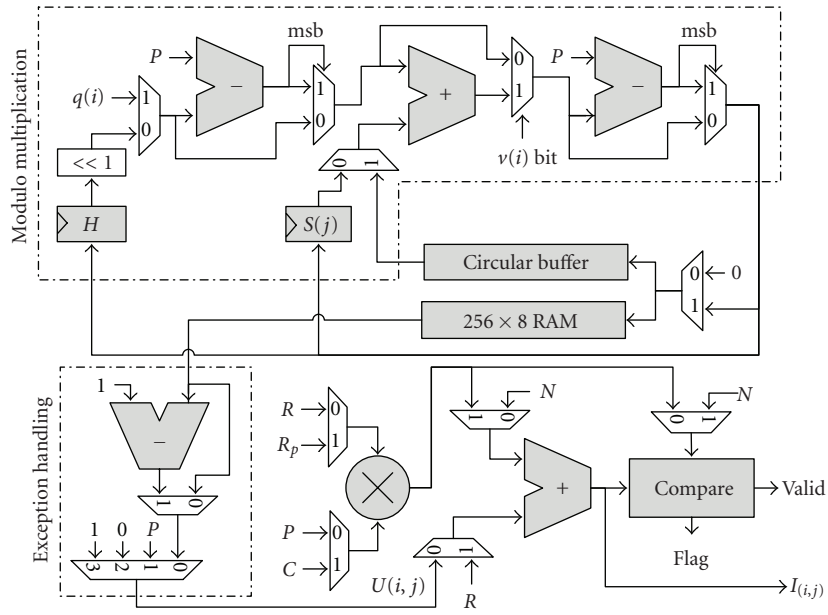

Figure 12: WCDMA turbo code interleaver hardware.

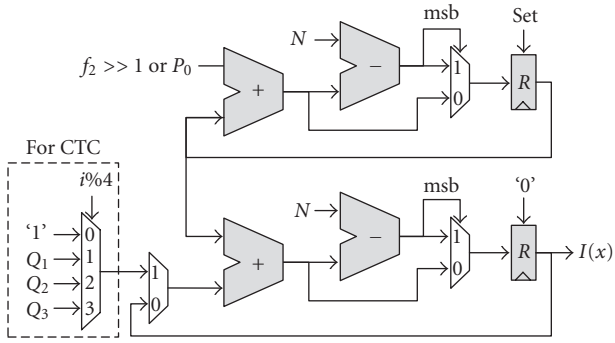

Figure 13: Simplified HW for 3GPP-LTE and CTC interleaver.

p, and associated integer v are computed. Some of these parameters can be computed using lookup tables while the others need some close loop or recursive computations. The simplifications considered in the implementation are discussed in the following paragraphs.

One of the main hurdles to generate on-the-fly interleaved address is the computation of intra-row permutation sequence $S(j)$. Before applying the intra-row permutations, the term $(j \times r(i)\%(p-1))$ is computed which produces random values due to $r(i)$ and modulo function. These random values appear as index to compute $S(j)$, due to which it may require many clock cycles to be computed on-the-fly. To resolve it, some precomputations are made and results are stored in a memory. These precomputations involve the computation of a modulo function which requires a divider for direct implementation. To avoid the use of divider, indirect computation of modulo function is done by using Interleaved Modulo Multiplication Algorithm [38]. It computes the modulo function in an iterative way requiring more than one clock cycles. Looking at maximum value of v, which is 5 bits, a maximum of 5 iterations are needed to compute one modulo multiplication. The algorithm to compute the Interleaved Modulo Multiplications is shown in Figure 11 and the hardware required is shown in Figure 12. This hardware produces the data for memory while in precomputation phase; however, same hardware is utilized to generate the address for the memory, while in execution phase. The usage of memory depends on the parameter p and it will be filled upto $(p-2)$ locations.

