A VLSI Design for the LTE Turbo Decoder

I. S. Souza[1] and V. T. dos Reis[2]

[1,2]Ilan Schnitman Souza and Vinícius Torres dos Reis, LSI-TEC, Salvador-Bahia, Brazil,
e-mail: ilan.souza@lsitec.org.br

ABSTRACT

Due to the need of high performance, new wireless telecommunications standards such as WIMAX and LTE are using turbo-codes as a forward error correction (FEC) choice. This design targets either a self-contained IP (Intellectual Property) or integration into the physical layer project. This work presents all steps for the implementation of an LTE standard turbo decoder: from algorithm modeling in high level programming language to architecture using a sliding window approach seeking throughput needed, getting into physical implementation at TSMC 65nm. Each aspect of the specification and performance were analyzed in their proper stages.

Index Terms: ASIC, VLSI, LTE, Digital Communications, Turbo-Codes.

1. INTRODUCTION

New wireless communication standards such as 3GPP Long Term Evolution (LTE) and WIMAX intend to provide high speed rates of upload and download ensuring data reliability. One of the main bottlenecks is to achieve the high throughput rates requested by those standards. The standards employ the near-Shannon limit error-coding known as turbo-codes, proposed in 1993 by Berrou [1]. The process of decoding shown in [1] requires several iterations over a data block to achieve maximum estimation. The main bottleneck is to achieve high throughput rates requested by those standards.

To overcome this barrier this paper presents the implementation of an 8x parallel LTE turbo decoder that increases throughput without performance degradation. The parallelism is allowed by the QPP interleaver [3] that processes the contention-free property.

2. TURBO-CODE BASIS

A. Turbo Encoder

The turbo encoder in the LTE standard is specified as shown in Figure 1 [2].

It is composed of two parallel convolutional 8-state encoders linked by an interleaver. Input data at time k, c_k, is fed to the first encoder in a regular order, generating the systematic x_k and the parity z_k while the second encoder is fed with data in an interleaver order c'_k, from this second encoder we get coded bit (*parity2*) z'_k. Figure 2 shows in a trellis diagram the possibilities for the states

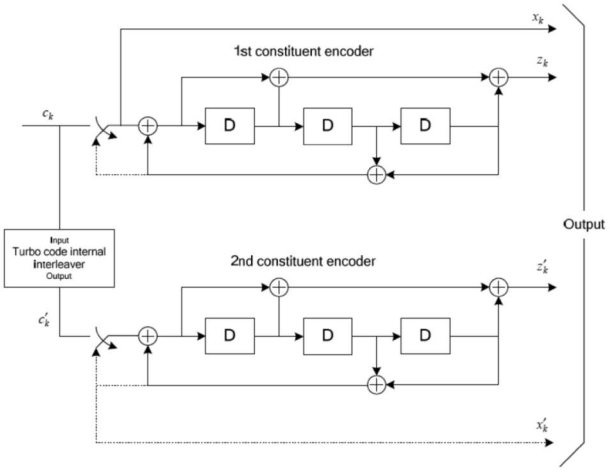

Figure 1. 3GPP LTE Turbo Encoder

based on the input of the encoder. To make it possible to send the encoder to a known state 3GPP standard establishes a mechanism to send both decoders to initial state, initially the first one followed by the second. The bits generated with this process are named tail bits. This structure makes it possible to achieve an approximately coding ratio, where K information bits are coded into bits, and those additional 12 bits relates to trellis termination, 6 for each encoder. It is also defined in [2], 188 possible block sizes (K bits) from 40 to 6144.

B. Turbo Decoder

The main idea behind turbo decoding is to improve the error rate based on successive exchange of the *extrinsic* information L_e obtained from the log-likeli-

hood ratio (LLR) between two soft-input/soft-output decoders (SISO).

