Very Low Cost Configurable Hardware Interleaver for 3G Turbo Decoding

Rizwan Asghar and Dake Liu
Department of Electrical Engineering, Linkoping University
Linkoping, Sweden
{rizwan, dake}@isy.liu.se

Abstract—A very low cost hardware interleaver for 3rd Generation Partnership Project (3GPP) turbo coding algorithm is presented. The interleaver is a key component of turbo codes and it is used to minimize the effect of burst errors in the transmission. Using conventional design methods, it consumes a large part of silicon area in the design of turbo encoder and decoder. The presented hardware interleaver architecture utilizes the algorithmic level hardware simplifications as well as the iterative modulo computation to achieve very low cost solution. After doing the hardware multiplexing and optimization the proposed architecture consumes only 1.5 k gates (without pre-computation) and 2.2 k gates (with pre-computation). In both cases the interleaved address is computed every clock cycle except the case of pruning, in which one additional clock cycle is consumed.

Keywords—hardware interleaver; WCDMA; turbo codes; vlsi architecture; hardware multiplexing;

I. INTRODUCTION

WIDEBAND CDMA systems in 3G standard use turbo codes for forward error correction (FEC). The turbo codes [1] invented in 1993 captured great importance due to exhibiting near Shannon-limit performance. The superior error correcting performance of turbo codes over convolutional codes is associated with the suitable interleaver design. The primary function of the interleaver is to improve the distance properties of the concatenated 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. Error correcting codes like turbo code can correct errors successfully as long as there are not too many errors in a single code word.

The turbo code has been adapted in 3rd Generation Partnership Project (3GPP) for W-CDMA systems [2], due to which turbo code as well as interleaver design for turbo codes captured more attention of the researchers. There are two classical kinds of interleavers, commonly known as block interleaver and convolutional interleaver. In a block interleaver the data is written row wise in a memory, configured as a matrix and read out column wise. In a convolutional interleaver, the data is multiplexed and implemented as a fixed number of shift registers. The turbo encoder uses the interleaver in-between two or more recursive systematic convolutional encoders (RSC) as shown in Fig. 1a while the turbo decoder uses multiple instances of interleaver and de-interleaver to decode the received bits iteratively. The structure for turbo decoding is shown in Fig. 1b.

The motivation of the research is to explore the low cost hardware interleaver for 3G, as well as to try to find a general and re-configurable architecture for different kinds of interleavers or de-interleavers of different radio communication systems.

In this paper, Section II presents the challenges and previous work done for the interleaver algorithm for W-CDMA systems. Section III provides detailed implementation of pre-computation steps. The working and implementation of controller and run time hardware for 3G interleaver is elaborated in sections IV and V. The implementation results along with conclusion are discussed in Section IV and VII.

II. COMPUTATIONAL CHALLENGES AND PREVIOUS WORK

Reference to 3G standard [2], the interleaver algorithm for turbo coding and decoding can be summarized as below. Here K is the block size, R is the row size and C is the column size in bits.

- Block Size: \( 40 \leq K \leq 5114 \)
- Row Size: \( R = 5 \) if \((40 \leq K \leq 159)\)  
  \( R = 10 \) if \((160 \leq K \leq 200)\) or \((481 \geq K \leq 530)\)  
  \( R = 20 \) if \((K =\text{ any other value})\)
The address generation algorithm for the interleaving is described as below:

- Find appropriate prime no ‘p’ and primitive root ‘v’
- Col Size : C = p-1 if (K ≤ R x (p-1)) 
  C = p if (R x (p-1) < K ≤ R x p) 
  C = p+1 if (R x p < K)
- Construct intra row permutation sequence S(j) by:
  \[ S(j) = \{ v \cdot S(j-1) \mod p \}; \quad j = 1, 2, \ldots, p-2 \]
- Find q(i) for i=1,2, …. R-1, take q(0) = 1. Determine the least prime integer q(i) in the sequence such that
  \[ \text{g.c.d}(q(i),p-1) = 1 \text{ and } q(i) > 6 \text{ and } q(i) > q(i-1) \]
- Apply inter row permutations to q(i) to find ri
  \[ r_i = T(q(i)) \]
- Perform the intra row permutations \( U_{i,j} \); (for i = 0,1, …… R-1; j = 0,1, ….p-2,)
  - If (C=p) : \( U_{i,j} = S \{ j \cdot r_i \mod (p-1) \} \); and \( U_{i,0} = 0 \);
  - If (C=p+1) : \( U_{i,j} = S \{ j \cdot r_i \mod (p-1) \} \); and \( U_{i,0} = 0 \);
  - and \( U_{i,p} = p \); and if (K = R x C) then exchange U(R-1,0) with U(R-1,p)
  \[ U_{i,j} = S \{ j \cdot r_i \mod (p-1) \} - 1; \]
- Perform the inter row permutations
- Read the address columns wise

