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

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 inefficient solution. Therefore, the hardware reuse among different interleaver modules to support multimode processing platform is of significance. 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 mm² 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.
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.

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:

1. on-the-fly computation of permutation patterns,
2. wide range of interleaving block sizes,
3. wide range of algorithms,
4. fast switching between different standards,
5. sufficient throughput for high-speed communications,
6. maximum standard coverage,
7. 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
To further understand 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 read address for each symbol, which is a major difference in address calculation. The differences in both 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 minor differences, the differences in both 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.

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Interleaver type</th>
<th>Algorithm/permutation methodology</th>
<th>HW cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Addr. Gen.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>@65 nm (μm²)</td>
</tr>
<tr>
<td>HSPA+ BTC</td>
<td>Multistep computation including intra-row permutation computation</td>
<td>12816</td>
<td>59.92</td>
</tr>
<tr>
<td></td>
<td>( S(j) = (v \times S(j - 1)) % p ); ( r(i) = T(q(i)) );</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( U(i, j) = S((j \times r(i)) % (p - 1)) ); ( qmod(i) = r(i) % (p - 1) );</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( RA(i, j) = {RA(i, j - 1) + qmod(i)} % (p - 1); ( I_{j, i} = C \times r(i) ) + ( U(i, j) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HSPA+ 1st, 2nd, and HS-DSCH int.</td>
<td>Standard block interleaving with given column permutations.</td>
<td>2288</td>
<td>29.96</td>
</tr>
<tr>
<td></td>
<td>( \pi(k) = \left( p \left( \frac{k}{R} \right) + C \times (k % R) \right) % K_x )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LTE QPP for BTC</td>
<td>Two step permutation</td>
<td>3744</td>
<td>72.0</td>
</tr>
<tr>
<td></td>
<td>( I_{(x)} = (f_1 \cdot x + f_2 \cdot x^2) % N )</td>
<td>Standard block interleaving with given column permutations.</td>
<td>2080</td>
</tr>
<tr>
<td>LTE Sub-Blk. int.</td>
<td>Standard block interleaving with given column permutations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiMAX Channel interleaver</td>
<td>Two step permutation</td>
<td>8944</td>
<td>9.0</td>
</tr>
<tr>
<td></td>
<td>( M_k = \left( \frac{N}{d} \right) \times (k % d) + \left[ \frac{k}{d} \right] );</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( J_k = s \times \left[ \frac{M_k}{s} \right] + \left( \left( M_k + N - \left[ \frac{d \times M_k}{N} \right] \right) % s \right) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BS int. b/w RS &amp; CC</td>
<td>Standard block interleaver without any permutations</td>
<td>2080</td>
</tr>
<tr>
<td>CTC interleaver</td>
<td>Two step permutation</td>
<td>7280</td>
<td>56.25</td>
</tr>
<tr>
<td></td>
<td>( 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 );</td>
<td>Standard block interleaving with given column permutations.</td>
<td>7280</td>
</tr>
<tr>
<td></td>
<td>( 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 );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLAN Channel interleaver</td>
<td>Two step permutation</td>
<td>8944</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>( M_k = \left( \frac{N}{d} \right) \times (k % d) + \left[ \frac{k}{d} \right] );</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( J_k = s \times \left[ \frac{M_k}{s} \right] + \left( \left( M_k + N - \left[ \frac{d \times M_k}{N} \right] \right) % s \right) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>802.11n Ch. Interleaver with frequency rotation</td>
<td>Two step permutation as above, with extra frequency interleaving, that is,</td>
<td>11563</td>
<td>24.54</td>
</tr>
<tr>
<td></td>
<td>( R_k = \left( J_k + \left( {(i_a - 1) \times 2 } % 3 \right) \times \frac{i_a - 1}{3} \times N_{B} \times N_{BPSC} \right) % N );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVB-H Outer conv. interleaver</td>
<td>Permutation defined by depth of first FIFO branch (( M )) and number of total branches.</td>
<td>12272</td>
<td>8.76</td>
</tr>
<tr>
<td></td>
<td>( H_1(w) = (w + \Delta) % 126 ); where ( \Delta = 0, 63, 105, 42, 21 ) and 84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( y_{H(w)} = x_q ) for even symbols; ( y_{H(w)} = x_{H(w)} ) for odd symbols;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>where ( H(q) = (\Delta)^2 \times 2^{N-1} + \sum_{t=0}^{N-2} R_t \times 2^t );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVB-H Inner bit interleaver</td>
<td>Six parallel interleavers with different cyclic shift</td>
<td>3120</td>
<td>0.738</td>
</tr>
<tr>
<td></td>
<td>( H_1(w) = (w + \Delta) % 126 ); where ( \Delta = 0, 63, 105, 42, 21 ) and 84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVB-H Inner symbol interleaver</td>
<td>Permutation defined by depth of first FIFO branch (( M )) and number of total branches.</td>
<td>3536</td>
<td>35.4</td>
</tr>
<tr>
<td>General purpose use</td>
<td>Row or/and Col. Perm. Given</td>
<td>Standard block interleaver with or without row or/and column permutation.</td>
<td>3952</td>
</tr>
<tr>
<td>Total cost ( \sum ) (all)</td>
<td>Independent implementations</td>
<td>~ 82619</td>
<td>~ 378.0</td>
</tr>
<tr>
<td>This work Reconfigurable Solution</td>
<td>HW Multiplexed Design</td>
<td>27757</td>
<td>72.0</td>
</tr>
</tbody>
</table>
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 memory organization block. These blocks are briefly described in the following subsections.
and appropriate multiplexer selection. The control signals Add_SEL, 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
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 $2 \times 6144 \times 5B$, 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
different algorithms. The versatility of interleaving algorithms makes it an inefficient 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:

