Performance Optimization of Radix-2^m Multipliers Using Carry Save Adders

Marcelo Rosa Fonseca
Eduardo A. C. da Costa
Sergio Bampi
José C. Monteiro

Universidade Católica de Pelotas - UCPel
Universidade Católica de Pelotas - UCPel
Universidade Federal do Rio Grande do Sul - UFRGS
IST / INESC-ID Lisboa

mrf@phantom.ucpel.tche.br
ecosta@atlas.ucpel.tche.br
bampi@inf.ufrgs.br
jcm@inesc-id.pt

ABSTRACT

In this work, we present a new radix-2^m array multiplier architecture using carry save adder (CSA) circuit in the partial product lines in order to speed-up the carry propagation along the array. The new multiplier architecture was recently proposed in [1]. This multiplier uses radix-2^m encoding, which leads to a reduction of the number of partial lines. However, the partial product lines are composed by conventional ripple carry adders (RCA), where the sum for a bit can not be computed before the lower bit carry is computed. In contrast to the ripple carry adders, the CSAs save all the carries from all the adds to the last stage. In our work we present improvements in the performance of the multiplier proposed in [1] by using a faster CSA along the circuit. We compare the multipliers in terms of area, delay and power by using Altera Quartus II tool. Synthesis and simulation of the multipliers are performed for Altera Stratix device. The results we present show that the multiplier proposed in [1] is significantly more efficient if CSAs are used for the partial product lines. Power savings up to 20% are achievable, with slight reduction in area and delay values.

I. INTRODUCTION

One of the major speed enhancement techniques used in modern digital circuits is the ability to add numbers with minimal carry propagation [2]. Carry save adders are one of the most widely used techniques for fast arithmetic in industry [3]. Multipliers and DSP accumulators tend to use carry save adders in their structure so they save all the carries from all the adds to the last stage. This is an important issue, since DSP applications require high computational speed and, at the same time, suffer from stringent power dissipation constraints [4].

In this work, we use carry save adders in the partial product lines of the new multiplier proposed in [1] in order to speed-up the carry propagation along the array. In this multiplier, a new approach is used to handle operands in 2's-complement with exactly the same structure as an array multiplier, with the same unsigned bit products for all the bits except those that involve a sign bit. The regularity of this multiplier makes it suitable for the application of carry save adders, since the ability of it to combine three or four numbers to two, in a time that is independent of the width of the numbers, is a much more efficient alternative than using traditional ripple carry adder.

The multipliers presented in this work were all implemented in Field Programmable Gate Array (FPGA) [5]. We use an Altera Stratix [6] device to compare the multipliers performance. The results show that the multiplier proposed in [1] is significantly more efficient after using carry save adders in the partial product lines. Power savings up to 20% are achievable. The less amount of full adders used in the partial product lines contributes for a slightly reduction in area and delay values. This aspect has a big impact on the reduction of the amount of switch-

ing activity and glitching transitions in the circuit, leading to a power consumption reduction.

This paper is organized as follows. The next section shows the design methodology adopted in this work. In Section III we present the carry save adder structure. Section IV makes an overview of relevant work related to fast multipliers. In Section V we present the proposed modified array architecture to handle signed operands. Performance comparisons between the multipliers are presented in Section VI. Finally, in Section VII we conclude this paper, discussing the main contributions and future work.

II. Design Methodology

There are two basic hardware design methodologies currently available: language based design and schematic based design. Language based design relies upon synthesis tools to implement the desired hardware. As shown in [5], schematic based design methodologies are no longer feasible for supporting the increase in architectural complexity evidence by modern FPGAs. Thus, we have used a language based design methodology as the implementation form for the multiplier architectures with VHDL being the specific language chosen. Basic entities are grouped together using structural VHDL to form bigger building blocks. This hierarchy allows us to adequate the circuit structure to the FPGA architecture.

The multiplier circuits were implemented in the Quartus II Tool from Altera [7]. This tool allows us to create a design using VHDL design languages. The multiplier circuits were synthesized to EP1S10B672C6 device (Stratix). The design flow for the multipliers implementation is shown in Figure 1.

