Faster Modulo $2^n + 1$ Multipliers without Booth Recoding

Ricardo Chaves
Instituto Superior Técnico / INESC-ID
Portugal, Lisbon
Email: ricardo.chaves@inesc-id.pt

Leonel Sousa
Instituto Superior Técnico / INESC-ID
Portugal, Lisbon
Email: leonel.sousa@inesc-id.pt

***Abstract*— This paper proposes an improvement to the fastest modulo $2^n + 1$ multiplier already published, without Booth recoding. Results show that by manipulating the partial products and modulo reduction terms and by inserting them adequately in the multiplication matrix, the performance of multiplication units can be improved more than 20%. This improvement is obtained at the expense of some extra circuit area, which can be disregarded for operands with sufficient length.**

I. Introduction

Multiplication modulo $2^n + 1$ is widely used, not only in Residue Number Systems (RNS) but also for cryptography, for example in the International Data Encryption Algorithm (IDEA), and systems that use the Fermat Numbers [1]–[3]. Some of the already proposed architectures for $2^n + 1$ multiplication are quite efficient [4]–[8], however they are still slower than the equivalent binary architectures, that in fact can unbalance the performance of some arithmetic systems, for example in RNS arithmetic that performs the $2^n + 1$ moduli operations in parallel with binary channels [3], [4], [9]. Although some of the most efficient architectures use Booth recoding, depending on the technology and system requirements the use of Booth recoding may be disadvantageous [8]. This paper proposes the improvement of the non Booth recoded modulo $2^n + 1$ multiplication architecture proposed by Zimmermann [4], which is one of the fastest known architectures of this type.

This paper is organized as follows. In section II, modulo (2^n+1) multiplication is formatted and expressions are manipulated in order to simplify its computation. Two new modulo $(2^n + 1)$ multiplication architectures are proposed in section III, and their characteristics are analyzed in section IV. Based on experimental results, presented in section V, the proposed multiplication architectures are evaluated and compared with the original architecture proposed by Zimmermann. Section VI concludes the paper.

II. Improved architecture

Unlike multiplication modulo $2^n - 1$ or modulo 2^n, the multiplication modulo $2^n + 1$ has to accommodate the cases where at least one of the operands is equal to 2^n. These cases are one of the main reasons why this unit is the slowest in most RNS moduli sets. In this paper we try to perform this calculation in an efficient way, trying to speedup the computation, without a significant increase in the circuit area required.

Being X and Y the multiplier and the multiplicand, represented with n bits:

$$A = \sum_{i=0}^{n} 2^i a_i = 2^n a_n + \sum_{i=0}^{n-1} 2^i a_i = 2^n a_n + A' \qquad (1)$$

this multiplication can by computed as:

$$\begin{aligned} \langle X \times Y \rangle_{2^n+1} &= \langle 2^n x_n \cdot 2^n y_n + 2^n x_n \cdot Y' + \\ &+ 2^n y_n \cdot X' + X' \cdot Y' \rangle_{2^n+1} . \end{aligned} \qquad (2)$$

By exploring the following properties:

$$\langle 2^n \rangle_{2^n+1} = \langle 2^n + 1 - 1 \rangle_{2^n+1} = \langle -1 \rangle_{2^n+1} \qquad (3)$$

and [3]

$$\langle 2^n A \rangle_{2^n+1} = \left\langle \overline{A} + 2 \right\rangle_{2^n+1} \qquad (4)$$

equation (2) can be written as:

$$\begin{aligned} \langle x_n \cdot y_n - x_n \cdot Y' - y_n \cdot X' + X' \cdot Y' \rangle_{2^n+1} = \\ = \langle P_3 + P_2 + P_1 \rangle_{2^n+1} . \end{aligned} \qquad (5)$$

In modulo $2^n + 1$, when $a_n = 1$ the other bits are zero ($A' = 0$), thus the multiplication can be divided in three distinct operations:

i) for $x_n = 0$ and $y_n = 0$:

$$P = P_1 = \langle X' \cdot Y' \rangle_{2^n+1} \ ; \qquad (6)$$

ii) when one and only one of the nth bits is equal to 1:

$$\begin{aligned} P = P_2 &= \langle 2^n y_n \cdot X' + 2^n x_n \cdot Y' \rangle_{2^n+1} \\ &= \langle y_n \cdot (-X') + x_n \cdot (-Y') \rangle_{2^n+1} \\ &= \left\langle y_n \cdot \overline{X'} + 2 + x_n \cdot \overline{Y'} + 2 \right\rangle_{2^n+1} \\ &= \left\langle y_n \cdot \overline{X'} + x_n \cdot \overline{Y'} + 4 \right\rangle_{2^n+1} \ ; \end{aligned} \qquad (7)$$