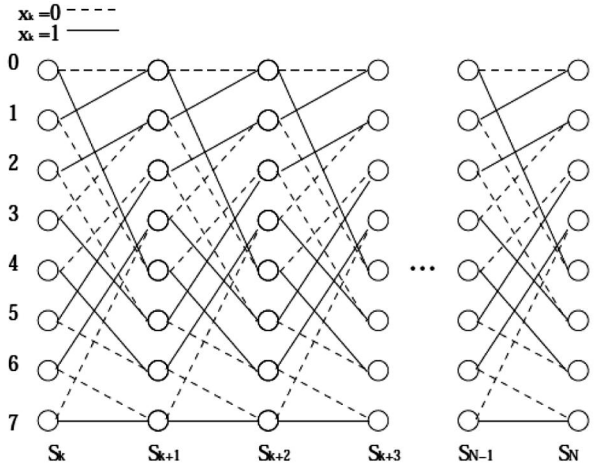

Figure 2. Trellis Diagram

As proposed by [1], in the beginning of the process, the first decoder calculates the first LLR using the systematic soft-bit x_k and the parity soft-bit z_k as inputs. Immediately after having calculated the LLR, the second SISO uses the interleaved version of the systematic soft-bit, the parity soft-bit z'_k (that comes from the encoding of interleaved data) and also the interleaved extrinsic information coming from the first SISO, now called as *a priori* information L_a. The process is going to be repeated for a specific number of iterations or after the LLR convergence. Hard decision is the process of taking the LLR to 0 or 1 depending on its sign. The decoder is depicted in Figure 3.

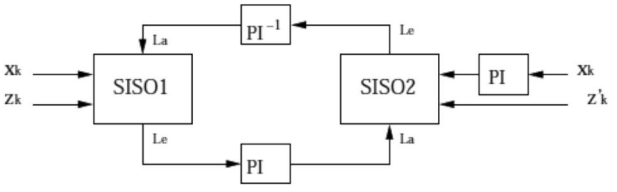

Figure 3. Turbo Decoder Diagram

C. MAX-LOG-MAP

In order to implement SISO decoders an optimal decoding solution called *Maximum A Posteriori* (MAP) was proposed by [4], but its increased complexity in comparison to the gain of performance was not attractive when compared to Viterbi decoding. However, in [5], an algorithm approximation called MAX-LOG-MAP was described. This approach maintains performance while presenting no more than four times the complexity of a Viterbi decoder for the same code. In order to define the *a posteriori* metric, this parameter is used not only to give the hard decision but also to calculate the *extrinsic* information. It is going to be used as *a priori* information in turbo decoding process. The branch metric calculates the probability of the encoder Finite State Machine (FSM) going from a state s' at the time to a state s at k. It is defined by:

$$\gamma_k(s',s) = lnp(y_k|s,s') + lnP(s|s') \quad (1)$$

where y_k is the sequence of received channel output symbols. Let the forward metric k be the probability of being taking to a state from a state defined by:

$$\alpha_k(s) = \max_{s'}[\alpha_{k-1}(s') + \gamma_k(s',s)];\ \alpha_0(0) = 0; \quad (2)$$

$$\alpha_0(s) = -\infty, s \neq 0 \quad (3)$$

Also the backward metric, that is calculated similarly to the forward metric but starting from the last node of the trellis, is given by:

$$\beta_{k-1}(s) = \max_{s}[\beta_k(s') + \gamma_k(s',s)];\ \beta_N(0) = 0; \quad (4)$$

$$\beta_N(s) = -\infty, s \neq 0 \quad (5)$$

Those metrics can be combined to calculate the approximation of the *a posteriori* probability ratio as follow:

$$L_k(s) \approx \max_{s',s:u_k=+1}[\alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)] - \max_{s',s:u_k=-1}[\alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)] \quad (6)$$

