Memory Conflict Analysis and Interleaver Design for Parallel Turbo Decoding Supporting HSPA Evolution

Rizwan Asghar, Di Wu, Johan Eilert and Dake Liu

Department of Electrical Engineering, Linköping University, Sweden
Email: {rizwan, diwu, je, dake} @ isy.liu.se

Abstract

HSPA evolution has raised the throughput requirements for WCDMA based systems where turbo code has been adapted to perform the error correction. Many parallel turbo decoding architectures have recently been proposed to enhance the channel throughput but the interleaving algorithm used in WCDMA based systems does not freely allow to use them due to high percentage of memory conflicts. This paper provides a comprehensive analysis for reduction of interleaver memory conflicts while generating more than one address in a single clock cycle. It also provides trade-off analysis in terms of area and power efficiency for multiple architectures for different functions involved in the interleaver design. The final architecture supports processing of two parallel SISO blocks and manages the conflicts by applying different approaches like stream misalignment, memory division and small FIFO buffer. The proposed architecture is low cost and consumes 4.3K gates at a frequency of 150 MHz. This work also focuses on reduction of pre-processing overheads by introducing the segment based modulo computation, thus providing further relaxation to SISO decoding process.

Keywords: Parallel interleaver, Block interleaver, Parallel turbo decoding, HSPA, UMTS, WCDMA.

1. Introduction

Quest for bandwidth is accelerating competition among wireless technologies. In order to enhance the bandwidth requirements, 3GPP-WCDMA has released a series of specifications from time to time. The advancements are mainly based on high speed packet access (HSPA) using more downlink/uplink channels and addition of modulation schemes. In this connection, Release 5 [1] has provided an upgrade to high speed downlink packet access (HSDPA) to support more efficient and bandwidth-intensive data applications. It enhanced the theoretical limit of data throughput up to 14Mbps. The evolution of HSPA (HSPA+) is carrying forward and after addition of MIMO and 64 QAM in Release 8 [2] the throughput requirements have reached to 43.2 Mbps where adaptive modulation with increased coding efficiency is one of the important techniques in improving the throughput. The channel coding involves turbo coding [3] with rate-one-third and a complicated interleaver algorithm. The scheme of turbo code adapted is the parallel concatenated code with two 8-state constituent encoders and an internal interleaver. One turbo code internal interleaver is used in between the two recursive systematic convolutional (RSC) encoders as shown in Fig. 1(a), while the turbo decoder uses multiple instances of interleaver and de-interleaver as shown in Fig. 1(b) to decode the received bits iteratively.

The technique of parallel SISO block processing is usually used to achieve higher throughput in channel coding and it is well established in the literature [4]–[7]. To do the parallel SISO processing parallel interleaver support is needed which is relatively easy for some types of interleavers, but the interleaver type used in HSPA+ is not targeted inherently to support the parallel processing. The large percentage of memory
conflicts is the hurdle to find a hardware efficient solution when parallel address generation is required. This paper explores the possibilities for the HSPA+ interleaver to be used to generate parallel addresses in real time and thus enabling parallel SISO processing. Section 2 of this paper provides the background and challenges involved in parallel interleaver address generation whereas section 3 provides explanation for parallel address generation along with the conflict analysis. Section 4 provides different architectures required for pre-processing and trade-offs to reduce the pre-processing overheads. Section 5 provides and details of final hardware architecture for parallel interleaver. Implementation results are discussed in section 6 followed by a conclusion.

2. Background and Challenges

The turbo code is widely being used in different communication systems. The superior error correcting performance of turbo codes over convolutional codes is associated with the suitable interleaver design. The prime interleaver being used in HSPA+ with turbo code has good distance properties and very helpful to minimize the effect of burst errors introduced in transmission. Looking at the implementation aspects the convolutional encoders are very simple to implement as compared to SISO decoding in turbo decoder, but the interleaver tends to be complex on both sides because of its variability. The big range of block sizes i.e. 40 to 5114 is needed to be supported and it becomes more complex as the block size can vary in every transmission time interval (TTI). The flow diagram for the implementation of HSPA+ interleaver is shown in the Fig. 2. Due to the complexity of frequent changes in the block size it requires some pre-processing with every change as prescribed in [11] – [15]. This pre-processing is associated with some extra cycle cost before starting the actual interleaving. For high throughput requirements and small TTI the reduced overhead cost is always beneficial.