Finding $q\text{mod}(i) = q(i)\%(p-1)$ instead of direct computation of least prime number sequence $q(i)$ gives the benefit of computing the RAM address recursively and avoiding computation of the modulo function. This idea was introduced in [13] and later on it has been used in [14, 16, 17]. The computation of $q(i)\%(p-1)$ can be managed by a subtractor and a look up table, provided that all the values of $q(i)$ placed in the look up table satisfy the condition $q(i) < 2(p-1)$. The similarities between different sequences for $q(i)\%(p-1)$ for all possible p values are very helpful to improve the efficiency of the lookup table. The parameters p and v are stored in combined fashion in a lookup table of size 52×14b. The lookup table is addressed via a counter. Against each value of p, the condition $(p \times R \geq N - R)$ is checked using a comparator to find the appropriate value for p and v. Once p is found, the total number of columns C can have only three values, that is, $p-1$, p, or $p+1$. Hence C is found in at most three clock cycles by checking the condition $(R \times C \geq N)$. The recursive function used to compute the RAM address with the help of parameter $q\text{mod}(i)$ is given by

$$RA(i,j) = \{RA(i,j-1) + q\text{mod}(i)\}\%(p-1). \quad (23)$$

The data from RAM are denoted as $U(i,j)$ after passing through some exception handling logic. Parameter $U(i,j)$ provides the intra-row permutation pattern for a particular row. The final interleaved address $I_{i,j}$ can be found

by combining the inter-row permutation with intra-row permutation as follows:

$$I_{i,j} = \{C \times r(i)\} + U(i,j). \tag{24}$$

The complete hardware for interleaver address generation for Turbo Code interleaver is shown in Figure 12. It can be mapped to the proposed unified interleaver architecture quite efficiently.

5.5. Turbo Code Interleaving in 3GPP-LTE and WiMAX. The newly evolved standard, 3GPP LTE [5], involves interleaving in the channel coding and rate matching section. The interleaving in rate matching is called subblock interleaving and is based on simple block interleaving scheme. The channel coding in LTE involves Turbo Code with an internal interleaver. The type of interleaver here is different and it is based on quadratic permutation polynomial (QPP), which provides very compact representation. The turbo interleaver in LTE is specified by the following quadratic permutation polynomial:

$$I_{(x)} = (f_1 \cdot x + f_2 \cdot x^2)\%N. \tag{25}$$

Here $x = 0, 1, 2, \ldots (N-1)$, with N as block size. This polynomial provides deterministic interleaver behavior for different block sizes and appropriate values of f_1 and f_2. Direct implementation of the permutation polynomial given in (25) is hardware in-efficient due to multiplications, modulo function, and bit growth problem. To simplify the hardware, (25) can be rewritten for recursive computation as

$$I_{(x+1)} = (I_{(x)} + g_{(x)})\%N, \tag{26}$$

where $g_{(x)} = (f_1 + f_2 + 2 \cdot f_2 \cdot x)\%N$. This can also be computed recursively as

$$g_{(x+1)} = (g_{(x)} + 2 \cdot f_2)\%N. \tag{27}$$

The two recursive terms mentioned in (26) and (27) are easy to implement in hardware (Figure 13) with the help of a LUT to provide the starting values for $g_{(x)}$ and f_2.

WiMAX standard [6] uses convolutional turbo coding (CTC) also termed duo-binary turbo coding. They can offer many advantages like performance, over classical single-binary turbo codes [39]. Parameters to define the interleaver function as described in [6] are designated as P_0, P_1, P_2, and P_3. Two steps of interleaving are described as follows.

Step 1. Let the incoming sequence be

$$u_0 = [(A_0, B_0), (A_1, B_1), (A_2, B_2), \ldots (A_{N-1}, B_{N-1})]; \tag{28}$$

for $x = 0 \cdots N-1$, *if* $(i\%2) = 1$, *then* $(A_i, B_i) = (B_i, A_i)$.