D. Interleaver

Coding scheme was carefully considered during LTE standard specification [2], a quadratic polynomial permutation (QPP) contention free interleaver was relevant for turbo-code choice [9] as it allows for a great degree of parallelization. To ensure high throughput turbo-code needs a contention free interleaver, then memory and SISO can be split in several blocks, assuring that each memory block will not be accessed by different SISOs at the same time. Interleaver is important in turbo-code scheme as it lower the correlation between neighbor bits. The same error must not appear in the same bit for the interleaved sequence and the non-interleaved sequence as it will be sent in different moments. To better perform this task a pseudo random sequence is created in a way the encoder and decoder can create de same sequence for the same data block. Considering the hardware architecture, the interleaver function is represented in (7), where and are constant values provided in LTE specification [2] and K is the number of bits in the encoding package. This relation would require an intensive calculation as it uses a square and a reminder operation. An approach to reduce operations is to consider the calculations are done in a progressive way. The reminder operations are done in every calculation cycle then it can be reduced to simple subtraction operation. The square would not be needed anymore, as knowing the previous value of interleaver position and an intermediate register

which keeps the summation of parameter every cycle, the operation is reduced to adders as shown in (10).

$$\pi_{(i)} = (f_1 i + f_2 i^2) mod k \quad (7)$$

$$\pi_{(i+1)} = (f_1 i + f_1 + f_2 (i^2 + 2i + 1)) mod k \quad (8)$$

$$(\pi_{(i+1)} - \pi_{(i)}) mod k = (f_1 + f_2 + 2 f_2 i) mod k \quad (9)$$

$$\pi_{(i+1)} = (\pi_{(i)} + (f_1 + f_2) mod k + (\sum_{(j=0)}^{i} (2 f_2) mod k) mod k) \, mod k \quad (10)$$

To allow parallelization there is a direct relationship between the interleaved address for each block. The calculations can be simplified to prove the relation is simplified to the usage of different memories with the same address. This result shows the QPP property, that each memory is used by a different SISO. The relationship between an interleaved address and another one *n* positions ahead is shown in (12).

$$\pi_{(i+n)} = (f_1 i + f_1 n + f_2 i^2 + 2 f_2 in + f_2 n^2) mod k \quad (11)$$

$$\pi_{(i+n)} = (\pi_{(i)} + f_1 n + 2 f_2 in + f_2 n^2) mod k \quad (12)$$

The reminder property shown in (13), is very useful for hardware simplification as perform this operation with constant value, especially if this is a 2's multiple, has easy implementation. It becomes a bit mask in the Least Significant Bits (LSB) of the variable.

$$\left(f_1 \frac{k}{n}\right) mod k = \frac{k}{n} (f_1 mod n) \quad (13)$$

The scheme using 2 process blocks has a simple implementation as it uses the information that is an odd number which allows the reduction from (14) to (15). From results it is easily extracted that each SISO will always access different memories.

$$\pi_{(i+\frac{k}{2})} = (\pi_{(i)} + (f_1 \frac{k}{2}) mod k + (f_2 ik) mod k + (f_2 \frac{k^2}{4}) mod k) mod k \quad (14)$$

$$\pi_{(i+\frac{k}{2})} = \left(\pi_{(i)} + \frac{k}{2}\right) mod k \quad (15)$$

Four process blocks can use the same analysis used for two as seen in (17). In this case the same simplifications are done based on the information that the value of *K* for block sizes with 4 sliding windows are always multiple of 16. The same analysis can be done for in (18).

$$\pi_{\left(i+\frac{k}{4}\right)} = (\pi_{(i)} + (f_1 \frac{k}{4}) mod k + (f_2 i \frac{k}{2}) mod k + (f_2 \frac{k^2}{16}) mod k) mod k \quad (16)$$

$$\pi_{(i+\frac{k}{4})} = \left(\pi_{(i)} + \frac{k}{4} (f_1 mod 4)\right) mod k \quad (17)$$

$$\pi_{(i+\frac{3k}{4})} = \left(\pi_{(i)} + \frac{3k}{4} (f_1 mod 4)\right) mod k \quad (18)$$

For 8 sliding windows the relationship between memories is more difficult to be detected but it is still valid. The simplification can be done as block size are always multiple of 64.

$$\pi_{(i+\frac{jk}{8})} = (\pi_{(i)} + (f_1 \frac{jk}{8}) mod k + (f_2 i \frac{jk}{4}) mod k + (f_2 \frac{jk^{16}}{64}) mod k) mod k \quad (19)$$

$$\pi_{\left(i+\frac{jk}{4}\right)} = (\pi_{(i)} + \frac{jk}{2} (f_1 mod 8 + \frac{jk}{4} (f_2 mod 4)(i mod k)) mod 4) mod k \quad (20)$$

3. VLSI IMPLEMENTATION

A. System Modeling

In order to have a design reference model and to early estimate performance results, a behavioral description were designed in a high-level programming language. MATLAB language has been chosen to perform this task as it can take advantage of its powerful matrix manipulation and to make use of some built-in communication processing functions. This model allows an early analysis of the algorithms and its performance, ensuring the correctness in using a specific solution.

