Implementation Trade-offs of the Jacobian Logarithm for Several Hardware Platforms

Pablo Ituero Pedro Echeverría Marisa López-Vallejo Carlos López-Barrio
ETSI Telecomunicación (UPM)
Ciudad Universitaria s/n
28040 Madrid, Spain
{pituero,petxebe,marisa,barrio}@die.upm.es

***Abstract*— State-of-the-art communication standards require faster and faster implementations of every detail of the system. This work focuses on the Jacobian Logarithm which is computed in the critical path of current Turbo decoders. Two novel hardware schemes for it are proposed. The analysis covers six scenarios under two different technologies, full-custom and FPGA. More than a 20% of frequency increase is achieved in comparison with previous implementations in full-custom, furthermore future designers are provided with the best choice under different hardware constraints and platforms.**

***Index Terms*— FPGA-based Design, Full-Custom Design, Jacobian Logarithm, Maximum a Posteriori (MAP), Turbo Codes.**

I. Introduction

In the context of 3G communication standards, Turbo codes [1] have gained a lot of interest due to the promise they offer in approaching the Shannon limit. Because of their excellent performance, the third generation wireless mobile system defined in the UMTS standard adopts the Turbo codes as the channel coding scheme for high data rate transmission. The maximum *a posteriori* (MAP) and Soft-Output Viterbi (SOVA) algorithms are used to compute the decoding. Nevertheless, these algorithms have a serious drawback when implemented in hardware: their enormous computational complexity. This is the reason why most implementations perform simplified versions of these algorithms working in the logarithmic domain (e.g. the Log-MAP algorithm [2]).

Moreover, the demand for higher and higher data rates requires extremely optimized hardware implementations. A careful design of every little component is primordial to achieve the desired results. In the case of Turbo decoders, since the algorithms are computed in the logarithmic domain, multiplications become additions, and additions can be computed using the Jacobian Logarithm $(ln(e^a + e^b) \approx max^*(a, b))$ which implies maximization operations with a correction term. Since this correction term only depends on the difference between exponents it can be stored in a one-dimensional lookup table [3]. Nevertheless, the previously introduced max^* operator, which in most cases happens to be in the critical path, is far from being optimized. Previous implementations for this module suppose a big delay for the whole system, being the limiting factor that prevents the decoder from speeding up.

Additionally, the diversity of current hardware platforms set hurdles for the designer to decide which implementation is the best for each of the technologies. Hardware elements that are critical in FPGA-based designs may become less critical in other implementation technologies. For these reasons, it is of great interest the characterization of a small architectural component as is the maximization module from the point of view of different implementation platforms. We have chosen as implementation technologies full-custom and FPGAs because they provide opposite points of the design space: from the best performance at the highest cost of full-custom designs to the highest flexibility and lowest cost of programmable devices. This work introduces two modifications to the maximization module and analyzes several scenarios for both full custom and FPGA implementations. Speed and area are the subject of the study, specifically the delay of the critical path in each case and the area in transistors or slices is provided.

Our analysis will show that a reduction of a 22% in delay is achieved with the new proposals for full-custom designs and will provide the designer with the best choice of the max^* module implementation under different hardware constraints and platforms. Interestingly, full custom and FPGA particularities will demand different optimal solutions.

Little previous work has been carried out specifically targeting this module, in fact most Turbo decoder implementations deal with it as a "black box". In [4] a double LUT structure is proposed to substitute the former ABS-LUT. Apart from the LUT approach, other methods have been proposed to the computation of the Jacobian Logarithm, albeit the come with a significant loose of precision. For instance in [5] a linear approximation to the Jacobi algorithm was introduced, in [6] the lookup table is reduced to just two elements, finally, a shifting [7] approach have also been proposed. FPGA implementations are tackled in [8] and [9]. To the best of our knowledge no previous work deals specifically with this module in a systematic way and targeting several technologies.

The structure of this document is the following. Section II sets the theoretical basis of the Jacobian Logarithm and sets the context where it is used. Section III puts forward the existing implementations of the max^* module as well as our two new proposals. Next section IV elaborates on the implementation issues of the design for each technology, also different analysis scenarios are presented. In section V the results of the various implementations are given. Finally section VI comments these results and draws the conclusions of the work.