Looking at the 3G standard [2], the wide range interleaver size, on the fly size change and requirement of different interleaver patterns for different block sizes has put some challenges to implement the interleaver. Implementing the memory based interleaver needs very big memory space to meet the whole requirements. Use of such a big memory is not feasible due to large hardware cost. To handle this situation researchers have proposed some architecture in [3] and [4]. The total hardware consumed in designs [3] and [4] is 32 k gates and 30 k gates respectively. The recent research [5] proposed an efficient hardware interleaver which costs 4 k gates. Our initial goal was to have the interleaver without pre-computation, as one processor is already available, while looking at system level. The pre-computation part was also done just to have the fair comparison with the existing designs. Our proposed architecture has even less hardware consumption after applying certain optimizations. The total hardware cost as compared to the above mentioned designs is 2.2 k gates (excluding 256x8 RAM). However, if the pre-computation is not done, that is, if there is some other processor available to do this job, the total hardware cost is more reduced to 1.5 k gates (excluding 256x8 RAM).

After profiling the behavioral source code, the following computation intensive operations are needed to fulfill the requirements:

- Modulo Functions
- Permutations (intra row and inter row)
- Multiplier
- Memory Usage
- Finding least prime integers
- Computing g.c.d.(greatest common divisor)
- Exception Handling

Some of these complex functions are implemented using the support from ROM, while the others are simplified to reduce the hardware usage.

III. PRE-COMPUTATION

The pre-computation in the hardware facilitates the quick change of the interleaver block size on the fly. As different block sizes have different interleaving patterns, thus every time some pre-computation is needed while changing the block size. The flow graph for the pre-computation phase is shown in Fig 4a. The only parameter passed to interleaver is the block size and rest of the parameters are computed by the hardware itself. The parameters computed in the pre-computation phase are number of rows (R), no of columns (C), least prime number sequence (q(i)), inter-row permutation pattern (T(i)), prime number (p) and associated integer (v). Additionally the intra-row permutation pattern (Si(j)) is also computed in the pre-computation phase and placed in a RAM. To compute number of rows (R) and inter-row permutation pattern (T), the logical functions serving as a lookup table are developed. These logical functions consume less hardware as compared to lookup table based implementations.

For finding the least prime number sequence (q), we focused on finding q mod (p-1) instead of finding q. This gives the benefit of computing the RAM address recursively and avoiding computation of modulo function. This idea was introduced in [3] and later [5] also used this. The benefit of computing q mod (p-1) instead of q and recursive computation of RAM address will be clear in the next sections. For
computing $q \mod (p-1)$, the following observations were made during simulations.

- About half of the $q(i)$ patterns are same
- Others can have 1 difference in the sequence
- Only one pattern has 2 differences

We managed all the $q$ values, such that $q(i) < 2(p-1)$ for all possible $q$ sequences. Using the above mentioned observations different $q$ sequences were divided into subgroups for different $p$ values, which reduced the total ROM requirement for these values. Logical functions were also computed to do this but it is very much tedious to implement those due to there size, so the grouped sequences are placed in ROM. After getting the $p$ value the subgroup is computed through some logical functions and then ROM location is accessed using the inter-row permutation pattern in run time to find the right $q$ sequence. After getting the value a small hardware shown in Fig. 2a, is used to get the required $q \mod (p-1)$ value. This hardware is multiplexed such that it can be used in pre-computation phase to compute the intra-row permutation sequence $S(j)$ as well.