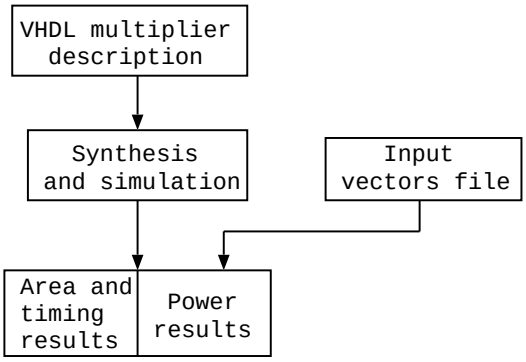

Fig. 1. Design flow for the multipliers implementation.

After verifying the functionality of the multipliers, the VHDL codes are re-simulated with annotated area, timing and power values. For the timing and power values, the same test vectors were used, verifying that the implementations were successful. Both sinusoidal and random vectors were used as input for the power consumption estimation.

III. Carry Save Adders

A carry save adder receives three numbers and partially adds them together. Instead of producing their sum as one number, it produces two numbers which have the same sum. The basic idea is that three numbers can be reduced to 2, in a 3:2 compressor, by doing the addition while keeping the carries and the sum separate. In this case, the key is to align and add the bits with the same significance as shown in the example of the Figure 2 which is presented in [8].

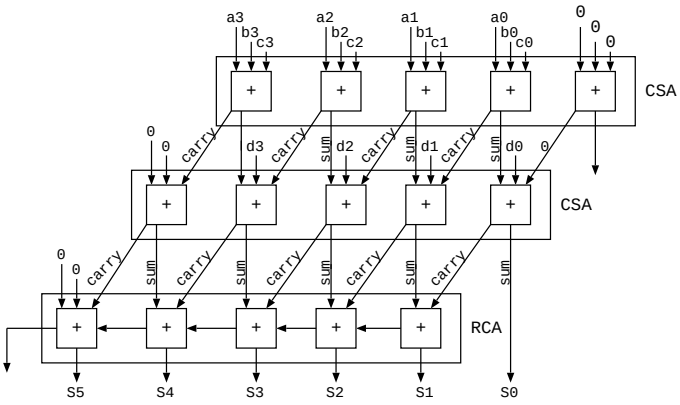

Fig. 2. Example of a carry save structure.

As can be observed in Figure 2, there is no carry propagation within each CSA cell. It is only the final recombination of the final carry and sum requires a carry propagating addition. Figure 3 shows a concrete example of the addition of six numbers using the carry save scheme [9].

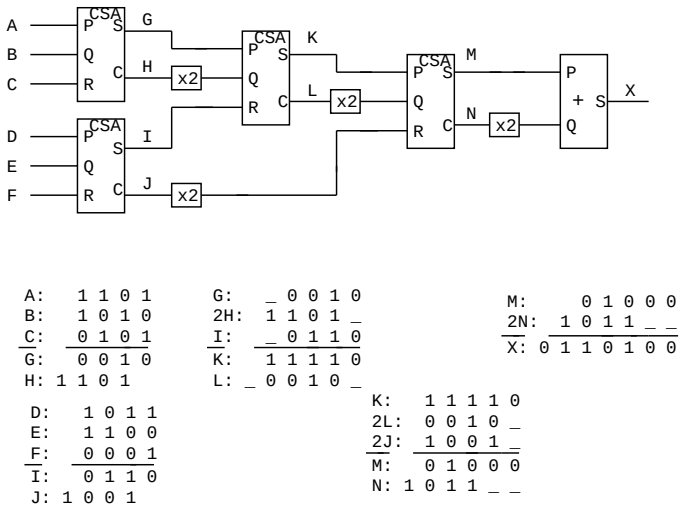

Fig. 3. Example of a 4-bit wide carry save addition.

As shown in Figure 3, a carry save adder (CSA) is shown

as a block with three inputs and two outputs. The full adders are not chained together. Each full adder receives three bits and produces two bits, such that 2×C + S = P + Q + R. In fact, you can treat all the bits put together (S) as a number and (C) as another number so you have just converted the problem of computing P + Q + R to S + 2C without waiting for any carries. The modules `x2` shown in Figure 3 requires no logic, it is only necessary to connect wires appropriately. The final addition M+2N requires a proper adder.

