A Monolithic LTE Interleaver Generator for highly parallel SMAP Decoders

Thomas Ilnseher, Matthias May, Norbert Wehn
Microelectronic Systems Design Research Group, University of Kaiserslautern
67663 Kaiserslautern, Germany
{ilnseher, may, wehn}@eit.uni-kl.de

***Abstract*—The LTE standard specifies a throughput of 150MBit/s, while the upcoming true 4G LTE Advanced standard will push this throughput to 1 GBit/s. To achieve this throughput while fulfilling the low power requirements of mobile devices, future receiver circuit architectures need to deploy a high internal parallelism. The LTE standard has been designed with this parallelism in mind, using a QPP interleaver inside the turbo code decoder. In this paper we present a novel method to implement the QPP interleaver which significantly reduces power and area of this circuit.**

I. Introduction

The demand for a seamless mobile multimedia experience drives the demand for bandwidth to new heights. To fulfill this demand, new technologies like the Long Term Evolution of 3GPP standards (LTE) [1] have been developed. They are tailored towards delivering very high bandwidth. The current 3.9G LTE standard can deliver up to 150MBit/s, with extensions to 300MBit/s already specified.

To become a true 4G standard, LTE advanced needs to adopt a maximum throughput of 1 GBit/s. Therefore, high speed receiver circuits need to be developed. A key component in such a receiver circuit is the turbo decoder [2] for channel decoding.

For turbo decoding a complete data block is iteratively processed in a loop which comprises two component decoders using the Max-Log-MAP algorithm [2], as depicted in Figure 1. These two decoders exchange so called extrinsic information (Λ^e). One decoder works on the original data sequence and the other one on an interleaved sequence.

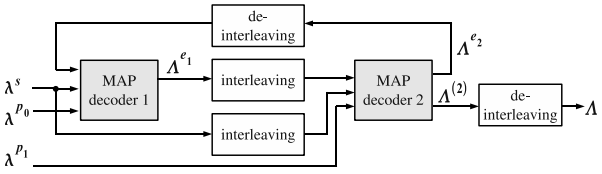

Fig. 1. General Turbo Decoder

The data is first processed by the MAP decoder 1 and then by the MAP decoder 2, so only one decoder works at any time. Therefore, both MAP decoders are mapped on the same hardware instance. MAP decoders process the block serially. Therefore, they are called *SMAP*. They produce one or two output bits per clock cycle.

To enhance the throughput of the turbo decoder, one needs to enhance the throughput of the MAP decoder. This is done by splitting the code block into p sub blocks, and process all sub blocks in parallel by parallel SMAP engines. This architecture is called parallel SMAP architecture.

In parallel SMAP turbo decoders, interleaving is a big issue. When p parallel SMAP decoders are used, they need to access p extrinsic information words in the same clock cycle. To support this parallel access scheme, the extrinsic memory is divided into p banks of the same size. Figure 2 shows an example for such an architecture as it is configured for odd half iterations, where MAP decoder 1 referring to Figure 1 is processed. During even half iterations, the extrinsic information is accessed in an interleaved sequence rather than regular. Here, conflicts can occur when more than one MAP decoder wants to access one memory bank at the same time, as shown in Figure 3 The interleaver is called *conflict free* if in every clock cycle, all memory banks are accessed by only one MAP decoder. In this case every SMAP decoder can access a dedicated memory bank through a crossbar.

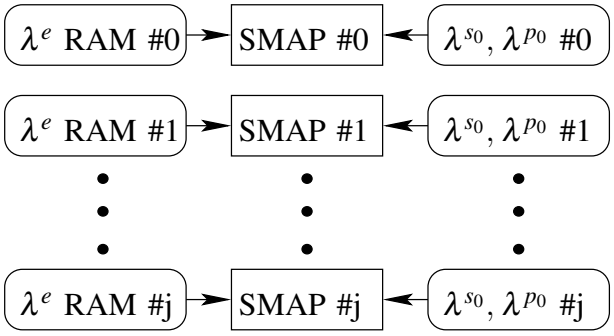

Fig. 2. Parallel MAP Decoder configured as MAP1

The only difference between HSPA and LTE turbo codes is the interleaver: LTE uses a more sophisticated *QPP* interleaver [3], which replaces the old UMTS interleaver deployed in HSPA. This QPP interleaver supports conflict-free parallel turbo decoding. Moreover, is is much less complex to compute the addresses at run time than for the UMTS interleaver. Therefore, most of the published LTE decoders [4]–[8] use

978-1-4577-0161-0/11/$26.00

such a dedicated circuit to compute the addresses on the fly.