The new sequence is

$$u_1 = [(A_0, B_0), (B_1, A_1), (A_3, B_3), \ldots (B_{N-1}, A_{N-1})]. \tag{29}$$

Step 2. The function $I_{(x)}$ provides the address of the couple from the sequence u_1 that will be mapped onto address x of the interleaved sequence. $I_{(x)}$ is defined by the set of four expressions with a switch selection as follows:

for $x = 0 \cdots N-1$
switch $(x\%4)$.
case 0: $I_{(x\%4=0)} = (P_0 \cdot x + 1)\%N$.
case 1: $I_{(x\%4=1)} = (P_0 \cdot x + 1 + N/2 + P_1)\%N$.
case 2: $I_{(x\%4=2)} = (P_0 \cdot x + 1 + P_2)\%N$.
case 3: $I_{(x\%4=3)} = (P_0 \cdot x + 1 + N/2 + P_3)\%N$.

Combining the four equations provided in step-2, the interleaver function $I_{(x)}$ becomes

$$I_{(x)} = (\beta_x + Q_x)\%N, \tag{30}$$

where β_x can be computed using recursion, that is, $\beta_{(x+1)} = (\beta_x + P_0)\%N$ by taking $\beta_0 = 0 \cdot Q_x$ is given by

$$Q_x = \begin{cases} 1, & \text{if } (j\%4 = 0), \\ 1 + \dfrac{N}{2} + P_1, & \text{if } (j\%4 = 1), \\ 1 + P_2, & \text{if } (j\%4 = 2), \\ 1 + \dfrac{N}{2} + P_3, & \text{if } (j\%4 = 3). \end{cases} \tag{31}$$

As range of β_x and Q_x is less than N, thus I_x can be computed by using addition and subtraction with compare and select logic as shown in Figure 13.

5.6. Convolutional Interleaving in DVB. The convolutional interleaver used in DVB is based on the Forney [40] and Ramsey type III approach [41]. The convolutional interleaver being part of outer coding resides in between RS encoding and convolutional encoding. The convolutional interleaver for DVB consists of 12 branches as shown in Figure 14. Each branch j is composed of first-in-first-out (FIFO) shift registers with depth $j \times M$, where $M = 17$ for DVB. The

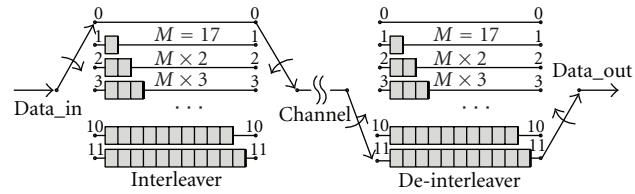

Figure 14: Convolutional interleaver and deinterleaver in DVB.

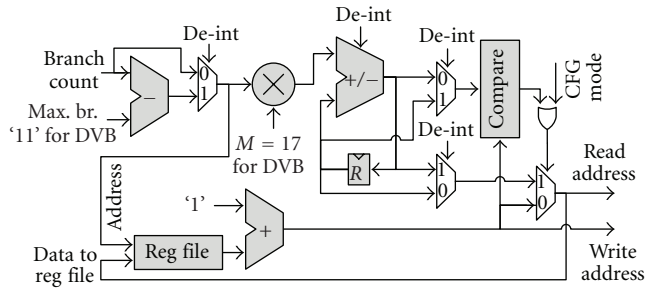

Figure 15: HW for RAM read/write address generation for convolutional interleaver.

Table 3: Precomputation cycle cost for different standards.

Standard	Worst case precomputation cycle cost
802.11 a/b/g—WLAN Channel interleaver	20
802.16e—WiMAX Channel interleaver	98
3GPP—WCDMA Block turbo code (Depends on Block size "N")	15 for ($N = 40$) 23 for ($N = 41$) 802 for ($N = 5040$) 563 for ($N = 5114$)
ETSI EN 300-744—DVB Inner symbol interleaver	15
802.11n—Extended WLAN	38
General purpose use	Depends on external HW, that is, loading the permutations
All others	Less than 3

Table 4: Summary of implementation results.