A carry save adder is very fast because it simply outputs the carry bits instead of propagating them to the left. As will be presented in the next section, we apply carry save adders in the partial product lines of an array multiplier circuit in order to speed-up the carry propagation along the array.

IV. Related Work on Fast Multipliers

Various arrangements for providing fast multiplying circuits for use in computer have been proposed [10], [11], [12], [13], [14], [15]. According to [13], probably the single most important advance in improving the speed of multipliers, pioneered by Wallace [10], is the use of carry save adders, to add three or more numbers in a redundant and ripple carry free manner. The Wallace method was improved by Dadda [11].

In our work we present the same principal source as for the Wallace multiplier, the use of carry save adders in order to speed-up the summation of the partial products. Although the Wallace is among the fastest multiplier, it suffers from a bad regularity. Moreover, the multiplying operation is performed bit by bit. In the multiplier we are using in this work, whose architecture was previously proposed in [1], it is possible to obtain improvements in delay and power. This due to the fact that the partial product terms are reduced, because the multiplication is performed by two bits at a time. Besides, the multiplier keeps the regularity of an array multiplier.

The reduction of the number of partial products in order to improve the multiplier performance was already previously presented in the Modified Booth architecture [16], [17], [18]. In [1] it was shown that its proposed multiplier is significantly more efficient than the Booth multiplier, both using ripple carry adders in the partial product lines. Thus, in our work we are presenting performance improvements in the multiplier proposed in [1] by using carry save adders in the partial product terms rather than the conventional ripple carry adder.

V. 2's Complement Multiplier using Carry Save Adders

In this section we summarize the methodology of [1] for the generation of regular structures for arithmetic operators using signed radix-2^m representation. We also show the multiplier using carry save adders in the partial product lines.

A. Radix-2^m Array Multiplier

For the operation of a radix-2^m multiplication, the operands are split into groups of m bits. Each of these groups can be seen as representing a digit in a radix-2^m. Hence, the radix-2^m multiplier architecture follows the basic multiplication operation of numbers represented in radix-2^m. The radix-2^m operation in 2's complement representation is given by Equation 1.

$$\begin{aligned} A \times B &= A' \times B' - A' b_{W-1} b_{\frac{W}{m}-1} 2^{W-m} \\ &\quad - a_{W-1} a_{\frac{W}{m}-1} \sum_{j=0}^{\frac{W}{m}-1} b_j 2^{W-m+j} \end{aligned} \quad (1)$$

This operation is illustrated in Figure 4.

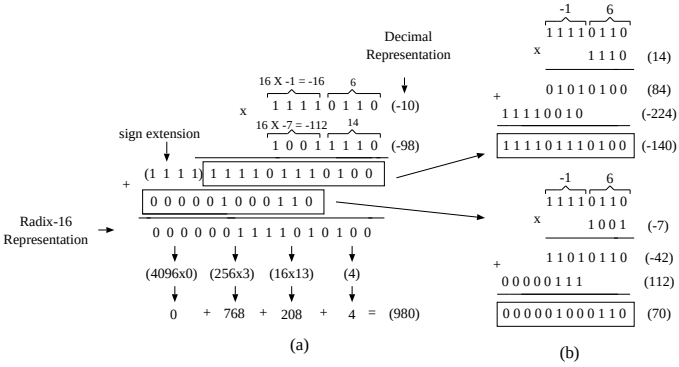

Fig. 4. Example of a 2's complement 8-bit wide radix-16 multiplication.

For the $W - m$ least significant bits of the operands unsigned multiplication can be used. The partial product modules at the left and bottom of the array need to be different to handle the sign of the operands.

For this architecture, three types of modules are needed as shown in Figure 5.