*iii)*and finally when both n-th bits of X and Y are 1:

$$P = P_3 = \langle 2^n \cdot 2^n \rangle_{2^n+1} = \langle 1 \rangle_{2^n+1} . \qquad (8)$$

Computation of P_1

To compute equation (6), it can be applied the approach presented in [4]:

$$\begin{aligned} P_1 &= \langle X' \cdot Y' \rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-1} 2^i x_i \cdot Y' \right\rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-1} x_i \cdot \left\langle 2^i Y' \right\rangle_{2^n+1} \right\rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-1} (x_i \cdot y_{n-i-1} \cdots y_0 \, \overline{y}_n \cdots \overline{y}_{n-i} + 1) + 2 \right\rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-1} (PP_i + 1) + 2 \right\rangle_{2^n+1} \end{aligned} \quad (9)$$

Instead of using a multiplication matrix with $n \times n$ bits inputs for adding the partial products (PP_i) followed by a CSA to add the constant 2 of equation (9) [4], the multiplication matrix can have a extra input in which the constant is directly added. This is advantageous when multipliers are implemented with a Wallace tree adders, since for the majority of n values the delay is approximately the same as for $n+1$ inputs: the number of FA in the critical path of a Wallace tree is given by $W(n+1) = \lfloor \frac{3}{2} W(n) \rfloor$ [10]. However this is only a rough approximation, that does not take into account the wire propagation delay, which is becoming increasingly more significant in nowadays technologies.

Computation of P_3

The calculation of P_3 is rather simple if one takes in consideration that when $x_n = y_n = 1$ the partial results P_1 and P_2 are null, and thus the final product is 1. This particular case can be accommodated simply by setting the least significant bit of the multiplication to 1 whenever $x_n = y_n = 1$: this operation can be efficiently performed by a simple OR gate. Since most efficient modulo adders have the result of the LSB calculated before the MSB [3], [4] this OR gate should not have any implication on the computational time.

Computation of P_2

The most significant improvement proposed for the multiplication unit resides in the way the partial product described in equation (7) is introduced in the final result. In [4], Zimmermann uses the fact that when the result of equation (7) is not null equation (6) is zero, and vice-versa, to select one of the results with a multiplexer. This multiplexer is located at the output of the parallel multiplexer itself, being in the critical path of the circuit. We propose to introduce this partial product directly in the multiplication matrix, by adding extra inputs in the Wallace tree. However, for each extra input line added to the matrix of adders (the same as adding an extra modulo 2^n+1 CSA) the final result is incremented [3]. Thus the constant initially introduced in the multiplication matrix as to be corrected in order to take in consideration both these extra inputs and the constant required by the partial product (P_2) in equation (7). The introduction os these extra inputs can be performed in two different way, leading to different architectures.

III. Two different architectures

Taking in to account that in modulo 2^n+1 when the n-th bit is equal to 1 all the other bits are 0, the two elements of equation (7) can never be different from 0 at the same time, thus the addition in equation 7 can be replaced by the logic OR operation:

$$P_2 = \left\langle \left((y_n \cdot \overline{X'}) \; OR \; (x_n \cdot \overline{Y'}) \right) + 4 \right\rangle_{2^n+1}, \quad (10)$$

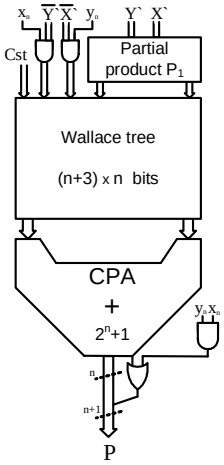
and 2 architectures can be derived to compute the multiplication modulo 2^n+1.