This work was funded by the CICYT project TIC2003-07036 of the Spanish Ministry of Education and Science.

II. The Jacobian Logarithm

The high computational cost of certain arithmetic operators, specially the multiplication, has forced the implementation of several algorithms into the logarithm domain. In this domain exponentiations are calculated as multiplications and multiplications as additions; the problem of dealing with additions is solved by means of the Jacobian logarithm. Let $\overline{x}$ denote the natural logarithm of x, $\ln(x)$. We have:

$$\ln(a+b) = \ln(e^{\ln a} + e^{\ln b}) = \ln(e^{\overline{a}} + e^{\overline{b}}) =$$

$$= max(\overline{a}, \overline{b}) + ln\left[1 + e^{(-|\overline{b}-\overline{a}|)}\right] =$$

$$= max(\overline{a}, \overline{b}) + f(|\overline{b} - \overline{a}|) = max^*(\overline{a}, \overline{b}) \qquad (1)$$

As the difference between a and b gets bigger, the term $f(|\overline{b} - \overline{a}|) = ln\left[1 + e^{(-|\overline{b}-\overline{a}|)}\right]$—called correction term—tends to zero, and can be neglected. If so, the additions in the log domain can be computed as maximization operations. When this approximation is not accurate enough to be considered, the addition operation is performed by computing the maximization and the correction term, which is referred to as the max^* operator.

Apart from these two options to compute the Jacobian Logarithm —with and without the correction term— other

The area and specially the delay of this operator has a big impact in current systems because it is normally part of the global critical path that determines the frequency of the whole system. Particularly in the case of Turbo decoders the critical path is most times constituted by add-compare-select-normalize structures [10], [11], which include this computation, furthermore since de the max^* operator must be executed twice for each node in the trellis during each half-iteration, it constitutes a significant, and sometimes dominant, portion of the overall decoder complexity [12].

III. Implementation proposals

Focusing on the implementation of the max^* operator, since the correction term only depends on the difference between a and b, the usual way to deal with this correction function is storing pre-computed values in a lookup table. It has been shown that only very few values need to be stored [3]. The standard implementation, which appears in most papers, is shown in figure 1. Far from being optimal, in a full custom design the critical path of this structure runs through the subtractor, then the multiplexer and finally throughout the adder. Under certain hardware platforms the limiting factor is the delay in the structure ABS-LUT. A double LUT structure that stores values for both positive and negative differences has been also proposed [4], this schema is shown in figure 2. Now the multiplexer dependent on the sign bit of the subtraction (sb) is left after the LUT access which prevents it from being the limiting factor.

The total delay of these two former implementations is constrained at least by two complete carry chains since the last adder cannot start the computation until the sign bit of the subtraction is available. To solve this problem, our first proposal makes use of the structure of the carry-select adder to equalize the delay of the upper and lower paths. This structure is shown in figure 3 and will be referred to as PCS-1 (Pseudo Carry-Select 1). Specifically the LSBs (Least Significant Bits) of the maximization result are added to the output of the LUT and the outgoing carry is used to select either the MSBs (Most Significant Bits) of the maximization result —no carry assumed— or those having been added one —carry assumed. The idea is that the plus-one computation is performed in parallel with the ABS-LUT-addition, the pretension is to balance both paths with a minimum increase in area.

Going one step further and supposing that the plus-one carry chain is the limiting delay, it would be ideal that this computation was performed previous to the first multiplexer. Our second proposal achieves this as shown in figure 4 and will be referred to as PCS-2 (Pseudo Carry-Select 2). In this case both operands are added one so in this case the critical path would necessarily run through the ABS-LUT-addition structure.

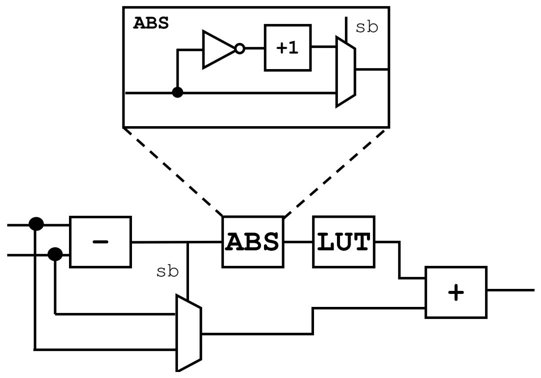


Fig. 1. Straightforward implementation of the max^* operator.