On the other hand the throughput enhancements can be supported by parallel SISO processing which requires parallel address generation from interleaver to write the data to memory in parallel. The parallel address generation from the interleaver algorithm is usually required to be conflict free, which is not the case with the prime interleaver being used in HSPA+. The conflict percentage from the two parallel generated addresses is very high as shown in Fig. 4, and there is no straight forward solution except using double memory size which is not cost efficient. This becomes a bottleneck while considering parallel SISO processing in HSPA+ data processing. The work presented in [11] – [15] covers single address generation and usually parallel address generation with this particular interleaver has been avoided due to high memory conflicts. Work in [7] - [9] provide good theoretical back ground and also propose the generation of conflict free interleavers but they cannot be directly used for already existing interleaver algorithm for HSPA+. A very good analysis of memory conflicts for interleaver in turbo decoder is provided in [10] and the stalling mechanism is used to stall the process in MAP decoders when a conflict occurs. The main focus of this work has been the buffer management and lacking the architectural aspects associated with address generation itself. Further the stall process may result in additional control complexity. A low cost solution for parallel interleaver implementation for existing prime interleaver algorithm in HSPA+ can open more opportunities to meet the high throughput requirements associated with HSPA evolution.

3. Parallel Address Generation and Memory Conflict Analysis

The interleaving algorithm for 3GPP-WCDMA is mentioned below. Here $N$ is the block size, $R$ is the row size and $C$ is the column size in bits.

- Find appropriate number of rows 'R'; prime number 'p' and primitive root 'v' for particular block size as given in the standard.
- $\text{Col Size} : \begin{cases} C = p-1; & \text{if } (N \leq R \times (p-1)) \\ C = p; & \text{if } (R \times (p-1) < N \leq R \times p) \\ C = p+1; & \text{if } (R \times p < N) \end{cases}$
Construct intra row permutation sequence $S(j)$ by:

$$S(j) = \lfloor v \times S(j-1) \rfloor \mod p; \quad j = 1, 2, \ldots, p-2$$

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

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

$$r(i) = T(q(i))$$

Perform the intra row permutations $U_{i,j}$ such that for $i = 0, 1, \ldots, R-1$ and $j = 0, 1, \ldots, p-2$:

- If $(C = p)$: $U_{i,j} = S \lfloor (j \times r(i)) \mod (p-1) \rfloor$ and $U_{i,(p-1)} = 0$
- If $(C = p+1)$: $U_{i,j} = S \lfloor (j \times r(i)) \mod (p-1) \rfloor$, 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 $(C = p-1)$: $U_{i,j} = S \lfloor (j \times r(i)) \mod (p-1) \rfloor - 1$

Perform the inter row permutations.

Read the address columns wise.

The complication in the implementation is evident due to the presence of complex functions like modulo computation, intra-row and inter-row permutations, multiplications, finding least prime integers, and computing greatest common divisor. After preprocessing and applying the intra-row and inter-row permutations a block of random addresses denoted by $y_i$ appears as shown below:

$$\begin{bmatrix}
y_1 & y_{(R+1)} & \cdots & y_{(C-1)(R+1)} \\
y_2 & y_{(R+2)} & \cdots & y_{(C-1)(R+2)} \\
\vdots & \vdots & \ddots & \vdots \\
y_R & y_{2R} & \cdots & y_{(C-1)R}
\end{bmatrix}$$

The output from the interleaver is the sequence read column by column from the permuted matrix. The output address $y_k$ within the matrix can be expressed as:

$$\text{loop } i : 1 \text{ to } C$$

$$\text{loop } j : 1 \text{ to } R$$

$$Y_k = y_{(j-1)R+i}$$

Generating two addresses $Y_k^1$ and $Y_k^2$ at the same time will reduce the overall loop size to half, as given below:

$$\text{loop } i : 1 \text{ to } \frac{C}{2}$$

$$\text{loop } j : 1 \text{ to } R$$

$$Y_k^1 = y_{(j-1)R+i}$$

$$Y_k^2 = y_{\left(\frac{(j-1)R+i}{2}\right)}$$

The two addresses generated simultaneously are used to write two data values in two memory locations. If these two addresses map to different memories i.e. one address for each memory then there is no conflict, but on the other hand if the two addresses are mapped to same memory then a situation of conflict occurs (Fig. 3) and it needs to be resolved on-the-fly. For the case of HSPA+ interleaver the number of conflicts reaches to 50% of the data size as shown in Fig. 4 (top sub-plots). One way to reduce the number of conflicts is to misalign the generation of two addresses by some arbitrary value. The misalignment can be achieved by introducing a delay line to one set of addresses and data values as shown in Fig. 5. The misalignment introduces the latency on one of the two sequences, thus the misalignment factor (MF) cannot be very long. There are three possible values for $R$ (number of rows) in the prescribed interleaver i.e. 5, 10 or 20 and good
MF must be within the total number of rows. It is observed that the inter-row permutation patterns for \( R=5 \) or \( 10 \) are supportive to misalignment technique but the permutation patterns for \( R=20 \) are not very much supportive. The best MF found is 3 for \( R=5 \) or \( R=10 \) and 5 for \( R=20 \). Using these values of MF the number of conflicts has reduced to around 10% as shown in Fig. 4 (bottom sub-plots). Still the conflict count is high enough that it cannot be managed without the support of extra memories. Alternately, a large amount of buffer registers can be used but it involves introducing higher latency as well.

Another approach might be to split the memory banks into relatively smaller sub-memories to reduce the number of conflicts. Further motivation to do this is the size of each memory (i.e. 2557) while using two banks. Usually memories are used having size in the power of 2. We divide the memory into 3 sub-memories (1k, 1k and 512) in order to have size in the power of 2 and at the same time support the conflict reduction. By applying this configuration to the interleaver address generation and data writing the number of conflicts for most of the block sizes reduced to zero (Fig. 6), but still many block sizes face some conflicts. The amount of conflicts in this configuration is very small and it can be handled by using buffer registers. The number of FIFO buffers can further be reduced by applying the progressive writes during the execution to the other memory bank. The total number of FIFO buffers needed for the two memory banks to cover full range of block sizes is plotted in Fig. 7. The buffer size for memory bank M2 is only one and most of the buffer size is associated with memory bank M1. It can also be noted that the maximum buffer size requirement which is 39 is only for a selective range of block sizes. If this particular range of block sizes can be avoided by zero padding during the execution then practically only one buffer register is needed to cover all the range. The cost comparison for different memory organizations is provided in Table 1. The increase in area and power is small while selecting splitted memory scheme over two memory bank scheme, whereas, it gives significant reduction in addressing complexity.

### 4. Pre-Processing

The only parameter passed to the interleaver is the block size \( N \). Therefore, in order to make the interleaver architecture fully autonomous, the 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 \( p \) and associated integer \( v \) are needed to be computed in hardware. Some of these parameters can be computed using lookup tables while the others need some close loop or recursive computations.

The most critical parameter which takes more clock cycles and more hardware to compute is the intra-row permutation pattern \( S(j) \). The function to be computed to find all the permutations is given by:

\[
S(j) = \{v \times S(j-1)\} \mod p \quad j = 1, 2, ..., p - 2
\]

Computation of the parameter \( S(j) \) during the execution is not hardware efficient as well as it exhibits

![Fig. 5. Misalignment of Address and Data to reduce memory conflicts.](image1)

![Fig. 5. Misalignment of Address and Data to reduce memory conflicts.](image2)

![Fig. 6. Reduction of conflicts by dividing the memories.](image3)

<table>
<thead>
<tr>
<th>Memory Configuration</th>
<th>Memory Size</th>
<th>Area (µm²)</th>
<th>Power (µW/MHz)</th>
<th>Addressing Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Memory</td>
<td>1×5K</td>
<td>28020</td>
<td>9.21</td>
<td>Very High</td>
</tr>
<tr>
<td>Two Memories</td>
<td>2×2K, 2×512B</td>
<td>38166</td>
<td>15.12</td>
<td>Very High</td>
</tr>
<tr>
<td>Splitted Memories</td>
<td>4×1K, 2×512B</td>
<td>44856</td>
<td>17.32</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 1. Cost Comparison for Diff. Memory Configurations.
unknown clock cycle delay due to the reason that the value \( S(j-1) \) is not known with each \( j \). It is computed recursively to achieve low cost solution and the results are placed in a small RAM called intra-row permutation RAM (IRP_RAM) to be used during execution. One way of computing this parameter is using the Interleaved Modulo Multiplication Algorithm [16] which needs more than one clock cycles to compute one value. The hardware to compute \( S(j) \) using modulo multiplication algorithm is shown in Fig. 8(a). Looking at the size of \( v \) which is 5 bits at maximum this algorithm can take 5 clock cycles to compute each \( S(j) \) recursively.