![Fig. 3: Hardware to Compute $S(j)$ and RAM Address.](image)

The parameters $p$ and $v$ are stored in combined fashion in a lookup table. The lookup table is addressed via a counter generated by the controller. With each increment in the counter, the condition $(P*R \geq K-R)$ is checked to find the appropriate $p$ and $v$. When the condition is satisfied, increment in counter is stopped and loop exits. Once $p$ is found, $C$ can have only three values i.e. $p-1$, $p$ or $p+1$. Hence $C$ is found in at most three clock cycles after finding $p$. In case $C = p+1$, the condition that $R*C = K$ is checked and appropriate flag for special case is also set to be used as an exceptional case. For computation of $p$ and $C$, the multiplication, addition and comparison are needed. These blocks are shown in Fig. 2b. These blocks are multiplexed to serve in pre-computation phase as well as in run time.

![Fig. 4: Flow Graph for (a) Pre-Computation Phase, (b) Run Time Computations.](image)

The computation of intra-row permutation pattern also need modulo computation. Here the modulo function is computed iteratively using the Interleaved Modulo Multiplication Algorithm [6]. The required modulo function is $[S(j-1)*v \mod P]$, so looking at $v$, which is 5bits, maximum of 5 iterations are needed to compute one modulo multiplication. The algorithm to compute the Interleaved Modulo Multiplications is shown in Fig. 2c. Three adders are needed to compute the $S(j)$ values. This can further be multiplexed and one adder may be used but as all these three adders are being used by our design in run time, no need to further multiplex in pre-computation phase. Here $H$ register is used to hold the old values during iterations.

![Fig. 3: Hardware to Compute $S(j)$ and RAM Address.](image)

The hardware for the computation of $S(j)$ is shown in Fig. 3. Hardware multiplexing is done in such a way that the first adder is also used to compute $q \mod (p-1)$ as shown in Fig. 2a, and the other two adders are used in the run time to compute the RAM address recursively. During pre-computation phase the ram write address is generated by the controller. The output ‘Modulo Out’ is used as the data out for RAM during pre-computation phase, while serves as the ram read address during run time. After computing the $S(j)$ values, these are placed in a 256*8bit RAM. The usage of this RAM depends
on parameter p and it will be filled up to (p-2) locations starting from zero. The requirement of 5 iterations may further be reduced in certain cases as v can have value 2, 3, 5, 6, 7 or 19 only. For the case v=19, we need 5 iterations, for v=5, 6 and 7 we need 3 iterations whereas the number of required iterations for v=2 and 3 is 2. This feature is used to reduce the overall clock cycles required for pre-computation of S(j).

IV. CONTROLLER DESIGN

The controller plays an important role for the pre-computation phase. The controller configures the multiply, add and compare hardware shown in Fig. 2b, and modulo computation hardware as shown in Fig. 2c to compute all the vital parameter R, C, p, v and S(j). The state diagram of the controller for our design is shown in Fig. 5. When the controller is in run mode, new pre-computation may be initiated by changing the block size parameter K and providing one pulse named ‘change_block_size’.

![Fig 5: Controller for 3G Hardware Interleaver.](image)

V. RUN TIME HARDWARE

After completing the pre-computation phase, the controller is set in ‘RUN’ state and the hardware is configured to perform run time computations. The RAM Address is computed using the hardware shown in Fig. 3. The flow graph for 3G interleaver in run time is shown in Fig. 4b and complete hardware for the interleaver is shown in Fig. 6. A circular buffer is also used to support recursive computation of RAM Address. As mentioned before, the parameter Q_mod is computed using the first adder in the modulo computation hardware shown in Fig. 3.

\[ Q_{\text{mod}}(i) = q(i) \mod (p-1) \]

The RAM Address is computed by computing the following recursive function.

\[ \text{Ram}_\text{Adr}(i,j) = \left[ \text{Ram}_\text{Adr}(i, j-1) + Q_{\text{mod}}(i) \right] \mod (p-1) \]

It can be seen that computing the RAM address using Q_mod instead of q helps to avoid the full computation of modulo multiplication. After computation of RAM Address the final interleaved address is computed by the following multiply-add function.

\[ i_{\text{addr}} = (C \times \text{Row}_\text{Perm}) + U(i,j) \]

The hardware used in pre-computation, as shown in Fig. 2b is used again to compute the final interleaved address in run time. The final address is tagged valid or invalid using the comparator. This is called pruning of the interleaver and is needed for the case when interleaver block size is not exactly equal to R*C. The hardware components in the design are used very efficiently such that the hardware used for pre-computation phase is also been used for the run time computations.