At this point the main definition is related to the MAX-LOG-MAP algorithm. In a first approach it is described without any constraints, however, as the architecture becomes more defined, the description gets closer to the hardware approach. The final model does the calculations exactly like the hardware does. This allows a very trustful analysis, as the performance simulated in this stage the same as in the final design. With this description most of the strength of the high-level language was put behind to make it a true reference model for the hardware. In this project the approximation to the hardware let the MATLAB model simulation really slow, a study about performance of software blocks allows the correct replacement of these blocks for less computing intense model using the programming language C.

A simulation environment was built to check the model and to perform the analysis. One of the verification environments built for the RTL (Register Transfer Level) project uses the input/output signals from the high-level simulation to check design functionality. Figure 4 shows the verification environment for both MATLAB and RTL codes. The input data is randomly generated as a vector of bits with a specified length, passing through all transmission process. Original data is encoded and the output is concatenated as a sequence of systematic and parity

bits, then the result array are modulated. Modulation simulation is a very simple process, it consists in grouping the bits depending on the modulation scheme and map it to a symbol. These symbols are exposed to the channel effect, which is simulated as the insertion of an Additive White Gaussian Noise (AWGS) in the signal. This approach is proven to get close to the real channel behavior. These signals then pass through an *LLR* finally been ready for decoder. These data can be stored to provide information for the RTL Verification Methodology called *Bitmatch*, this will be discussed deeply lately.

The output data of the decoder compares with the generated in the beginning of the verification chain. This comparison gives valuable performance information, as it shows the number of frames that are not decoded correctly. This information can be represented as in Table 2 with the noise and modulation scheme.

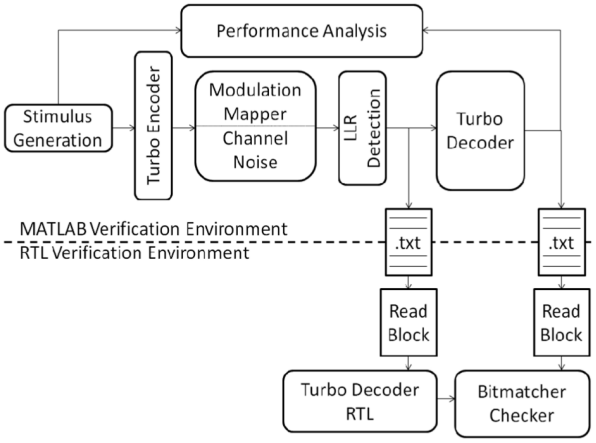

Figure 4. MATLAB verification environment

B. Design

Focusing in achieving the required high throughput, the proposed implementation was a parallel architecture going from 1 up to 8 SISO decoders depending on the input frame size. This architecture proposes those input data to be broken in the proper number of windows, defined by Table 1 and each window is saved in a memory and processed individually by its respective SISO. In order to reduce the loss of performance caused by this approach, overlapped windows were used in those computations.

Systematic, *Parity* and *Parity2* received data are spited, according with the number of SISO processors and stored into 3 distinct block memories containing 8 memories each. The addressing of those memories is possible through the contention free interleaver and the control unit CTRL. To *extrinsic* information storage, an extra memory block containing 8 memories are required. CTRL block is going to ensure the correct number of SISO decoders to be used depending on K and manage cycles of processing, defined as SISO1 and SISO2. Multiplexed data coming from *systematic1* or *systematic2* is provided to SISO. Also addressing of systematic block of memories allows it to feed SISO processor with systematic data or the interleaved one depending on the cycle. As shown in Figure 6, SISO processors are shared and SISO1 cycle regards to the calculation of L_k and L_e, using as inputs *systematic*, *parity1* and L_a, when available. SISO2 cycle uses *systematic interleaved*, *parity2* and L_a. The sharing of SISO processors is possible as the calculations are done in different time windows. The sequence of processing starts in SISO2 cycle and finishes in a SISO1 cycle saving in this way a memory at output as the data in this case does not need to be de-interleaved and also saves *fracKN* clock cycles, where N is the number of SISOs used. At each selected SISO L_k and L_e are calculated as shown in Figure 7.