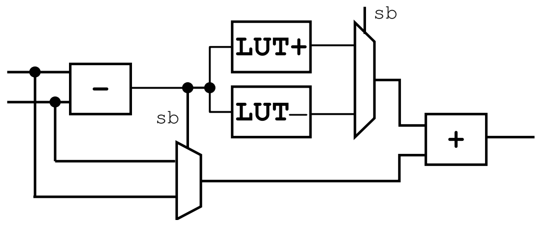


Fig. 2. Double LUT implementation of the max^* operator.

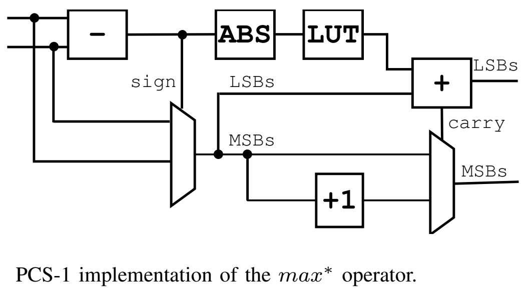


Fig. 3. PCS-1 implementation of the max^* operator.

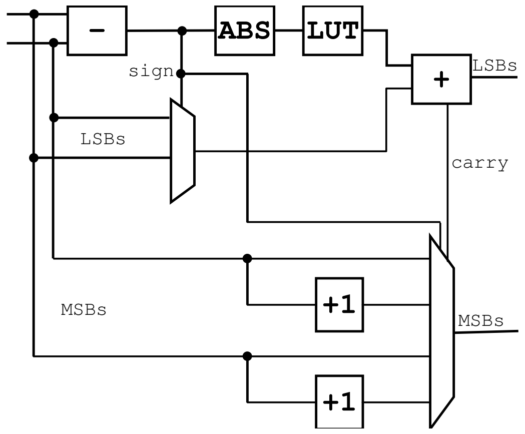


Fig. 4. PCS-2 implementation of the max^* operator.

TABLE I
CONFIGURATIONS OF THE IMPLEMENTATION SCENARIOS.

	1A	1B	1C	2A	2B	2C
Single LUT	X	X	X			
Double LUT				X	X	X
Plain Adder	X			X		
PCS-1		X			X	
PCS-2			X			X

IV. IMPLEMENTATION ISSUES

The purpose of this work is to select the best solution in terms of area and delay for the implementation of the max^* operator. Taking into account that today technologies are diverse and that different hardware platforms can lead to different optimum solutions, we will restrict the scope of the study to full-custom and FPGA technologies.

The full-custom experimental work has been carried out with Spice simulations in the Cadence environment, using a $0.35\mu m$ technology from Austria MicroSystem, dual-poly quadruple-metal CMOS process.

In the case of the FPGA, we have targeted the Xilinx VIRTEX-II (xc2v4000-4), this family of FPGAs is a 0.15 μm / 0.12 μm CMOS 8-layer metal process and is optimized for high speed with low power consumption. Each slice of this FPGA includes two 4-input function generators programmable as a 4-input LUT, 16 bits of distributed RAM, or a 16-bit variable-tap shift register element. The designs are described in VHDL and synthesized targeting low delay using the Xilinx tool ISE 6.2.

Six different scenarios are analyzed in this work, starting from the standard and the double-LUT implementations, they will be then combined with the two proposals presented previously. Table I summarizes the scenarios.

A. *Full Custom Implementation*

In the full-custom implementation, the performance of the different proposed architectures is closely linked to the several adder elements —subtractor, adders and plus-one adders— which determine each scenario. Therefore, the overall performance is conditioned to the base cells of any adder circuit, the full-adder cells and the half-adder cells.