`Type I` are the unsigned modules used in the previous section. `Type II` modules handle the m-bit partial product of an unsigned value with a 2's complement value. Finally, `Type III` modules that operate on two signed values. Only one `Type III` module is required for any type of multiplier, whereas $2\frac{W}{m} - 2$ `Type II` modules and $(\frac{W}{m} - 1)^2$ `Type I` modules are needed. We present

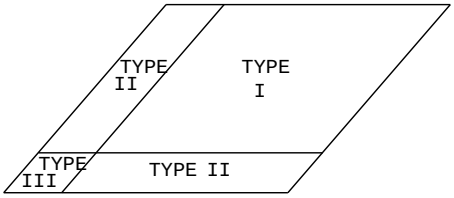

Fig. 5. General structure for a 2's complement radix-2^m multiplier.

a concrete example for $W = 8$ bit wide operands using radix-4 ($m = 2$) in Figure 6.

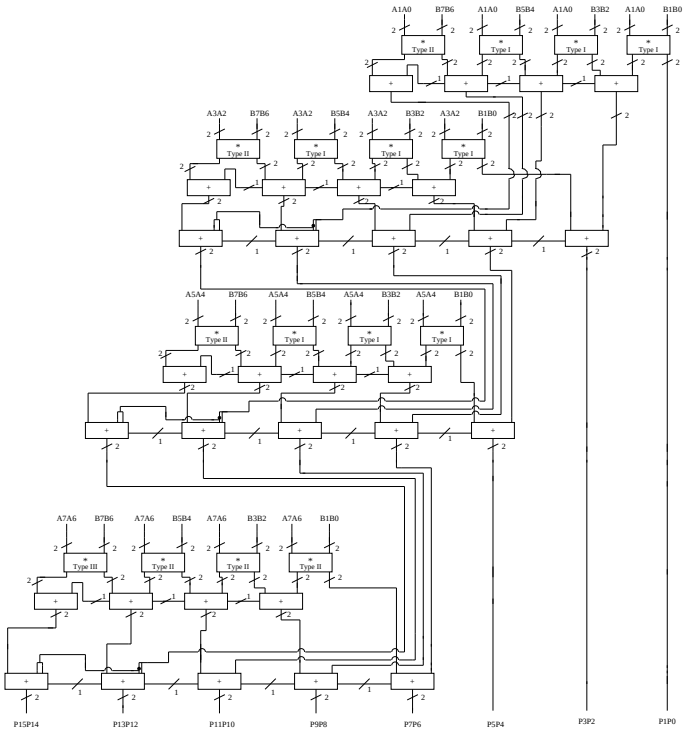

Fig. 6. Example of a 8-bit wide 2's complement radix-4 array multiplier.

B. Radix-2^m Array Multiplier using CSA

The operation of a 2's complement multiplication using carry save adders for the summation of partial product terms, for operators with $W = 6$ bits using radix-4 ($m = 2$), is illustrated in Figure 7. For the example shown, the partial product terms are obtained by multiplying each 2-bit groups of the multiplier and multiplicand terms. Thus, each partial product line is computed by a 2×6 multiplication as depicted in Figure 7. The final product for the radix-2^m multiplication is obtained by adding each 2-bit groups of the partial product terms in a carry save (CSA) form, as shown in Figure 7. It should be observed that only the final recombination of the final carry and sum requires an addition with carry propagation. In the addition with carry propagation, the two most significant bits should be not considered. Note that the two first partial product terms are sign extended up. To make the array more regular, a sign extension technique is used with extra bits in the partial product.

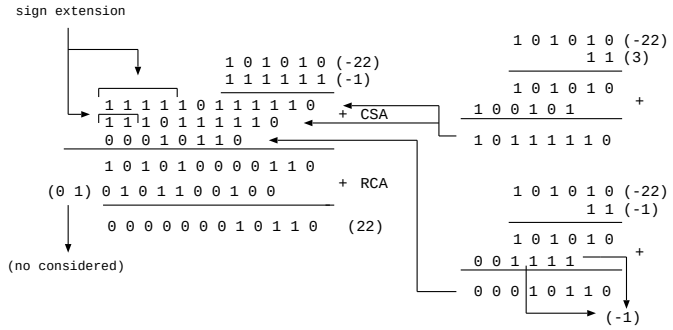

Fig. 7. Example of a 6-bit wide 2's complement multiplication using CSA.