Parameter	Value
Target technology	65 nm
Memory configuration	$2048 \times 6b \times 4$; $1024 \times 6b \times 4$
Total memory	72 Kbit
Memory area	97972 μm^2
Memory power consumption	10.5 mW
Logic area	28436 μm^2
Total area	0.126 mm^2
Clock rate	166 MHz
Throughput (Max)	664 Mbps
Total power consumption	11.7 mW

packet of 204 bytes consisting of one sync byte (0×47 or $0\times B8$) is entered into the interleaver in a periodic way. For synchronization purpose the sync bytes are always routed to *branch*-0 of interleaver.

Convolutional interleaving is best suited for real time applications with some added benefits of half the latency and less memory utilization as compared to block interleaving. Recently, convolutional interleavers have been analyzed to work with Turbo codes [42–44], with improved performance, which make them more versatile; thus general and reconfigurable convolutional interleaver architecture integrated with block interleaver functionality can be of significance.

Implementation of convolutional interleavers using first-in-first-out (FIFO) register cells is silicon inefficient. To achieve a silicon efficient solution, RAM-based implementation is adopted. The memory partitioning is made in such a way that by applying appropriate read/write addresses in a cyclic way, it exhibits the branch behavior as required by a convolutional interleaver. RAM write and read addresses are generated by the hardware shown in Figure 15. The hardware components used here are almost the same as used by interleaver implementation for other standards, thus providing the basis for multiplexing the hardware blocks for reuse. To keep track of next write address for each branch, 11 registers are needed, which provides the idea of using cyclic pointers instead of using FIFO shift registers. For each branch the corresponding write address is provided by the concerned pointer register and next write address (which is also called current read address) is computed by using an addition and a comparison with the branch boundaries. Other reference implementations have used branch boundary tables directly, but to keep the design general, the branch boundaries are computed on-the-fly using an adder and a multiplier in connection with a branch counter.

For implementing a convolutional deinterleaver, the same hardware is used by implementing the branch counter in reverse order (decrementing by 1). In this way, same branch boundaries are used, and the only difference is that the sync byte in the data is now synchronized with the largest branch size as shown in Figure 14. Keeping the same branch boundaries for the deinterleaver, the width of the pointer register becomes fixed. This gives an additional benefit that the width of pointer register may be optimized efficiently.

6. Integration into Baseband System

The multimode interleaver architecture can perform interleaving or deinterleaving for various communication systems. It is targeted to be used as an accelerator core with a programmable baseband processor. The usage of the multimode interleaver core completely depends on the capability of the baseband processor. For lower throughput requirements only a single core can be utilized with baseband processor and the operations are performed sequentially. However, as a matter of fact, usual system level implementations require interleaver at multiple stages. Number of stages can be up to three, for example, WCDMA (turbo code interleaving, 1st interleaving, and 2nd interleaving). A fully parallel implementation can be realized by using three instances of the proposed multimode interleaver core, but in order to optimize the hardware cost a wise usage would be to use two instances hooked up with the main bus of the processor as shown in Figure 16. In this way the interleaving stages can be categorized as channel interleaving and coding/decoding interleaving. Further optimizations can be made in the two cores to fit in the particular requirements, for example, one interleaver core dedicated for coding/decoding and the second core dedicated for channel interleaving. By doing so the reduction of silicon cost associated with address generation is not significant, however, memory sizes can be optimized as per the targeted implementations, which can reduce the silicon cost significantly. For current implementation of multimode interleaver, the input memory used for any kind of decoding is considered to be the part of baseband processor data memory. In this way the extra memory inside interleaver core can be avoided which might be redundant in many cases. However, the integration of input memory in the main decoding operation is facilitated by the interleaver core by providing the address for input memory. In this way the interleaved/deinterleaved data can be fed to the decoder block in synchronized manner.