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

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

Here N is the block size corresponding to number of coded bits per allocated subchannels and the parameter s is defined as \( s = \max\{1, N_{\text{BPSC}}/2\} \) where \( N_{\text{BPSC}} \) is the number of coded bits per subcarrier, (i.e., 1, 2, 4 or 6 for BPSK, QPSK, 16-QAM, or 64-QAM, respectively). 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, …, \( (N-1) \). The direct implementation of the above mentioned equations is very much hardware inefficient 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 \( N_{\text{BPSC}} \) 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 \( N_{\text{COL}} \), and parameter \( N/d \) is taken as total number of rows \( N_{\text{ROW}} \), 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 \( [k/d] \) 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:

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

which satisfies against \( k \) when \( i = (k%d) \) and \( j = [k/d] \). 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

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

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

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:

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

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

\[
r_k^2 = [(1 - (k \% 2)) - (k \% 2)] \left\{ 1 - \left( \frac{k}{d} \% 2 \right) \right\} + [(k \% 2) - 1] + (k \% 2) \left\{ \frac{k}{d} \% 2 \right\}. \tag{9}
\]

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

\[
r_{i,j}^2 = [(1 - (i \% 2)) - (i \% 2)] \left\{ 1 - (j \% 2) \right\} + [(i \% 2) + (1 - (i \% 2))] \left\{ j \% 2 \right\}. \tag{11}
\]

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

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

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
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.

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

$$J_{i,j} = \begin{cases} j, & \text{if } (i = 0), \\ J_{(i-1),j} + C + r_{i,j}^2, & \text{otherwise}, \end{cases} \quad (13)$$

where $i$ and $j$ represent two dimensions and their range is given by (4). Defining $i' = (i \mod 3)$ and $j' = (j \mod 3)$, $r_{i,j}^2$ is given as

$$r_{i,j}^2 = \left\{ \begin{array}{ll} C_i, & \text{if } (i = 0), \\ R_{i,j} - C_i, & \text{otherwise} \end{array} \right. \quad (14)$$

The term $r_{i,j}^2$ provides the inter-row and inter-column permutation for $s = 3$ against row counter $i$ and column counter $j$. The expression for $r_{i,j}^2$ 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

$$I_{i,j} = \begin{cases} j, & \text{if } (i = 0), \\ I_{(i-1),j} + C + r_{i,j}^2, & \text{otherwise}. \end{cases} \quad (15)$$

Here parameter $s$ distinguishes different modulation schemes. For BPSK/QPSK $r_{i,j}^2 = 0$, and for 16-QAM and 64-QAM, $r_{i,j}^2$ and $r_{j,i}^2$ are given by (9) and (14), respectively. The hardware realization supporting all modulation schemes through a very small lookup table. The generic form for (6), complicated, but eventually, it gives a hardware efficient implementation as it involves only two additions, some registers, and a very small lookup table.