First GAMMA calculation block reads L_a, Systematic and Parity stored in memory blocks and is calculated for 16 possibilities. At the same time calculation starts, it is possible to start calculation at BETA/ALPHA block. β is calculated at this moment because CTRL set up the block with β initialization parameters and read memories in a reverse way, matching with backwards calculation. It is calculated for all possible states (in this calculation needs a correction factor to improve max-log approximation results). β must be normalized for all states and the results normalized are stored in a internal memory (one more time correction factor are needed).

Table 1. Number of SISOs related to the frame size

$K < 784$	1
$784 \leq K < 1568$	2
$1568 \leq K < 3136$	4
$3136 \leq K$	8

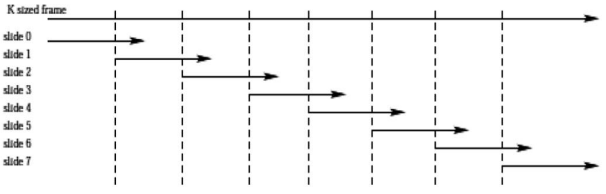

Figure 5. Sliding Window scheme

After β calculation finishes, $\boldsymbol{\gamma}$ is calculated again, now in a direct way, so calculation starts. One more time using max-log approximation adds a correction factor. L_k will be calculated right after α calculation has its initial results. It is going to use also the stored β and $\boldsymbol{\gamma}$ at this processing. At the same time window the *extrinsic* information is calculated using systematic, parity and L_k, this last one can give same attenuation depending on reliability of channel represented by .

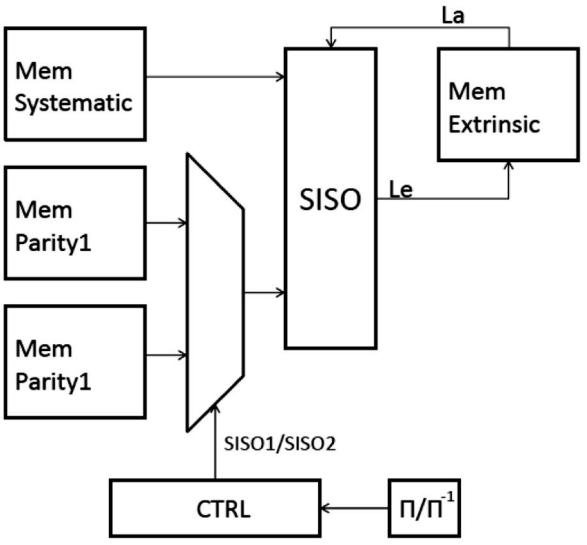

Figure 6. Architecture proposed for Turbo Decoder

C. Verification

This project uses two different approaches to perform verification. One is based on performance analysis and the other in the comparison with a reference model. They both have its strengths and weakness, and they were used in a complementary way in different stages of the design.

As described previously the system model uses an environment that allows a good segregation of Design Under Test (DUT), which is the reference model for the RTL. This characteristic can be used to exchange information between the MATLAB verification environment and the RTL (Verilog) environment. In this approach to run a set of configuration, the MATLAB must run first, generating input data to the DUT as well as the output that are stored in files, this can be seen as an off-line simulation. The stored data are used in a Verilog environment to feed the RTL DUT and compare the results.

The second verification environment is based on an object oriented SystemVerilog methodology. Modifications were done trying to achieve easy integration and reusability. The goal in this methodology is to allow usability to verify not only turbo decoder block but to easily integrate it in the whole LTE core verification too.