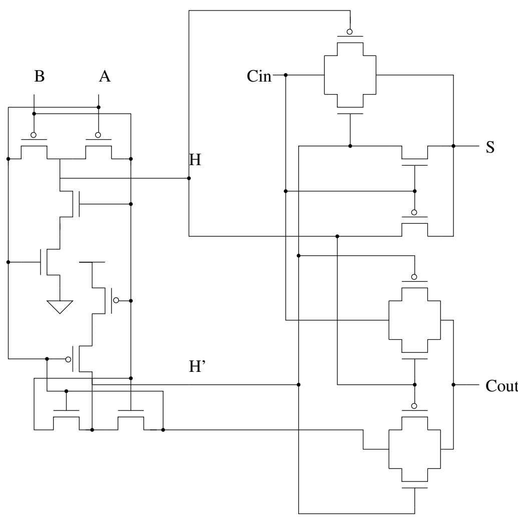


Fig. 5. Selected full-adder.

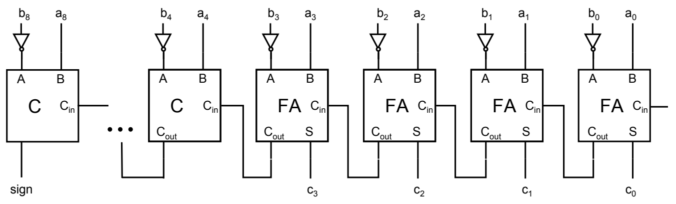
Based on the results of [13], the full-adder of 16 transistors on figure 5 was selected because it presents the fastest carry propagation in combination with a minimum number of transistors, due to the simultaneous generation of the XOR and XNOR gates (H and H' on figure 5). In the same way, it was selected a half adder composed of the same XNOR gate as the full-adder for the carry signal.

Additionally, to the optimal selection of the base cells, an optimization of each adder has been done. In the subtractor it is only needed the resulting four LSBs and the carry out signal, thus a reduced full-adder without the sum generation module is used, see figure 6(a). For the other two adders, a majority of half-adders are used because this module achieves a better speed performance than the full-adder, this is displayed in the figures 6(b) and 6(c).

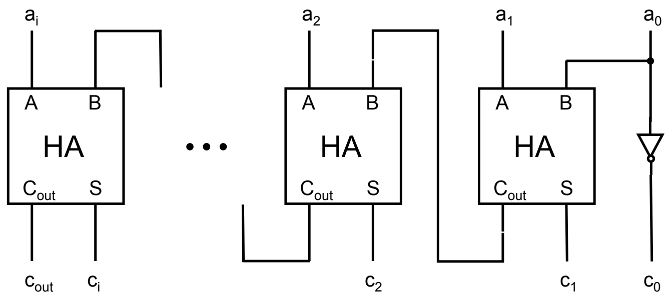
B. *FPGA Implementation*

As far as the FPGA implementation is concerned, the final result was importantly biased by the architecture of the FPGA itself; more precisely the 4-input function generator present at the basic slice determines the minimum mapping of a function that yields one bit, no matter how small it is —e.g. a multiplexor with one bit to select from two one-bit inputs is mapped into one slice, exactly as a much more complex function with four one-bit inputs and a one-bit output. Apart from this, routing has also a big impact in the final result, since in the FPGA connections can entail a considerable percentage of the total delay. As a matter of fact, an implementation that might logically seem faster for a full custom design, might not be so for an FPGA, sometimes adding more logic to execute in parallel could imply increasing the routing of the circuit and therefore mean a delay increase, contrarily to the expected.

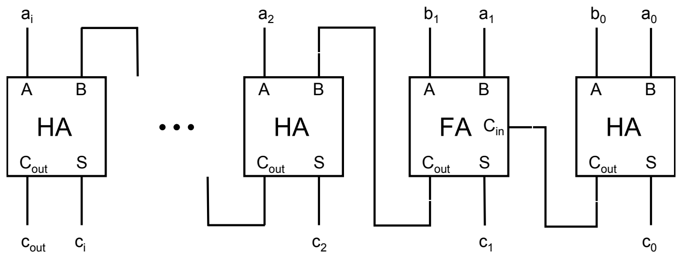
In order to have an accurate measure of a delay in an FPGA, the input and output loads must be carefully taken into account. Hence in our delay evaluations the whole add-compare-select-normalize structure was synthesized.