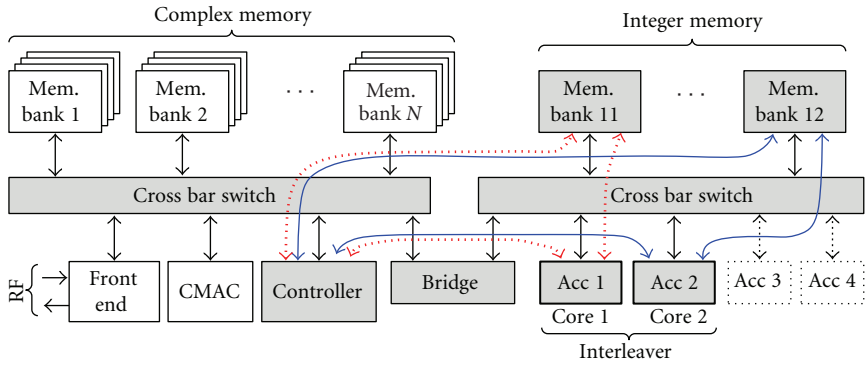

Figure 16: Integration of interleaver core with baseband processor.

Table 5: HW comparison with other implementations.

Implementation	Standard coverage	Technology	Operating frequency	Power	Memory size	Total core size
Xilinx [28] Virtex-5	General purpose (commercial use)	FPGA	262/360 MHz Speed Grade -1/-3	—	18 Kbits	210 LUTs + Memory
Altera [29] FLEX-10KE	General purpose (commercial use)	FPGA	120 MHz	—	16 Kbits	392 LEs + Memory
Lattice [30] ispXPGA	General purpose (commercial use)	FPGA	132 MHz	—	36 Kbits	284 LUTs + Memory
Shin and Park [13]	WCDMA turbo code; cdma2000	0.25 μm	—	—	35 Kbits	2.678 mm^2
Asghar et al. [18]	WCDMA, LTE, WiMAX and DVB-SH Turbo Code Interleaver Only	65 nm	200 MHz	10.04 mW	30 Kbits	0.084 mm^2
Chang and Ding [23]	WiMAX, WLAN, DVB	0.18 μm	100 MHz	—	12 Kbits	0.60 mm^2
Chang [24]	WiMAX, WLAN, DVB	0.18 μm	150 MHz	—	12 Kbits	0.484 mm^2
Wu et al. [25]	WiMAX, WLAN, 802.11n	0.18 μm	200 MHz	—	32 Kbits	0.72 mm^2
Asghar and Liu [26]	WiMAX, WLAN, DVB	0.12 μm	140 MHz	3.5 mW	12 Kbits	0.18 mm^2
Asghar and Liu [27]	WiMAX, WLAN, 802.11n	65 nm	225 MHz	4 mW	15.6 Kbits	0.035 mm^2
Horvath et al. [20]	DVB bit and symbol interleaver	0.6 μm	36.57 MHz	300 mW	48 Kbits	69 mm^2
Chang [21]	DVB bit and symbol interleaver	0.35 μm	—	—	52.2 Kbits	2.9 mm^2
This work	All range including WLAN, WiMAX, DVB, HSPA+, LTE, 802.11n and General purpose implementation	65 nm	166 MHz	11.7 mW	72 Kbits	0.126 mm^2

Although the main focus is to support the targeted standards, however, programmability of the processor may target some different types of interleaver implementation which is not directly supported by this core. To make it still usable, support for some indirect implementation of any block interleaver with or without having row or column permutations is also provided. In this case the interleaver core is configured to implement a general interleaver with external permutation patterns. The permutation patterns are computed inside baseband processor using its programmability feature and loaded in a couple of the interleaver memories during precomputation phase. Excluding these memories, a restriction on maximum block size (i.e., 4096) will be imposed in this case. This type of approach is adopted by all commercially available interleaver implementations like Xilinx [28], Altera [29], and Lattice Semiconductor [30]. The computation of interleaver permutations on processor side and loading them into memory can impose more computation and time overheads on the processor side. Another drawback is that it does not support fast switching between different interleaver

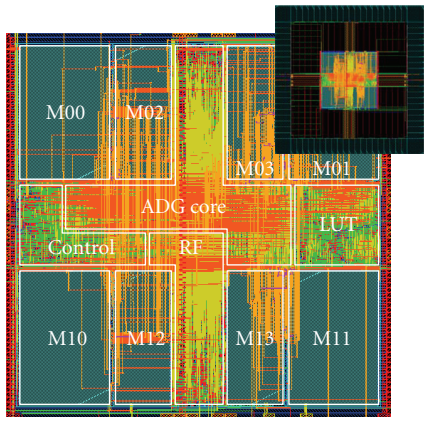

Figure 17: Layout of proposed multimode interleaver.

implementations. A real multimode processor may require fast transition from one standard to another; therefore, it is not a perfect choice for a real multimode environment. However, it is supported by the proposed multimode interleaver core for the completeness of the design.