The modified verification environment uses random generated input data to feed the design. The environment uses most modern verification items such as

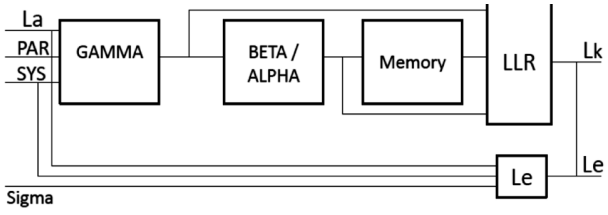

Figure 7. SISO Diagram

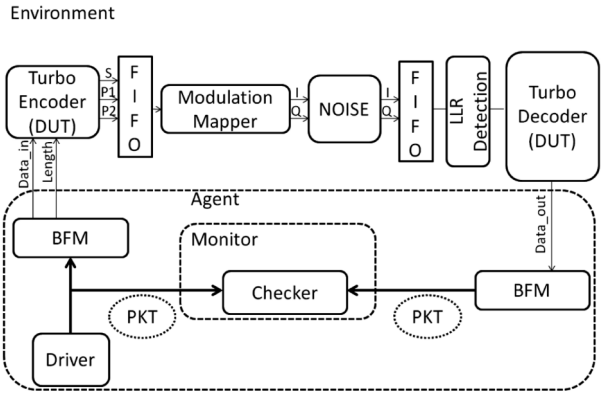

Figure 8. Verification Environment

driver, monitor, etc, and the turbo encoder, a FIFO that controls data flow and generates signaling data, modulator (modulation map), a channel emulator and finally the turbo decoder the is the Design Under Test (DUT). The flow can be seen in Figure 8. First the encoder receives a random generated, after that, the modulation mapper maps bits to symbols, this is the way to send data through the channel. The modulation scheme can be BPSK, QPSK, 16QAM and 64QAM. The noise is denoted in *dB* representing the degradation added in the symbols. An additive White Gaussian Noise (AWGN) is a good approach to emulate the channel, in this system a zero mean and variance 1 random generated noise signal, with a constant spectral density and Gaussian distribution in amplitude. After noise insertion, data feeds the decoder and finally it checks the error correction level the system achieves. It measures the Bit Error Rate (BER) and the Frame Error Rate (FER) which is the number of sent blocks that have error in bits after been decoded.

D. Performance Analysis

The main concerns in the project are related to Frame Error Rate under a Signal Noise Ratio (SNR) condition, throughput, power consumption and silicon area. The last two parameters can only be measured after the physical implementation, and they are not defined in the specification. They are both affected by the technology and frequency.

The FER describes the design error correction capability as it reveals the number of frames with deco-

Table 2. Performance with different block sizes

Nº of SISOs	Package size	Simulation Samples	FER
1	40	2000	0.0675
1	768	2000	0.0055
2	784	2000	0.0035
2	1536	2000	0.0090
4	1568	2000	0.0060
4	3072	2000	0.0095
8	3136	2000	0.0085
8	6144	2000	0.0120

ded data different from the transmitted. This parameter was used instead of Bit Error Rate (BER) as the frame has a mismatch in one bit, the whole packet is corrupted. Table 2 shows an example of the analysis, for 8.4dB SNR modulated with 64QAM with 8 iterations.

Another interesting data is presented in Table 3 and Figure 9, showing the FER with different modulation. For the same block size the FER has a great variation depending on the modulation scheme. It is widely known that as the modulation becomes more complex, carrying more data per symbol, the demodulation gets more error prone. Other blocks of the LTE system will determine the modulation scheme adoption, the turbo decoder does not need this information, and it is only used in this project to analyze performance.

In the LTE project, turbo decoder is one of the main bottlenecks to achieve the specification data rate. The throughput directly affects the amount of received data the processor can manage without any stall. This parameter calculation is not very simple, and requires the analysis of several blocks of LTE. The complete interpretation of the standard may not be of great interesting in this paper, but some analysis can be done. The latency is

$$len + iter(3\left(\frac{len}{window} + 10\right) + 6)(\text{iter} - 1)\left(\frac{\text{len}}{\text{window}} + 13\right) + \left(\frac{\text{len}}{\text{window}} + 15\right) \quad (21)$$

determined by the number of clock cycles required to perform the error correction,

where *len* is the package size, *iter* number of iterations and *window* number of sliding SISOs used. The latency directly affects the throughput, as with a faster system it can process more data in the same period. The amount of clock cycles an input data takes to be processed is represented in (21). The final throughput is calculated based on (21). The adopted frequency has to be considered as the throughput depends on the amount of bits calculated in a second.