The linear address generated by these circuits are split into a bank number and an address inside the bank. For lower degrees of parallelism p like $p=2$ or $p=4$, the splitting is traditionally done by comparing the absolute interleaver address with p thresholds. But this method scales badly, as it is $O(p^2)$.

To avoid a splitting of the linear addresses, [4] uses a master-slave sorting network. They need p interleaver generators and one sub block address generator. The p interleaver addresses are sorted using the master sorter network, and the switch configuration in every compare/exchange entity is the output of the master network. These switch configuration settings are fed into the slave network, which consists only of exchange gates, and permutes the data from the RAM. The advantage of this approach is that the slave network is smaller than a crossbar (it scales with $O(n \cdot \log(n)^2)$, but comes at the cost of the master network, which is comparable in size to p pipelined dividers.

A more traditional method is to use either a (pipelined) divider, or a multiplication of the linear addresses with the inverse sub block size, and then to permute the data using a crossbar. The decoders in [5], [7] use p parallel address generators, so they use one of these two methods.

This paper describes a novel monolithic interleaver generator for LTE that can directly compute the bank numbers and the addresses inside the bank. With this interleaver generator, an area and power reduction by an order of magnitude can be achieved compared to traditional architectures.

Using the master slave sorting network is still possible with our proposed interleaver generator, by sorting the bank numbers. This has the advantage of a reduced bit width in the master sorting network (from 13 bits to $\log_2(p)$ bits).

This paper is structured as follows: Section II gives the mathematical background on our architecture. Section III shows our proposed hardware architecture. Section IV shows synthesis and power results of our new architecture compared with a classical one. Section V concludes this paper.

II. Analysis of the LTE Interleaver

The LTE Interleaver uses the quadratic form with $\pi(i)$ being the interleaved address:

$$\pi(i) = (f_1 \cdot i + f_2 \cdot i^2) \bmod K \tag{1}$$

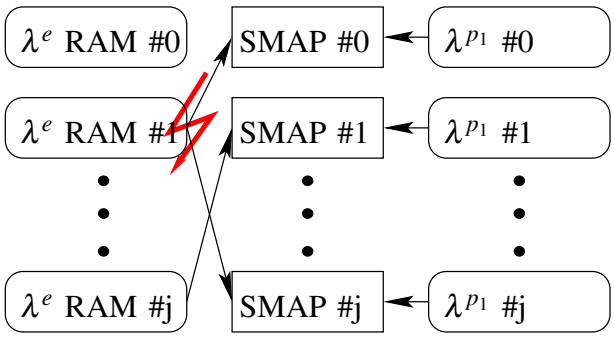

Fig. 3. Parallel MAP Decoder configured as MAP2 with conflicts

The LTE standard [9] contains a table with valid pairs of f_1, f_2, K. K is the block size, while f_1 and f_2 are parameters for the interleaver. f_1 is a prime number, while f_2 is even. It is already known [3] that this interleaver is conflict free for any parallelism that is a divider of the block size K. This means that if the extrinsic memory is divided into p banks, all p SMAPs access a different memory bank at a time. We do not need the interleaver address by itself. We rather require the memory address inside each memory bank, and the corresponding bank number. These addresses could be derived from $\pi(i)$ by division.

To reduce the number of parallel divisions, which are costly operations in terms of area and power, we use a mathematical approach to simplify the interleaver. The required mathematical backgrounds will be briefly described in the following.

Let p be the parallelism of our parallel MAP engine (the number of SMAPs). By inspecting the LTE standard [9] we can see that valid values for p are 8 for small blocks $K < 2048$ and 64 for larger blocks $K \geq 2048$. Also smaller factors for p, which are a divider of 8 or 64 respectively, can be used. This means that all powers of two up to 64 can be used as p.