7. Implementation Results

The reconfigurable hardware interleaver design shown in Figure 4 provides the complete solution for multimode radio baseband processing. The wide range of standard support is the key benefit associated with it. The RTL code for the reconfigurable interleaver design was written in Verilog HDL and the correctness of the design was verified by testing for maximum possible cases. Targeting the use of interleaver core with a multimode baseband processor, one of the important parameters to be investigated is precomputation cycle cost. A lower precomputation cycle cost is beneficial for fast switching between different standards. Table 3 shows the worst case cycle cost during precomputation for different interleavers. It is observed that the cycle cost in WCDMA is higher for some block sizes, but still it works fine, as it is less than the frame size and it can be easily hidden behind the first SISO decoding by the turbo decoder. The worst case precomputation cycle cost for other interleaver implementations is not very high. Therefore, the design supports fast switching among different standards and hence it is very much suitable for a multimode environment.

The multimode interleaver design was implemented in 65 nm standard CMOS technology and it consumes 0.126 mm^2 area. The chip layout is shown in Figure 17 and the summary of the implementation results is provided in Table 4. The design can run at a frequency of 166 MHz and consumes 11.7 mW power in total. Therefore, having 4-bit parallel processing for four spatial streams (e.g., 802.11n) maximum throughput can reach up to 664 Mbps. However, this throughput is limited to 166 Mbps for single stream communication systems. Table 5 provides the comparison of the proposed design to others in terms of standard coverage, silicon cost, and power consumption. The reference implementations have lower standard coverage as compared to the proposed design. Though more silicon is needed for more standard coverage, our solution still provides a good tradeoff with an acceptable silicon cost and power consumption.

8. Conclusion

This paper presents a flexible and reconfigurable interleaver architecture for multimode communication environment. The presented architecture supports a number of standards including WLAN, WiMAX, HSPA+, 3GPP-LTE, and DVB, thus providing coverage for maximum range. To meet the design challenges, the algorithmic level simplifications like 2D transformation of interleaver functions and recursive computation for different implementations are used. The major focus has been to compute the permutation patterns on-the-fly with flexibility. Architecture level results have shown that the design provides a good tradeoff in term of silicon cost and reconfigurability when comparing with other reference designs with less standard coverage. As compared to individual implementations for different standards, the proposed unified address generation offers a reduction of silicon by a factor of three. Finally, the basic requirement of a multimode processor platform, that is, fast switching between different standards has been met with minimal precomputation cycle cost. It enables the processor to use the interleaver core for one standard at some time and use it for another standard in the next time slot by just changing the configuration vector and small preprocessing overheads.

References

[1] A. Nilsson, E. Tell, and D. Liu, "An 11mm^2, 70 mW fully-programmable baseband processor for mobile WiMAX and DVB-T/H in 0.12 μm CMOS," *IEEE Journal of Solid State Circuits*, vol. 44, pp. 90–97, 2009.

[2] E. Tell, A. Nilsson, and D. Liu, "A low area and low power programmable baseband processor architecture," in *Proceedings of the 5th International Workshop on System-On-Chip for Real-Time Applications (IWSOC '05)*, pp. 347–351, Banff, Canada, July 2005.

[3] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, "A software-defined communications baseband design," *IEEE Communications Magazine*, vol. 41, no. 1, pp. 120–128, 2003.

[4] 3GPP, "Technical specification group radio access network; multiplexing and channel coding (FDD)," Technical Specification 25.212 V8.4.0, December 2008.