Architecture I: In this architecture the two elements of P_2 are introduced in distinct inputs of the adders matrix, thus 2 extra units are automatically added, resulting in:

$$\begin{aligned} P_1+P_2 &= \left\langle \sum_{i=0}^{n-1} (PP_i + 1)[+2] + y_n \cdot \overline{X}' + x_n \cdot \overline{Y}' + 4 \right\rangle_{2^n+1} = \\ P_1+P_2 &= \left\langle \sum_{i=0}^{n-1} (PP_i + 1) + y_n \cdot \overline{X}' + x_n \cdot \overline{Y}' + 4 \right\rangle_{2^n+1} = \\ P_1+P_2 &= \left\langle \sum_{i=0}^{n-1} (PP_i+1) + (PP_n+1) + (PP_{n+1}+1) + 4 \right\rangle_{2^n+1} = \\ P_1+P_2 &= \left\langle \sum_{i=0}^{n+1} (PP_i + 1) + 4 \right\rangle_{2^n+1}, \end{aligned} \quad (11)$$

with the $[+2]$ constant term being automatically added.

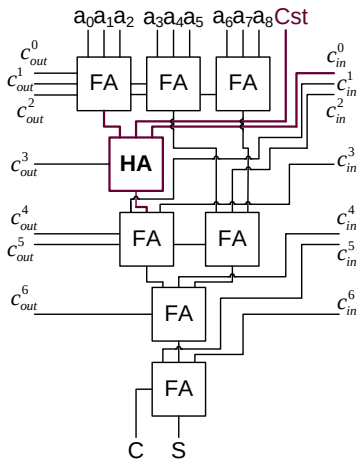
The resulting architecture is represented in figure 1.



1: Architecture I for modulo 2^n+1 multiplication

Note that the constant is introduced in the last input of the Wallace tree, in order to allow a better optimization of the CSA matrix, since a FA with a constant input can be simplified to

a Half Adder (HA), which in some cases can slightly reduce the critical path. To illustrate this, figure 2 describes one of this cases, in a 10 to 2 compressor.



2: Wallace tree optimization

Architecture II: The result of equations (7) and (8) can be computed by selecting the term with the n-th bit equal to 1, resulting in the following four cases:

Case 1 ($x_n = y_n = 0$):

$$\begin{aligned} P_2 &= \langle \overline{y_n X'} + \overline{x_n Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 X'} + \overline{0 Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 \cdots 0} + 1 \cdots 1 + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 \cdots 0} + (-2) + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 \cdots 0} + 2 \rangle_{2^n+1} \quad . \end{aligned} \tag{12}$$

Case 2 ($x_n = 1 \ ; \ y_n = 0$):

$$\begin{aligned} P_2 &= \langle \overline{y_n X'} + \overline{x_n Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 X'} + \overline{x_n Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{x_n Y'} + 1 \cdots 1 + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{x_n Y'} + 2 \rangle_{2^n+1} \quad . \end{aligned} \tag{13}$$

Case 3 ($x_n = 0 \ ; \ y_n = 1$):

$$\begin{aligned} P_2 &= \langle \overline{y_n X'} + \overline{x_n Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{y_n X'} + \overline{0 Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{y_n X'} + 1 \cdots 1 + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{y_n X'} + 2 \rangle_{2^n+1} \quad . \end{aligned} \tag{14}$$

Case 4 ($x_n = 1 \ ; \ y_n = 1$):

$$\begin{aligned} P_2 &= \langle \overline{y_n X'} + \overline{x_n Y'} + 4 \rangle_{2^n+1} = \\ P_2 &= \left\langle \overline{y_n (0 \cdots 0)} + \overline{x_n (0 \cdots 0)} + 4 \right\rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 \cdots 0} + 1 \cdots 1 + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 \cdots 0} + (-2) + 4 \rangle_{2^n+1} = \\ P_2 &= \langle \overline{0 \cdots 0} + 2 \rangle_{2^n+1} \quad . \end{aligned} \tag{15}$$

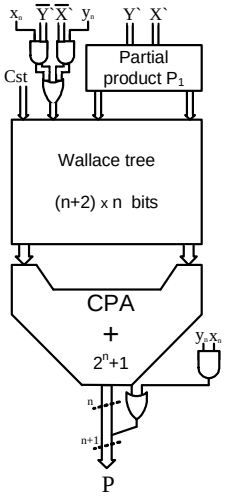
Since $y_n X' \neq 0$ and $x_n Y' \neq 0$ conditions are mutually exclusive conditions, and when $x_n = y_n$ the operation $y_n X' = x_n Y' = (0 \cdots 0)$ the partial product (P_2) can be calculated as:

$$P_2 = \left\langle \overline{(y_n X') \, OR \, (x_n Y')} + 2 \right\rangle_{2^n+1} \quad . \tag{16}$$