As p is a divider of K, we define the sub block size S as:

$$S := \frac{K}{p} \tag{2}$$

As we do not need the absolute interleaver address, but the bank number b, and the address in the bank a, we define:

$$a(i) := \pi(i) \bmod S \tag{3}$$

$$b(i) := \left\lfloor \frac{\pi(i)}{S} \right\rfloor \tag{4}$$

The address/bank number pair for the jth SMAP at clock cycle c is (remember that an input value is read in each clock cycle):

$$a_j(c) = a(c + j \cdot S) = \pi(c + j \cdot S) \bmod S \tag{5}$$

$$b_j(c) = b(c + j \cdot S) = \left\lfloor \frac{\pi(c + j \cdot S)}{S} \right\rfloor \tag{6}$$

Using Equation 1, $a_j(c)$ can be computed as:

$$\begin{aligned} a_j(c) &= \pi(c + j \cdot S) \bmod S \\ &= \left(f_1 \cdot (c + j \cdot S) + f_2 \cdot (c + j \cdot S)^2\right) \bmod S \\ &= \left(f_1 \cdot c + f_2 \cdot c^2 + S \cdot (f_1 j + f_2 \cdot (2cj + Sj^2))\right) \bmod S \\ &= \left(f_1 \cdot c + f_2 \cdot c^2\right) \bmod S = \pi(c) \bmod S \\ &= a(c) \end{aligned} \tag{7}$$

Since $a_j(c) = a(c)$, the address is independent of the SMAP. Thus, it can be used to address all individual memory banks. This already widely known property [3] of the interleaver allows us to merge the individual memory banks into one single wide memory.

To avoid the floor operation on the bank number generation, we can rewrite:

$$b(i) = \left\lfloor \frac{\pi(i)}{S} \right\rfloor = \frac{\pi(i) - \pi(i) \bmod S}{S} = \frac{\pi(i) - a(i)}{S} \tag{8}$$

The term $\pi(i)-a(i)$ is always greater or equal to 0, and also smaller than K. Hence, it can be substituted by $(\pi(i)-a(i))\bmod K$. Similarly, $a(i)$ is smaller than K and greater or equal to 0. So using the formula

$$(x+y)\bmod z = (x \bmod z + y \bmod z)\bmod z$$

we can rewrite:

$$\begin{aligned} b(i) &= \frac{\pi(i)-a(i)}{S} \\ &= \frac{\big((f_1\cdot i+f_2\cdot i^2)\bmod K - a(i)\bmod K\big)\bmod K}{S} \\ &= \frac{\big(f_1\cdot i+f_2\cdot i^2-a(i)\big)\bmod K}{S} \end{aligned} \tag{9}$$

For the jth SMAP, we can compute the bank number as:

$$\begin{aligned} b_j(c) &= \frac{\big(f_1\cdot(c+j\cdot S)+f_2\cdot(c+j\cdot S)^2-a_j(c)\big)\bmod K}{S} \\ &= \frac{\big(f_1c+f_2c^2-a(c)+S(f_1j+2f_2jc+f_2j^2S)\big)\bmod K}{S} \\ &= \left(\frac{\pi(c)-a(c)}{S}+\big(f_1\cdot j+f_2\cdot j^2\cdot S+2f_2\cdot j\cdot c\big)\right)\bmod p \\ &= \big(b(c)+\big(f_1\cdot j+f_2\cdot j^2\cdot S\big)+2f_2\cdot j\cdot c\big)\bmod p \end{aligned} \tag{10}$$

By defining two constants, $C1_j$ and $C2_j$:

$$C1_j := \big(f_1\cdot j+f_2\cdot j^2\cdot S\big)\bmod p \tag{11}$$

$$C2_j := 2f_2\cdot j \bmod p \tag{12}$$

We can compute:

$$b_j(c) = (b(c)+C1_j+C2_j\cdot c)\bmod p \tag{13}$$

As all valid values for p are powers of 2 and smaller or equal to 64, all $C1_j$ are constants with up to six bits. Further, the modulo operation can be implemented with very low complexity. All values for f_2 are even values, therefore $C2_j$ can be rewritten as:

$$C2_j = 4\cdot\left(\left(\frac{f_2}{2}\cdot j\right)\bmod\frac{p}{4}\right) \tag{14}$$

$$C2_j = C2_{(j \bmod \frac{p}{4})} \tag{15}$$

This means that at most a four bit wide multiplication needs to be done. Also note that at most 16 values for the term $C2_j\cdot c$ need to be computed (Equation 15).

III. Hardware Architecture

The architecture of our new interleaver generator is shown in Figure 4. As we still need to compute $a(c)$ and $b(c)$, we use one interleaver generator to generate $\pi(c)$. Then we need to split this into $a(c)$ and $b(c)$. One way to accomplish this is to use a fully pipelined divider which computes $b(c)$ as the result and $a(c)$ as the reminder. A pipelined divider is however a large circuit and also has the drawback that it needs several pipeline stages.