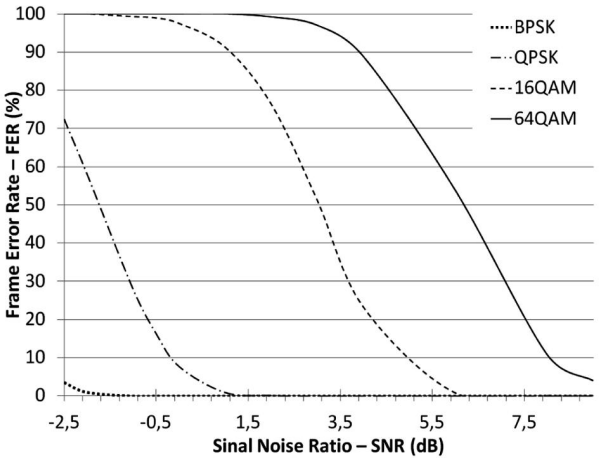

Figure 9. Performance (FER) with different modulation

E. Implementation Results

The design proposed was synthesized and physical implemented for TSMC 65nm CMOS technology. The target frequency was 120MHz, as it is the maximum clock for the whole system it was integrated. This turbo-code architecture could support higher frequency, but to integrate it in a physical layer core it was important to use a coherent clock signal. One of the main reasons not to adopt faster frequency is the power consumption that is increased with the speed, not only for the turbo decoder core but also for other parts of the chip such as clock generator. An important information in the VLSI implementation as seen in Table 4 is the area usage. This is relevant in order to compare different architectures efficiency. In this design, memories are responsible for most of the area utilization, specifically 82% of the entire turbo decoder area. This oversized memories, when compared with [8], are given by the fact of stored fixed-point variables, such as systematic, *parity1, parity2,* L_e and metrics use a higher bit length. Throughput results are dependent of frequency and parallelization choice, with a trade-off between performance and area/power consumption. Some substantial results can be viewed in Table 4.

Table 3. Performance with different modulations (FER%)

SNR(dB)												
modulation	-2.5	-2	-1	-0.5	0	1	2	3	4	6	8	9
BPSK	3.45	0.95	0	0	0	0	0	0	0	0	0	0
QPSK	72.5	58.05	27.95	16.35	7.7	0.9	0.1	0	0	0	0	0
16QAM	1	1	99.25	98.9	97.45	91.25	76.4	51.25	23.3	0.75	0	0
64QAM	1	1	1	1	1	1	99.2	97	88.9	53.85	10.75	3.95

Table 4. Comparison with other architectures

Work	Map Algorithm	Parallelism	Iterations	Frequency (MHz)	Throughput	Technology	Area (mm^2)	Power (mW)
This	MAX-LOG-MAP	1/8 SISO	6	120	75 Mbps	65 nm	6.7	296.2
[6]	LOG-MAP	32 SISO	6	200	711 Mbps	65 nm	-	-
[7]	MAX-LOG-MAP	4/8 SISO	5.5	302.6	326.4 Mbps	130 nm	3.57	788.9
[8]	MAX-LOG-MAP	64 SISO	6	400	1.28 Gbps	65 nm	8.3	845