This equation is similar to the ones used to calculate the other partial products(PP_i). They only differ by an OR gate in the data path. While the constant of P_2 can simply be added to the already existent constant in the multiplication matrix used to calculate P_1, the remaining part of equation (16) requires an extra input in the (P_1) matrix. The constant value to be added becomes:

$$Cst = \langle (2 + 2) - 1 \rangle_{2^n+1} = \langle 3 \rangle_{2^n+1} \, , \tag{17}$$

As it was mentioned above, this new input in this structure is equivalent to a additional CSA. The proposed architecture II is represented in figure 3.



3: Architecture II for modulo $2^n + 1$ multiplication

IV. Performance Analysis

The Tyagi's theoretical model [11] can be used to evaluate the potential improvements obtained with the new proposed architectures. In this simple model, the time delay is calculated considering that a simple gate introduces one delay unit and interconnection delays are not taken into account. Each of these gates adds one area unit to the total area. The delay and area for the several units that compose the multiplication units are described in tables I and II.

As mentioned before, the main gain in these multiplications architectures lays on the fact that the partial products P_2 are introduced directly in the matrix of adders at the beginning of the calculation. This results in an improvement mostly due to the fact that the Wallace tree calculation time has a logarithmic evolution, given by the number of FA stages [10], as described in table III.

Given that the Wallace tree structure does not significantly improves the calculation time when the number of inputs

I: Theoretical time delay

Unit	Delay
Or/And gate	1
HA /MUX	2
FA	4
Add(n)	$2 \times \lceil \log_2 n \rceil + 6$
WT(n)	$\lfloor \frac{3}{2} \times WT(n-1) \rfloor$
mult model1	$1 + WT(n+3) + Add$
mult model2	$2 + WT(n+2) + Add$
mult ZimN	$1 + WT(n) + 4 + Add$

II: Theoretical area occupation delay

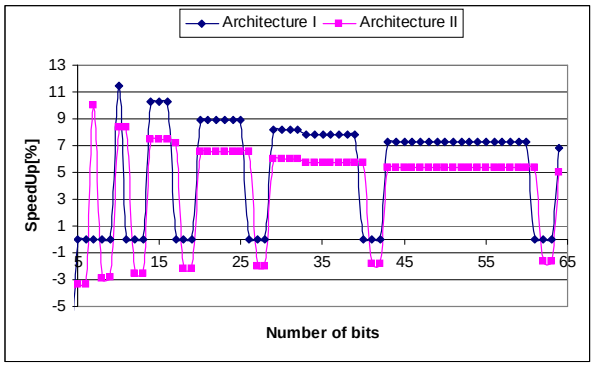
Unit	Area
Or/And gate	1
HA /MUX	3
FA	7
Add(n)	$\frac{3}{2}\lceil \log_2 n \rceil + 8n$
WT(n)	$(n-2) \times n \times FA$
mult model1	$(n+2) \times n + WT(n+3) + Add$
mult model2	$(n+3) \times n + WT(n+2) + Add$
mult ZimN	$n \times n + WT(n) + 6n + Add$

III: Number of FA stages in a Wallace-tree structure.

j	3	4	5-6	7-9	10-13	14-19	20-28	29-42	43-63	...
WT(j)	1	2	3	4	5	6	7	8	9	...

is reduced, it is expected that in average, the proposed multiplication structures may show worst results for these particular cases regarding to the Zimmermann structure. This last structure does not require so many inputs in the matrix of adders.

From this analysis, it can be observed that the proposed units should be slightly faster. In average, the multiplier architecture I requires less 3 to 4 time units and architecture II about 1 time unit less than the traditional architecture (figure 4). It is expected a slight increase in the occupied area, specially for smaller units where the extra input in Wallace tree is more relevant.