As S stays constant while the whole code block is being decoded, it is much more efficient to use a multiplier to multiply $\pi(c)$ with $1/S$, which can be implemented in fixed point hardware by a multiplication with $\lceil 2^n/S\rceil$, and divide the result by 2^n. A sufficient large value for n needs to be chosen. As this will yield only $b(c)$, we use Equation 7 to generate $a(c)$, using a similar circuit than the $\pi(c)$ generator.

To get a five bit result, one has to multiply the linear address with a 14 bit constant, resulting in a bulky 13×14 bit multiplier. By subtracting $a(c)$ from $\pi(c)$, we get $b(c)\cdot S$, an exact multiple of S. We can then multiply this number $2^n/S$, using a smaller value for n. Therefore the size of the multiplier is greatly reduced. For a value of $p=32$, we could reduce the required value of n to 13 for $S\geq 145$, 12 for $145>S\geq 66$ and 13 for $66>S$. This results in a 13×6 bit multiplier.

The computed value $b(c)$, as well as the value c, generated by a simple counter, are fed into a permutation generator that generates all p $b_j(c)$ values to control the crossbar. The permutation generator leverages Equation 13. The permutation generator will add the value $b(c)$ to $C1(j)$ and $C2(j)\cdot c$. The values of $C2(j)\cdot c$ are generated directly using multipliers, as their bit width is only $\log_2(p)-2$ (which is 3 in the case of $p=32$, and 4 in the case of $p=64$). Also Equation 14 shows that only $p/4-1$ multiplications need to be made (7 in case of $p=32$).

The design has only two pipeline stages, the first pipeline stage is the $\pi(c)$ and $a(c)$ generation, while the second stage is the multiplication with $2^n/S$ and the permutation generation. The interleaver generator can be configured to output $a(c)$ one clock cycle earlier than the $b_j(c)$ permutation vector to compensate the delay of a synchronous RAMs.

The $C1_j$ values are precomputed using a dedicated circuit, which needs only f_1, f_2, and S. The circuit is built using two multipliers, doing $C1_j=(j\cdot(f_1+f_2\cdot j\cdot S))\bmod p$. It produces one $C1_j$ value per clock cycle, needing not more than 33 clock cycles to compute all $C1_j$ values. Therefore this kind of preprocessing can be done in the first half iteration even for GBit/s turbo code decoders. The preprocessor can be parallelized to produce two values per clock cycle to support even faster decoders.

The inputs to the interleaver are: f_1, f_2, K for the $\pi(c)$ generator, $f_1 \bmod S$, $f_2 \bmod S$ and S for the $a(c)$ generator, and $2^n/S$ for the multiplier. $f_1 \bmod S$, $f_2 \bmod S$ and $2^n/S$ can either be derived from f_1, f_2, K and p using a serial divider, or they can be stored in a table, extending the f_1, f_2, K table that needs to be present anyway. The serial divider would use 18 clock cycles, and can run in parallel to the $C1_j$ preprocessor.

IV. Results

To evaluate the benefits of our new architecture, a state-of-the-art architecture was implemented. The interleaver was designed for a parallelism of 32, which is a divider of 64 and is suitable for a 1 GBit/s turbo code decoder. This state-of-the-art architecture uses 32 interleaver generators which are preloaded with start offsets which generate the $\pi(c+j\cdot S)$ values, and one interleaver that generates the memory address $a(c)$. This type of interleaver architecture is used for example in [5]. Unfortunately, [5] does not describe how the bank numbers are

TABLE I
POST P&R RESULTS

	State of the art	New
Area (μm^2)	148801	13411
Cell Count	38636	2201
Best Case Power (mW)	62.8	6.7
Nominal Case Power (mW)	57.7	6.0
Worst Case Power (mW)	49.9	5.2

generated from the absolute interleaver addresses. In our state-of-the-art architecture, the memory address was subtracted from all 32 interleaver addresses, so that the division by S could be implemented using a multiplication with a six bit constant. This seems to be the most efficient way to implement the division by S, requiring only a 6×13 bit multiplier and only one output pipeline stage. The complete latency is 2 clock cycles (one for the interleaver itself, one for the multiplication).

This was compared to our new architecture, where we use one interleaver to generate the $\pi(c)$ value and another one to get the $a(c)$ value. From these two values, $b_0(c)$ is generated by the same multiplication method as in the classical interleaver. The $b_0(c)$ value is fed into the permutation generator without any pipeline registers, to have the same latency as the classical architecture.