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 $i_s$ is given as follows:

$$R_k = \left[ J_k - \left\{ ((i_s - 1) \times 2)^{\text{mod} 3} + 3 \left\lfloor \frac{i_s - 1}{3} \right\rfloor \right\} \times N_{\text{ROT}} \times N_{\text{BPSC}} \right]^{\text{mod} N}. \quad (16)$$

Here $N_{\text{ROT}}$ 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 $i_s$, thus each spatial stream faces different frequency rotations. Defining the rotation term as $J_{\text{ROT}}$, that is,

$$J_{\text{ROT}} = \left\{ ((i_s - 1) \times 2)^{\text{mod} 3} + 3 \left\lfloor \frac{i_s - 1}{3} \right\rfloor \right\} \times N_{\text{ROT}} \times N_{\text{BPSC}}, \quad (17)$$

we have

$$R_k = (J_k - J_{\text{ROT}})^{\text{mod} N}. \quad (18)$$

The range for the term $(J_k - J_{\text{ROT}})^{\text{mod} N}$ is not bounded and it can have value greater than $2N$; thus direct implementation cannot be low cost. Analyzing the two terms $[J_k^{\text{mod} N}]$ and $(-J_{\text{ROT}})^{\text{mod} N}$ 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 $r_{ss} = (-J_{\text{ROT}})^{\text{mod} N}$ also holds for all run time computations. Equation (18) in combination with (10) can be written as

$$R_k^{i_s} = (J_{i,j} + r_{ss})^{\text{mod} N}. \quad (19)$$

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

\[
C = p - 1, \quad \text{if } (N \leq R \times (p - 1)),
\]
\[
C = p, \quad \text{if } (R \times (p - 1) < N \leq (R \times p)),
\]
\[
C = p + 1, \quad \text{if } (R \times p < N).
\]

(ii) Col Size:

\[
S(j) = \left( v \times S(j - 1) \right) \mod p; \quad j = 1, 2, \ldots, p - 2.
\]

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

\[
H = 2 \times H
\]

\[
H = 0; i = k - 1 \quad \text{Start}
\]

\[
H > P \quad \text{Yes} \quad H = H - P
\]

\[
H \leq P \quad \text{No} \quad \text{Finish}
\]

\[
v(i) = 1 \quad \text{Yes} \quad H = H + S(j - 1)
\]

\[
v(i) = 1 \quad \text{No} \quad H = H - P
\]

\[
i = i - 1 \quad \text{Yes} \quad i < 0 \quad \text{No}
\]

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

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

\[
r(i) = T(q(i)).
\]

(vi) Perform the intra-row permutations \( U_{ij}, j \), for \( i = 0, 1, \ldots, R - 1 \) and \( j = 0, 1, \ldots, p - 2 \).

If \( (C = p) \): \( U_{ij} = S((j \times r(i)) \mod (p - 1)) \) and \( U_i(p - 1) = 0 \).

If \( (C = p + 1) \): \( U_{ij} = S((j \times r(i)) \mod (p - 1)) \), and \( U_i(p - 1) = 0, U_i(p) = p \), and if \( (N = R \times C) \) then exchange \( U(R - 1, 0) \) with \( U(R - 1, p) \).

If \( (C = p - 1) \): \( U_{ij} = S((j \times r(i)) \mod (p - 1)) \mod (p - 1) - 1 \).

(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

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
using Interleaved Modulo Multiplication Algorithm [38]. It
divider, indirect computation of modulo function is done by
divider for direct implementation. To avoid the use of
the computation of a modulo function which requires a
are stored in a memory. These precomputations involve
T o resolve it, some precomputations are made and results
may require many clock cycles to be computed on-the-fly.

One of the main hurdles to generate on-the-fly inter-
leaved 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 ran-
dom 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 \( 52 \times 14b \). 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

\[
RA(i, j) = \{RA(i, j - 1) + q\text{mod}(i)\} \% (p - 1).
\]  

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 partic-
ular row. The final interleaved address \( I_{i,j} \) can be found

\[
I_{i,j} = RA(i\%C, j) + (i\%C)k,
\]  

where \( k \) is the number of times the \( j \)th row occurs in the
interleaved address.
by combining the inter-row permutation with intra-row permutation as follows:

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

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 inefficient 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 \( x = 0 \cdot \cdot \cdot N - 1, \) if \( (i\%2) = 1, then (A_i, B_i) = (B_i, A_i). \)

The new sequence is

\[ u_1 = [(A_0, B_0), (B_1, A_1), (A_2, B_2), \ldots, (B_{N-1}, A_{N-1})]. \tag{29} \]

**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 \( x = 0 \cdot \cdot \cdot N - 1 \)

\[ \text{switch } (x\%4). \]

\[ \text{case 0: } I_{(x\%4=0)} = (P_0 \cdot x + 1)\%N, \]

\[ \text{case 1: } I_{(x\%4=1)} = (P_0 \cdot x + 1 + N/2 + P_1)\%N, \]

\[ \text{case 2: } I_{(x\%4=2)} = (P_0 \cdot x + 1 + P_2)\%N, \]

\[ \text{case 3: } I_{(x\%4=3)} = (P_0 \cdot x + 1 + N/2 + P_3)\%N. \]