[5] 3GPP-LTE, "Technical specification group radio access network; E-UTRA; multiplexing and channel coding, release 8," Technical Specification 3GPP TS 36.212 v8.0.0, 2007–2009.

[6] IEEE 802.16e-2005, "IEEE standard for local and metropolitan area networks—part 16: air interface for fixed broadband wireless access systems—amendment 2," 2005.

[7] IEEE 802.11-2007, "Standard for local and metropolitan area networks—part 11: WLAN medium access control (MAC) and physical layer (PHY) specs," rev. of IEEE Std. 802.11-1999.

[8] IEEE P802.11n/D2.0, "Draft standard for enhanced WLAN for higher throughput," February 2007.

[9] ETSI EN 300-744 V1.5.1, "Digital video broadcasting (DVB); framing structure, channel coding and modulation for digital terrestrial television," November 2004.

[10] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1983.

[11] B. Sklar, *Digital Communications: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2nd edition, 2001.

[12] D. Liu, *Embedded DSP Processor Design, Application Specific Instruction Set Processors*, Morgan Kaufmann, San Mateo, Calif, USA, 2008.

[13] M.-C. Shin and I.-C. Park, "Processor-based turbo interleaver for multiple third-generation wireless standards," *IEEE Communications Letters*, vol. 7, no. 5, pp. 210–212, 2003.

[14] R. Asghar and D. Liu, "Very low cost configurable hardware interleaver for 3G turbo decoding," in *Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications (ICTTA '08)*, pp. 1–5, Damascus, Syria, April 2008.

[15] P. Ampadu and K. Kornegay, "An efficient hardware interleaver for 3G turbo decoding," in *Proceedings of IEEE Radio and Wireless Conference (RAWCON '03)*, pp. 199–120, August 2003.

[16] Z. Wang and Q. Li, "Very low-complexity hardware interleaver for turbo decoding," *IEEE Transactions on Circuits and Systems II*, vol. 54, no. 7, pp. 636–640, 2007.

[17] R. Asghar and D. Liu, "Dual standard re-configurable hardware interleaver for turbo decoding," in *Proceedings of the 3rd International Symposium on Wireless Pervasive Computing (ISWPC '08)*, pp. 768–772, Santorini, Greece, May 2008.

[18] R. Asghar, D. Wu, J. Eilert, and D. Liu, "Memory conflict analysis and implementation of a re-configurable interleaver architecture supporting unified parallel turbo decoding," *Journal of Signal Processing Systems*. In press.

[19] J. B. Kim, Y. J. Lim, and M. H. Lee, "A low complexity FEC design for DAB," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '01)*, vol. 4, pp. 522–525, Sydney, Australia, May 2001.

[20] L. Horvath, I. B. Dhaou, H. Tenhunen, and J. Isoaho, "A novel, high-speed, reconfigurable demapper-symbol deinterleaver architecture for DVB-T," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '99)*, vol. 4, pp. 382–385, Orlando, Fla, USA, May-June 1999.

[21] Y. -N. Chang, "Design of an efficient memory-based DVB-T channel decoder," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 5, pp. 5019–5022, Kaohsiung, Taiwan, May 2005.

[22] H. Afshari and M. Kamarei, "A novel symbol interleaver address generation architecture for DVB-T modulator," in *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT '06)*, pp. 989–993, Bangkok, Thailand, October 2006.

[23] Y.-N. Chang and Y.-C. Ding, "A low-cost dual-mode deinterleaver design," in *Proceedings of IEEE International Conference on Consumer Electronics (ICCE '07)*, pp. 1–2, Las Vegas, Nev, USA, January 2007.

[24] Y. N. Chang, "A low-cost dual mode de-interleaver design," *IEEE Transaction on Consumer Electronics*, vol. 54, no. 2, pp. 326–332, 2008.

[25] Y.-W. Wu, P. Ting, and H.-P. Ma, "A high speed interleaver for emerging wireless communications," in *Proceedings of the International Conference on Wireless Networks, Communications and Mobile Computing*, vol. 2, pp. 1192–1197, Maui, Hawaii, USA, June 2005.