There exist some exceptions as well in the algorithm which are listed below:

- If \((C=p)\) : \(U(i,p-1) = 0\);
- If \((C=p+1)\) : \(U(i,p-1) = 0; U(i,p) = p; \) and
- if \((K = R \times C)\) then exchange \(U(R-1,0)\) with \(U(R-1,p)\).

Some flags related to first and last row and column are generated from row and column counters. Using these flags along with a couple of multiplexers a very small logic is needed to serve the purpose of exception handling.

![Fig 6: Complete Hardware for 3G Interleaver.](image)
### TABLE 1. PRE-COMPUTATION AND CLOCK CYCLE COMPARISON FOR DIFFERENT BLOCK SIZES

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Pre-Computation Cycles</th>
<th>Run Time Computation Cycles / frame</th>
<th>Average Bit / cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>15</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>23</td>
<td>50</td>
<td>0.8</td>
</tr>
<tr>
<td>500</td>
<td>117</td>
<td>530</td>
<td>0.943</td>
</tr>
<tr>
<td>3840</td>
<td>988</td>
<td>3840</td>
<td>1</td>
</tr>
<tr>
<td>3841</td>
<td>615</td>
<td>3860</td>
<td>0.995</td>
</tr>
<tr>
<td>5040</td>
<td>802</td>
<td>5040</td>
<td>1</td>
</tr>
<tr>
<td>5114</td>
<td>563</td>
<td>5120</td>
<td>0.9988</td>
</tr>
</tbody>
</table>

### TABLE 2. HARDWARE USAGE COMPARISON FOR DIFFERENT INTERLEAVER IMPLEMENTATIONS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Implementation</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>ROM (all possible patterns)</td>
<td>&gt; 100 Mbit</td>
</tr>
<tr>
<td>2.</td>
<td>RAM (big off-chip Mem. required)</td>
<td>~ 66.5 Kbit</td>
</tr>
<tr>
<td>6.</td>
<td>Our Design Excluding 2Kb RAM (with pre-computation)</td>
<td>~ 2.2K gates</td>
</tr>
<tr>
<td>7.</td>
<td>Our Design Excluding 2Kb RAM (without pre-computation)</td>
<td>~ 1.5K gates</td>
</tr>
</tbody>
</table>

### VI. IMPLEMENTATION RESULTS

The RTL code for the hardware blocks is written in Verilog and the correctness of the design is checked by comparing the results generated by hardware with those generated using MATLAB. The number of clock cycles spent for pre-computation (verified through simulations) is mentioned in Table 1. These cycles mainly depend on the value of p and v for a particular block size. It can be seen that maximum pre-computation cycles are 988 when block size is 3840 (p=191, v=19). The computation of interleaved address during run time is every cycle, except the case when block size is not exactly equal to R*C. In this case the interleaved address is computed at most every two clock cycles, however on the average the interleaved address is computed almost every clock cycle as shown in Table 1.

The design is synthesized after setting constraint as area using 90nm standard CMOS technology. Excluding the 256x8 bit RAM, the total hardware consumed by our design is 2.2 k gates, which is better than or almost same as that of the reference designs as shown in Table 2. We also synthesized the design without pre-computation, which consumed 1.5 k gates with 256x8 bit RAM.

### VII. CONCLUSION

In this paper a very low cost design for hardware interleaver for 3G turbo decoding with minimal latency is presented. The final results show that it is the smallest design; however, more optimizations can be performed in order to reduce the hardware cost. Our principle goal is to investigate the hardware interleaver for multiple standards. In this regard, the interleaver design presented here can serve as a vital building block and this research can provide a basis towards general and re-configurable design for different radio communication systems.

### REFERENCES