(a) Subtractor



(b) Adder +1



(c) Adder 9b-2b

Fig. 6. Adders optimizations

V. Results

In this section the results of the implementations under different scenarios are presented. Area and timing figures for both full custom and FPGA are put forward, section VI draws the main comments and conclusions from these results.

A. Full Custom Results

The results obtained in the full custom implementation were very close to our previous predictions. Figures 7 and 8 display the full custom results for the delay and area, respectively. The delay results are given in nanoseconds, whereas the area results show the number of transistors in each scenario.

In the first four scenarios —1A to 2A— the critical path runs through the upper part of design, this means the ABS-LUT structure for the first three and the double LUT structure for the forth. Note that scenario 1C happens to be slower than scenario 1B due to the big delay caused by the last multiplexer in the PCS-2 scheme. In scenarios 2B and 2C the critical path runs through the plus-one adders that are introduced. Concerning area, the results appear to be very sensible according to the structure of each scenario.

B. FPGA Results

In contrast, FPGA results were highly marked by the elevated weight that interconnections delay had in the eventual figures. Figure 9 shows the delay in nanoseconds for all the implementations; the first two data in each scenario split up the total delay in logic and routing delay, respectively. Figure 10 shows the area in slices.

In this case the slowest logic path is less critical than the routing overhead that the FPGA introduces when new logic is added. The percentage of routing delay varies from a 27% in scenario 2B to a 46% in scenario 2C. The increase in logic delay in scenarios 1B and 2B is due to the slow implementation of the output multiplexer. The ABS-LUT structure takes up the same number of slices than the double LUT structure, this explains the coincidence in the results of scenarios 1A-C and 2A-C.

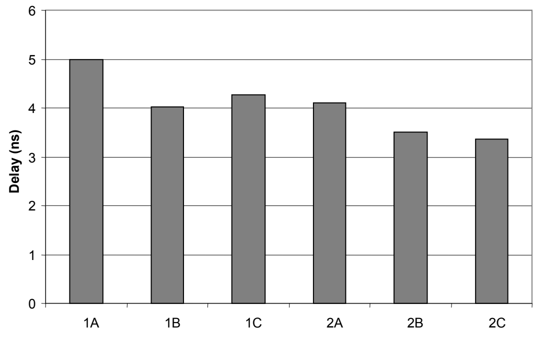


Fig. 7. Full custom delay results.

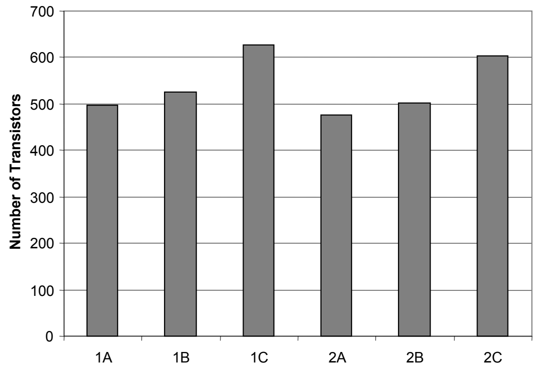


Fig. 8. Full custom area results.

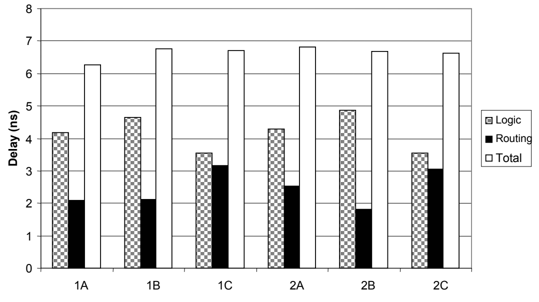


Fig. 9. FPGA delay results.