Both designs were synthesized and placed and routed in a 65 nm CMOS low power CMOS process for 300MHz. Power numbers where generated using Synopsys PrimeTime utilizing post P&R simulation activity files. Table I lists area and power.

Unfortunately, all of our references do not give power numbers on the interleaver generator alone, but implementation details given suggest that their interleavers are identical to our state of the art decoder.

It can be seen that our proposed method to built such an interleaver generator yields a $10\times$ area reduction as well as a $9\times$ power reduction for $p = 32$. This factor will be larger for $p = 64$, and will be smaller for $p < 32$. In the classical approach, every bank number output requires four 14 bit adders, two 13 bit registers, and one 13×6 bit multiplier. In addition one $a(c)$ generator is required for the whole interleaver.

By using our new architecture, the first bank number output requires the same resources as in the classical architecture, while all other bank number outputs only require two $\log_2(p)$ bit adders, and additionally one $log_2(p) - 2 \times log_2(p) - 2$ multiplier per four output ports. As the first bank number output is equal to the classical interleaver, it is clear that the power and area reduction increases when p increases.

If the classical architecture is used together with a master slave sorting network as in [4], the multipliers can be avoided. Using our new method has the advantage of a reduced bit width in the master sorting network, so a similar area reduction can be expected.

V. CONCLUSION

A novel interleaver generator for LTE was presented. This interleaver generator achieves a $9\times$ power reduction and a $10\times$ area reduction over a classical approach. All preprocessing in the interleaver generator can be done in the first half iteration even for very fast turbo code decoders, which also simplifies the controller of the turbo code decoder.

An estimate of the power consumption of a GBit/s turbo code decoder in this technology is $500\,mW$ to $1500\,mW$ ([5] state to have $845\,mW$ in 65 nm, but at a very low supply voltage, which would reduce the power of the interleaver further), which means that the turbo code decoder's power consumption can be reduced by 6% to 20%.

REFERENCES

[1] Third Generation Partnership Project, "3GPP home page," www.3gpp.org.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," in *Proc. 1993 International Conference on Communications (ICC '93)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[3] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.

[4] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "A 390Mb/s 3.57mm^2 3GPP-LTE Turbo Decoder ASIC in 0.13μm CMOS," in *Proc. IEEE International Solid-State Circuits Conference - Digest of Technical Papers, 2010. ISSCC 2010*, vol. 53, San Francisco, USA, Feb 2010, pp. 274–275.

[5] Y. Sun and J. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," *Integration VLSI Journal*, 2010.

[6] T. Ilnseher, M. May, and N. Wehn, "A Multi-Mode 3GPP-LTE/HSDPA Turbo Decoder," in *Proc. IEEE International Conference on Communication Systems (IEEE ICCS 2010)*, Nov. 2010, accepted for publication.

[7] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *Proc. IEEE Custom Integrated Circuits Conference CICC '09*, Sep. 2009, pp. 487–490.

[8] S.-J. Lee, M. Goel, Y. Zhu, J.-F. Ren, and Y. Sun, "Forward error correction decoding for WiMAX and 3GPP LTE modems," in *Proc. 42nd Asilomar Conference on Signals, Systems and Computers*, Oct. 2008, pp. 1143–1147.

[9] Third Generation Partnership Project, *3GPP TS 36.212 V10.0.0; 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 10)*, Dec. 2010.

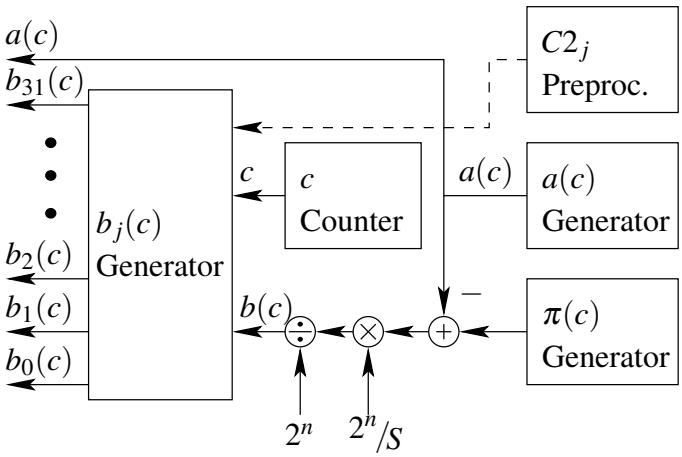

Fig. 4. New Interleaver for $p = 32$