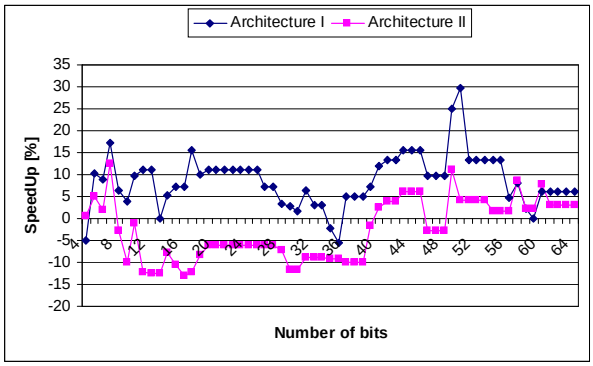
4: Relative theoretical delay in regard to the Zimmermann architecture

The transitions points inflexion points in this figure show the transitions were the Wallace tree data path has an abrupt increase, this type variation does not occur in the classic adders matrix, that has a linear increase.

V. Experimental results

In order to evaluate the real improvement of the proposed modulo 2^n+1 multiplication architectures, these units were described in VHDL and implemented using the design analyzer tool from synopsis and the $0.13\mu m$ standard cell technology from UMC [12].

The graphic in figure 5 depicts the improvements experimentally obtained with the new modulo 2^n+1 multiplication architectures, in regard to the one proposed by Zimmermann in [4]. From these results, it can be observed that the architecture I provides a significant improvement in the delay time regarding the Zimmermann's architecture. However, this improvement is not always present nor constant, depending on the number of bits. For example in the multipliers with 37 bits, the Zimmermann architecture has a better performance, since for this particular dimension the advantage in the Wallace tree is not significant. However, these differences in the computation time are not as abrupt as expected by the Tyagi model, since in practice the interconnection of the several logic gates is not considered by this theoretical model. However it has a significant impact in the delay time and also in the total circuit area, making the delay evolution in the Wallace tree more linear and less logarithmic.



5: Relative experimental delay in regard to the Zimmermann architecture

The new multiplication unit (architecture I) is in average 9% better than the proposed by Zimmermann [4].In fact, for bigger word lengths the speed up obtained by this multiplication architecture is in average above 10%.

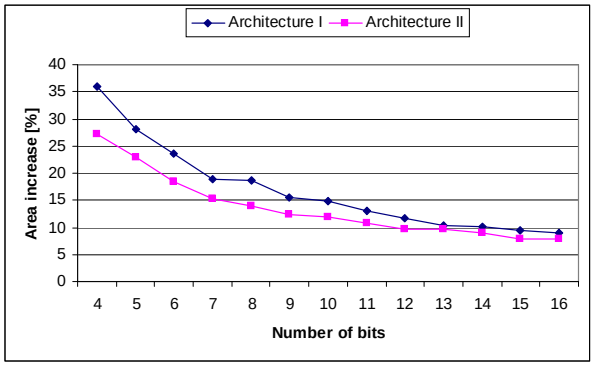
Architecture II is in average worst than the proposed by Zimmermann. However this is not always true with the new multiplication unit becoming faster than the original, for word lengths greater than 40 bits. To better illustrate the evolution of the relation between these units, table IV shows the average speedup of these new units.

IV: Average speedup of the new multiplication architectures.

	SpeedUp [%]	
Number of bits	Arch. I	Arch. II
4 - 64	9	-3
40 - 64	11	3

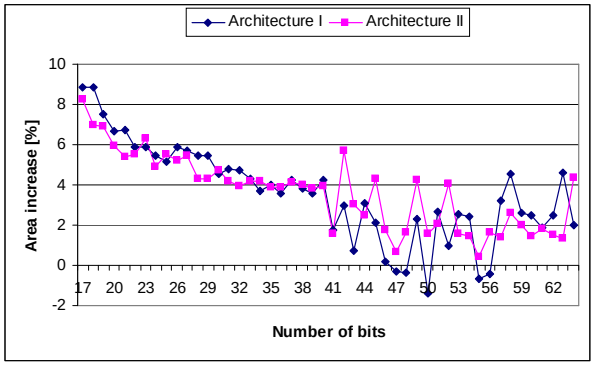
The approach taken to introduce the partial product P_2 in the new multiplication architectures causes an offset in the circuit

area. This is only a significant disadvantage, in comparison to the architecture proposed by Zimmermann, when the number of bits is less than 8, as depicted in figure 6 (the circuit area increases above 20%).