We propose here segmentation based modulo computation (SBMC) as an alternative approach to compute modulo function. The idea is to use only addition functions in series and every addition function should be followed by a modulo addition. This method might not be hardware efficient when the multiplication terms are wide range, however, it is beneficial if any one of the parameter in the multiplication term is relatively smaller in range. Here parameter \( v \) is having limited number of values i.e. 2,3,5,6,7 or 19. Thus SBMC can be optimized to achieve low cost solution. The proposed hardware for computing the intra-row permutation using SBMC is shown in Fig. 8(b). The main advantage of using this scheme over Modulo Multiplication Algorithm is the single clock cycle execution for finding each new \( S(j) \), thus it can significantly reduce the pre-computation overheads. While supporting the generation of two interleaved addresses per clock cycle, two parallel IRP_RAMs each having size 256 x 8b are required to store the intra-row permutation patterns. Keeping in view the hardware in-efficiency only one is used here with size 128x16b and the 16 bit data output is further divided to be used for two different sections of address generation. The second address with which the data has to be merged is computed by:

\[
j_{aux} = \left( j + \frac{C}{2} \right) \mod p
\]

The parameters \( j \) and \( j_{aux} \) are compared and smaller one is used to write the data. The same expression and comparison is used to resolve the data distribution for generation of two addresses per clock cycle. The other parameters to be computed in the pre-processing phase are prime number \( p \), associated integer \( v \) and total number of columns \( C \). The parameter \( p \) is stored in lookup table with a smaller sub-lookup table for parameter \( v \). The lookup table is addressed via a counter and against each value of \( p \) the condition \( (p \times R \geq N - R) \) is tested using a comparator, to find the appropriate values. Once \( p \) is found, the total number of columns \( C \) can have only three values i.e. \( p - 1 \), \( p \) or \( p + 1 \), thus requiring at most three clock cycles to find \( C \) by checking the condition \( (R \times C \geq N) \). The hardware required for computing the parameters during the pre-processing phase remains idle during the execution phase, therefore it can be reused to reach to a low cost solution.

5. Hardware for Parallel Address Generation

The parallel processing of turbo decoder with two SISO blocks requires the interleaver to generate two addresses every clock cycle and at the same time resolve the memory conflicts. One may think of
replicating the hardware two times to handle the processing requirements of two parallel SISO blocks, but it is not the optimal way. The computation of final interleaved address requires that the intra-row permutation data computed during the pre-processing must be obtained in correct order from IRP_RAM. The data output from the register file is denoted as \( U(i, j) \) and it is given by:

\[
U(i, j) = S[j \times r(i) \% (p - 1)]
\]  
(2)

The address \( RA \) to the IRP_RAM given by \( j \times r(i) \% (p - 1) \) involves modulo function which is not hardware efficient if it is implemented through division. We present here three alternates along with a comparison to select a hardware efficient approach.

The first method involves recursive computation of addresses as shown in Fig. 9. The data is written into memory row wise but while reading back column wise, the IRP_RAM address of previous column is needed to find next address. The function used to compute the address ‘\( RA \)’ recursively is given by:

\[
RA(i, j) = [RA(i, j - 1) + qmod(i)] \% (p - 1)
\]  
(3)

The parameter \( qmod(i) \) is computed from the least prime numbers sequence \( q(i) \) with the modulo operation i.e. \( qmod(i) = q(i) \% (p - 1) \). The condition applied on eq. (3) to be computed correctly with low cost is \( q(i) < 2(p - 1) \). The similarities between different sequences for \( q(i) \% (p - 1) \) for all possible \( p \) values are exploited to improve the efficiency of lookup table for \( q(i) \). The computational part associated with recursive approach is very small and limited to just few adders, but it needs a circular buffer of size 20 x 8b in order to keep the old address of whole column. The hardware required for computation of address ‘\( RA \)’ using recursive approach is shown in Fig. 10(a). This approach is very low cost and can operate at very high clock rate due to smaller critical path. If we use the same hardware for computing the intra-row permutation patterns, five clock cycles are required for each value of \( S(j) \). To reduce it to single clock cycle we need to use the mix of the recursive approach and the segmentation based modulo computation. This second approach is not very much hardware efficient (see Fig. 10(b)) but gives the benefit to reduce the pre-processing time.