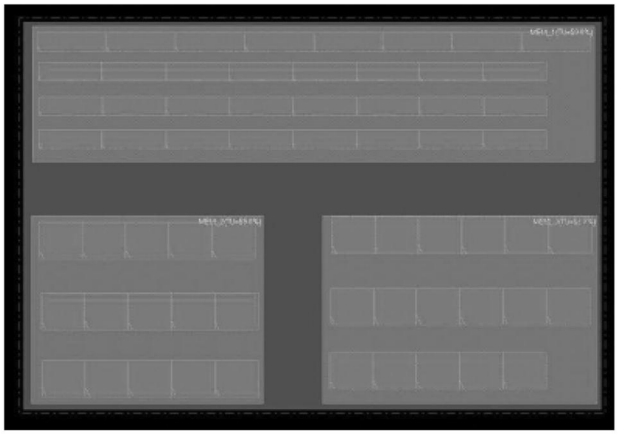

Figure 10. Floorplan view

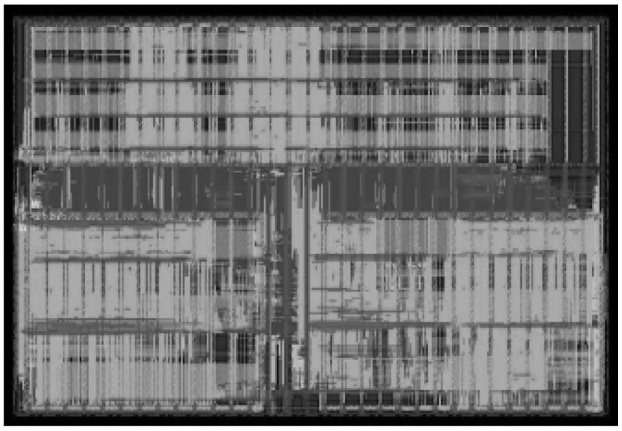

Figure 11. Routed view

4. CONCLUSION

This paper details the implementation of a parallel turbo decoder for LTE standard, analyzing aspects such as modeling, design, verification and physical implementation to TSMC 65nm CMOS technology. The proposed circuit was designed to achieve high throughput. Interleaver and MAP algorithm optimizations in a parallelized architecture made possible achieve high throughput rates, without performance losses. A comparison with other authors must take in account different aspects like frequency and level of parallelization that will result in higher throughput but increasing area and power consumption.

ACKNOWLEDGEMENTS

The authors would like to thank LSI-TEC and Raja Gosula for the support during this work.

REFERENCES

[1] C. Berrou and A. Glavieux. “Near Optimum Error Correcting Coding and Decoding Turbo-Codes” at IEEE Trans. Commun, vol 44, no. 10, Oct. 1996, pp. 1261-71

[2] 3rd Generation Partnership Project, Technical Specification Group (E-UTRA). Multiplexing and Channel (Release 0) 3GPP Organization Partners TS 36.212, Rev 8.3.0, May 2008.

[3] Wong, Cheng-Chi; Lee, Young-Yu; Chang, Hsie-Chia. “A 188-size 2.1mm² reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system” in Symposium on VLSI Circuits, 2009, Kyoto, Japan, pp. 288-89

[4] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. “Optimal decoding of linear codes for minimizing symbol error rate” in IEEE Trans. Inform. Theory, vol. IT-20, pp. 284-287, 1974.

[5] A. J. Viterbi. “An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes” IEEE Journal on Selected Areas in Communications, vol. 16, no 2, pp. 260-264, Februari 1998

[6] Y. SUN, Y. Zhu, M. Goel and J. R. Cavallaro. “Configurable and Scalable High Throughput Turbo Decoder Architecture for Multiple 4G Wireless Standards” in IEEE International Conference on Application-Specific System, Architecture and Processors (ASAP). Pp. 209-214, July 2008.

[7] Christoph Studer, Christian Benkeser, Sandro Belfanti, Qiuting Huang. “Design and implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE”, IEEE Journal of Solid State Circuits, Volume: 46, no. I, January 2011.

[8] Y. Sun and J. R. Cavallaro. “Efficient Hardware Implementation of a Highly-Parallel 3GPP LTE, LTE Advance Turbo Decoder”, Integration, the VLSI Journal, 2012, doi: 10.1016/j.vlsi.2010.07.001.

[9] A. Nimbalker, Y. Blankenship, B. Classon and T. K. Blankenship. “ARQ and QPP Interleavers for LTE Turbo Coding”, Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE, doi: 10.1109/WCNC. 2008.187