6: Relative circuit area: word length bellow 16 bits

For a number of bits bigger than 16, the circuit area is less than 10% higher, when the speedup is of about 10%, for architecture I. However, the relative area continues decreasing with the increase of the number of bits. The increase in the circuit area becomes approximately 2% for operands with more than 40 bits, as depicted in figure 7, for speedups above 10%.

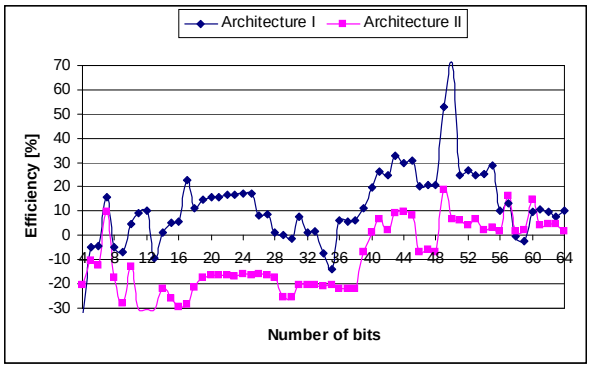


7: Relative circuit area: word length above 16 bits

In order to better analyze the improvements of these new architectures, the efficiency was measured with the product of the circuit area (A) by the square of the time (T) AT^2. The results depicted in figure 8 show a significant improvement in the architecture I, with average values above 20%. As expected, architecture II only becomes more efficient than the architecture proposed by Zimmermann for multipliers with more than 40 bits, and in most cases less efficient than architecture I.

VI. Conclusion

In conclusion this paper presents improved architectures for modulo 2^n+1 multiplication, based on the originally proposed by Zimmermann. One of these architectures is about 10% faster, while the other becomes faster for widths greater than



8: Relative efficiency (AT^2) in regard to the Zimmermann architecture

40 bits. In some cases, even faster than the first proposed architecture. This improvement in the calculation delay has a cost in the area occupation, especially for units with fewer bits. However the area occupation rapidly tends to values closer to those of the Zimmermann architecture, converging to values of area occupation of only 2% higher than the improved architectures.

Using the AT^2 as an efficiency measurement, it can be concluded that for units with more than 9 bits the proposed architecture I is in average more than 20% more efficient than the original architecture. The same occurs for the second architecture, but only for multiplication units with more than 40 bits.

References

[1] P. Mohan, *Residue Number Systems: Algorithms and Architectures, volume 677 of Engineering and Computer Science*. Kluwer Academic, 2002.

[2] M. A. Soderstrand, W. K. Jenkins, G. A. Jullion, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, 1986.

[3] A.S.Ashur, M.K.Ibrahim, and A. Aggoun, "Novel RNS structures for the moduli set $(2^n-1, 2^n, 2^n+1)$ and their application to digital filter implementation," *Signal Processing*, vol. 46, 1995.

[4] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," *Proc. IEEE Symp. on Computer Arithmetic*, pp. 158–167, April 1999.

[5] Z. Wang, G. Jullien, and W. Miller, "An efficient tree architecture for modulo $2^n + 1$ multiplication," *Journal of VLSI Signal Processing*, August 1996.

[6] Y. Ma, "A simplified architecture for modulo $(2^n + 1)$ multiplication," *IEEE Transactions on Computers*, vol. 47, March 1998.

[7] A. V. Curiger and H. Kaeslin, "Regular VLSI architecture for multiplication modulo $(2^n + 1)$," *IEEE Journal of Solid-State Circuits*, vol. 26, July 1991.

[8] L. Sousa and R. Chaves, "A universal architecture for designing effcient modulo 2^n+1 multipliers," *accepted to be published IEEE Transactions on Circuits and Systems: regular papers*, 2005.

[9] R. Chaves, "RDSP: A digital signal processor with suport for residue arithmetic," Master's thesis, Instituto Superior Tecnico, Lisboa, 2003. written in portuguese.

[10] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Computers*, vol. EC, no. 13, pp. 14–17, 1964.

[11] A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Transactions on Computers*, vol. 42, p. 11631170, October 1993.

[12] UMC, *L250 Key Process Features*.