The third approach is completely based on segmentation based modulo computation and it is a non-recursive way to compute the address \( RA \). It can directly be applied to the term \( j \times r(i) \% (p - 1) \) to get the address for register file. As discussed earlier, this approach can be efficient if any one of the parameter in the multiplication term is relatively smaller in range.
Here we know that the parameter $r(i)$ can have only 22 values of prime numbers up to 89, thus SBMC approach can be used after applying some optimizations. The hardware required to compute the function $j \times r(i) \% (p-1)$ using SBMC approach is shown in Fig. 10(c). It needs more additions then the recursive approach but it provides a good tradeoff between area, power and pre-processing cycle cost (Table 2). It can also be directly used to compute intra-permutation patterns in the pre-processing phase, thus providing single clock cycle support for computing each permutation pattern $S(j)$. This approach cannot be better for very high clock frequency due longer critical path, but still it can be a good candidate for this application due to two reasons: 1) The maximum clock frequency supported by SBMC approach is currently sufficient to meet the throughput requirements for HSPA evolution; 2) Pipelining can always be introduced to enhance the operating speed of this approach.

The computed address is used to get the correct intra-row permutation $U(i,j)$ from the RAM. It also needs to pass through some exception handling logic to correct itself for special cases associated with $C = p$, $C = p + 1$ or $C = p - 1$ as given in the algorithm. The final interleaved addresses $Y_{1,j}$ and $Y_{2,j}$ can be found by combining the inter-row permutation with intra-row permutation as follows:

$$Y_{1,j} = \{C \times r(i)\} \div U_1(i,j) \quad (4)$$

$$Y_{2,j} = \{C \times r(i)\} \div U_2(i,j) \quad (5)$$

The complete hardware for the generation of parallel interleaved addresses and data handling is shown in Fig. 11. A FIFO buffer of size 39 is used as discussed in Section 3 to handle the conflicts occurring in one of the memory bank. The generation of interleaved address during run time is one address per clock cycle, except the case when block size is not exactly equal to $R \times C$. In this case data written into the RAM is zero padded and later on data pruning is performed using the comparators.

6. Implementation Results

The architecture for parallel interleaver address generation (Fig. 11) is modeled using Verilog HDL and synthesized with ST Microelectronics flow for 65nm CMOS Technology. The design is synthesized to operate at a frequency of 150 MHz and the area comparison with other implementations is provided in Table 3. The comparison shows that the proposed architecture is low cost with added benefit of parallel SISO support. The presented design consumes 1.4 mW power in total (@ 150MHz). The power consumed by the address generation logic including FIFO is 550 µW and the rest is consumed by the 128x16b RAM. The reported designs provide the area and operating frequency but unfortunately no power figures are available by now to compare with the architecture discussed here.

The cost associated with the pre-processing is also reduced significantly. The cycle cost comparison for different block sizes is provided in Table 4. The cycle cost in the reference designs is mainly associated with
parameter $p$ and $v$ and computation of each $S(j)$ value requires more than one clock cycles (i.e. depending on $v$). However, the SBMC approach used in this work provided a low cost alternative which requires one clock cycle for each $S(j)$ value.

7. Conclusion

This paper presented a low cost parallel interleaver address generation architecture to support the parallel turbo decoding for HSPA evolution. The complexity of the parallel interleaver address generation and high memory conflicts has been a bottleneck to enhance the throughput using parallel SISO processing. Different techniques are used to deal with the memory conflicts and hence the parallel SISO processing is enabled with the interleaver algorithm used for HSPA+. The processing of SISO decoding is further relaxed by achieving low cycle cost overheads during pre-processing phase. The worst case cycle cost is 310 which can easily be afforded.

References