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

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

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

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

As range of \( \beta_x \) 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
Recently, convolutional interleavers have been analyzed to less memory utilization as compared to block interleaving. Applications with some added benefits of half the latency and branch synchronization purpose the sync bytes are always routed to B packet of 204 bytes consisting of one sync byte (0 \times 47 \text{ 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.
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 pre-computation 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

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Standard coverage</th>
<th>Technology</th>
<th>Operating frequency</th>
<th>Power</th>
<th>Memory size</th>
<th>Total core size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx [28]</td>
<td>General purpose</td>
<td>FPGA</td>
<td>262/360 MHz, Speed Grade -1/-3</td>
<td>—</td>
<td>18 Kbits</td>
<td>210 LUTs + Memory</td>
</tr>
<tr>
<td>Virtex-5</td>
<td>General purpose</td>
<td>FPGA</td>
<td>120 MHz</td>
<td>—</td>
<td>16 Kbits</td>
<td>392 LEs + Memory</td>
</tr>
<tr>
<td>Altera [29]</td>
<td>General purpose</td>
<td>FPGA</td>
<td>132 MHz</td>
<td>—</td>
<td>36 Kbits</td>
<td>284 LUTs + Memory</td>
</tr>
<tr>
<td>FLEX-10KE</td>
<td>General purpose</td>
<td>FPGA</td>
<td>WCDMA turbo code; cdma2000</td>
<td>—</td>
<td>35 Kbits</td>
<td>2.678 mm²</td>
</tr>
<tr>
<td>Lattice [30]</td>
<td>General purpose</td>
<td>FPGA</td>
<td>WCDMA, LTE, WiMAX and DVB-SH Turbo Code Interleaver Only</td>
<td>—</td>
<td>30 Kbits</td>
<td>0.084 mm²</td>
</tr>
<tr>
<td>Shin and Park [13]</td>
<td>General purpose</td>
<td>0.25 μm</td>
<td>—</td>
<td>—</td>
<td>35 Kbits</td>
<td>2.678 mm²</td>
</tr>
<tr>
<td>Asghar et al. [18]</td>
<td>WiMAX, LTE, DVB and DVB-SH Turbo Code Interleaver Only</td>
<td>65 nm</td>
<td>200 MHz</td>
<td>10.04 mW</td>
<td>30 Kbits</td>
<td>0.084 mm²</td>
</tr>
<tr>
<td>Chang and Ding [23]</td>
<td>WiMAX, WLAN, DVB</td>
<td>0.18 μm</td>
<td>100 MHz</td>
<td>—</td>
<td>12 Kbits</td>
<td>0.60 mm²</td>
</tr>
<tr>
<td>Chang [24]</td>
<td>WiMAX, WLAN, DVB</td>
<td>0.18 μm</td>
<td>150 MHz</td>
<td>—</td>
<td>32 Kbits</td>
<td>0.72 mm²</td>
</tr>
<tr>
<td>Wu et al. [25]</td>
<td>WiMAX, WLAN, 802.11n</td>
<td>0.18 μm</td>
<td>200 MHz</td>
<td>—</td>
<td>12 Kbits</td>
<td>0.484 mm²</td>
</tr>
<tr>
<td>Asghar and Liu [26]</td>
<td>WiMAX, WLAN, DVB</td>
<td>0.12 μm</td>
<td>140 MHz</td>
<td>3.5 mW</td>
<td>12 Kbits</td>
<td>0.18 mm²</td>
</tr>
<tr>
<td>Asghar and Liu [27]</td>
<td>WiMAX, WLAN, 802.11n</td>
<td>65 nm</td>
<td>225 MHz</td>
<td>4 mW</td>
<td>15.6 Kbits</td>
<td>0.035 mm²</td>
</tr>
<tr>
<td>Horvath et al. [20]</td>
<td>DVB bit and symbol interleaver</td>
<td>0.6 μm</td>
<td>36.57 MHz</td>
<td>300 mW</td>
<td>48 Kbits</td>
<td>69 mm²</td>
</tr>
<tr>
<td>Chang [21]</td>
<td>DVB bit and symbol interleaver</td>
<td>0.35 μm</td>
<td>—</td>
<td>—</td>
<td>52.2 Kbits</td>
<td>2.9 mm²</td>
</tr>
<tr>
<td><strong>This work</strong></td>
<td>All range including WLAN, WiMAX, DVB, 802.11n and General purpose implementation</td>
<td>65 nm</td>
<td>166 MHz</td>
<td>11.7 mW</td>
<td>72 Kbits</td>
<td>0.126 mm²</td>
</tr>
</tbody>
</table>
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 mm² 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 trade-off 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


Figure 17: Layout of proposed multimode interleaver.