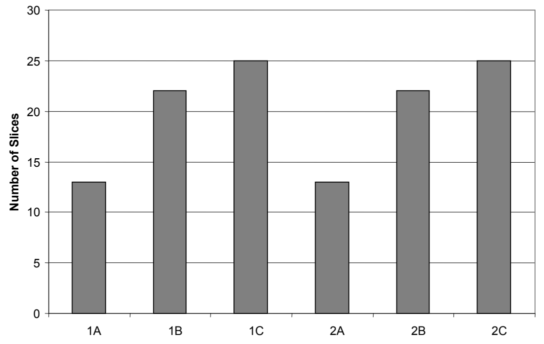


Fig. 10. FPGA area results.

VI. Comments and Conclusions

Interesting ideas are yielded from the analysis of previous results. Firstly the area variation in the different implementations has little impact in a global context where increments of tens of transistors or slices can be neglected. The attempts to improve the delay in the FPGA occurred to be useless basically because of the enormous delay that the routing overhead produced; therefore we recommend to use the straightforward implementation, no matter which constraint is targeted. On the other hand as far as the full custom results in delay are concerned, an important improvement is achieved in comparison with previous implementations; as a mater of fact with respect to scenario 1A, scenario 2C decreases the critical path in 1.7 ns, this means that if this implementation of the max^* operator was embedded in the typical add-compare-select-normalize structure, which generally constitutes the critical path in a Turbo decoder, a 22% of frequency increase would be achieved. Our recommendation is to use this implementation, 2C, independently of the factor that is to be optimized since the area increase is relatively small.

Summarizing the work, two new implementations for the Jacobian Algorithm based on the carry-select adder have been proposed. Six different scenarios have been analyzed for both full custom and FPGA technologies. The proposals have proven to be specially good for full custom implementations where a 22% frequency improvement is attained, whereas the results in FPGA have shown that the routing overhead that the extra logic supposes causes an eventual increase in the delay. The analysis concludes with our suggestions for the designers considering different technologies.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” *IEEE ICC*, May 1993.

[2] M. Valenti, “Iterative detection and decoding for wireless communications,” Ph.D. dissertation, Virginia Polytechnic Institute and State University, July 1999.

[3] P. Robertson and P. Hoeher, “Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding,” *European Trans. on Telecommunication*, no. 8, pp. 119–125, Mar-Apr 1997.

[4] I. A. Al-Mohandes and M. I. Elmasry, “Low-energy design of a 3g-compliant turbo decoder,” in *Circuits and Systems, 2004. NEWCAS 2004. The 2nd Annual IEEE Northeast Workshop on*.

[5] J.-F. Cheng and T. Ottosson, “Linearly approximated log_map algorithms for turbo decoding,” in *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*.

[6] W. Gross and P. Gulak, “Simplified map algorithm suitable for implementation of turbo decoders,” *Electronics Letters*, vol. 34, pp. 1577–1578, Aug 1998.

[7] L. Xi-Zhong, M. Zhi-Gang, Y. Yi-Zheng, and C. Yan-Min, “A simplification of the log-map algorithm for turbo decoding,” in *Circuits and Systems, 2004. Proceedings. The 2004 IEEE Asia-Pacific Conference on*.

[8] S. Sharma, S. Attri, and F. Chauhan, “A simplified and efficient implementation of fpga-based turbo decoder,” in *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*.

[9] R. Marin, A. Garcia Garcia, L. Gonzalez Perez, and J. Gonzalez Villarruel, “Hardware architecture of map algorithm for turbo codes implemented in a fpga,” in *Electronics, Communications and Computers, 2005. CONIELECOMP 2005. Proceedings. 15th International Conference on*.

[10] Z. Wang, Z. Chi, and K. K. Parhi, “Area-efficient high-speed decoding shchemes for turbo decoders,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, no. 6, December 2002.

[11] J. Han, A. T. Erdogan, and T. Arslan, “High speed max-log-map turbo siso decoder implementation using branch metric normalization,” *IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design*, pp. 173–178, 2005.

[12] M. Valenti, “An efficient software radio implementation of the umts turbo codec,” *12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 2, pp. G–108 – G–113, 2001.

[13] A. M. Shams, T. K. Darwish, and M. A. Bayoumi, “Performance analysis of low-power 1-bit cmos full adder cells,” *IEEE Transactions on VLSI*, vol. 10, no. 1, pp. 20–29, February 2002.