[26] R. Asghar and D. Liu, "Low complexity multi mode interleaver core for WiMAX with support for convolutional interleaving," *International Journal of Electronics, Communications and Computer Engineering*, vol. 1, no. 1, pp. 20–29, 2009.

[27] R. Asghar and D. Liu, "Low complexity hardware interleaver for MIMO-OFDM based wireless LAN," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '09)*, pp. 1747–1750, Taipei, Taiwan, May 2009.

[28] Xilinx Inc., "Interleaver/De-Interleaver," Product Specification, v5.1, DS250, March 2008.

[29] Altera Inc., "Symbol Interleaver/De-Interleaver Core," Mega Core Function User's Guide, ver. 1.3.0, June 2002.

[30] Lattice Semiconductor Inc., "Interleaver/De-Interleaver IP Core," ispLever Core User's Guide, ipug_61_02.5, August 2008.

[31] X.-F. Wang, Y. R. Shayan, and M. Zeng, "On the code and interleaver design of broadband OFDM Systems," *IEEE Communications Letters*, vol. 8, no. 11, pp. 653–655, 2004.

[32] R. Van Nee, V. K. Jones, G. Awater, A. Van Zelst, J. Gardner, and G. Steele, "The 802.11n MIMO-OFDM standard for wireless LAN and beyond," *Wireless Personal Communications*, vol. 37, no. 3-4, pp. 445–453, 2006.

[33] H. Niu, X. Ouyang, and C. Ngo, "Interleaver design for MIMO-OFDM based wireless LAN," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC '06)*, vol. 4, pp. 1825–1829, Las Vegas, Nev, USA, 2006.

[34] J. Baltersee, G. Fock, and H. Meyr, "Achievable rate of MIMO channels with data-aided channel estimation and perfect interleaving," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 12, pp. 2358–2368, 2001.

[35] S. Ramseier, "Shuffling bits in time and frequency—an optimum interleaver for OFDM," in *Proceedings of the IEEE International Conference on Communications (ICC '03)*, vol. 5, pp. 3418–3422, May 2003.

[36] V. D. Nguyen and H.-P. Kuchenbecker, "Block interleaving for soft decision viterbi decoding in OFDM systems," in *Proceedings of the 54th IEEE Vehicular Technology Conference (VTC '01)*, vol. 1, pp. 470–474, 2001.

[37] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and encoding: turbo-codes," in *Proceedings of IEEE International Conference on Communications (ICC '93)*, vol. 2, pp. 1064–1070, Geneva, Switzerland, May 1993.

[38] G. R. Blakley, "A computer algorithm for calculating the product A*B mod M," *IEEE Transactions on Computers*, vol. 32, no. 5, pp. 497–500, 1983.

[39] J.-H. Kim and I.-C. Park, "Duo-binary circular turbo decoder based on border metric encoding for WiMAX," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '08)*, pp. 109–110, Seoul, Korea, March 2008.

[40] G. D. Forney, "Burst- correcting codes for the classic bursty channel," *IEEE Transactions on Communications*, vol. 19, no. 5, part 2, pp. 772–781, 1971.

[41] J. L. Ramsey, "Realization of optimum interleavers," *IEEE Transactions on Information Theory*, vol. 16, no. 3, pp. 338–345, 1970.

[42] S. Vafi and T. Wysocki, "Weight distribution of turbo codes with convolutional interleavers," *IET Communications*, vol. 1, no. 1, pp. 71–78, 2007.

[43] E. K. Hall and S. G. Wilson, "Stream-oriented turbo codes," *IEEE Transactions on Information Theory*, vol. 47, no. 5, pp. 1813–1831, 2001.

[44] S. Vafi and T. Wysocki, "On the performance of turbo codes with convolutional interleavers," in *Proceedings of Asia-Pacific Conference on Communications*, pp. 222–226, Perth, Wash, USA, October 2005.