In the multiplier using carry save adders, it is used the same number of partial product lines as for the structure shown in Figure 6 (3 partial product lines for a 8-bit wide example). However, ripple carry modules are only required for the final addition of the adder tree, as can be observed in Figure 8.

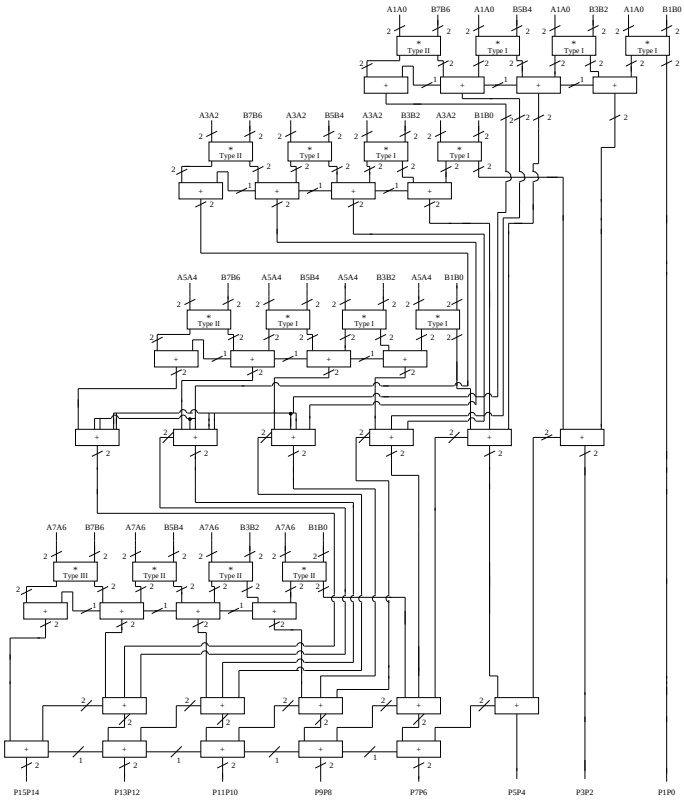

Fig. 8. Example of a 8-bit wide 2's complement radix-4 array multiplier using CSA.

As can be observed in Figure 8, the two first lines of the

partial product terms are grouped in a single line composed by six modules of adders (the sixth module is necessary to add the last adder modules of each operand). Most of these adder modules in the partial product lines are not full adders, but simple half adders that propagates the carry to the next partial product line. This occurs because by applying the basic three input adder in a recursive manner, any number of partial products can be added and reduced to 2 numbers without a ripple carry adder. A single addition using carry propagation is only needed in the final step to reduce the 2 numbers to a single, final product.

VI. Performance Comparisons

In this section, we present results for W=8 and 16-bit multiplier architectures. The multiplier proposed in [1] is compared after using ripple carry and carry save adders along the array. The multiplier used as example is a 2's complement binary array using radix-4 (m=2). Area, delay and power results were obtained in the Quartus-II environment. The multiplier circuits are synthesized to Stratix device from Altera. We synthesize the multipliers using FPGAs because of their flexibility and low cost. Area results are presented in terms of logic cells. Delay results were obtained using the worst delay propagation between the input and output signals. Power results were obtained by using total internal power value. This value represent the addition of total standby internal power and total I/O buffer internal power and total logic element internal power. For the power simulation we have applied a random pattern signal and a *real trace* input signal which represent two sinusoidal signals with 90 degree phase difference, both with 100 input vectors. The multipliers used in this work were easily described in VHDL language. This due to the fact that the regularity of the multipliers contributes for a simple blocks instantiation.

A. Area Results

Although the multipliers using RCA and CSA present the same number of partial product lines, as shown in Figures 6 and 8, it should be observed that the multiplier with CSA uses a smaller amount of full adders than the multiplier using RCA. This due to the fact that by applying the basic three input adder in a recursive manner, any number of partial products can be added and reduced to 2 numbers without a ripple carry adder, as mentioned before. We have observed that for the 8-bit wide example, while we have 12 full adder modules present in the multiplier with RCA, only 9 full adder modules composes the multiplier structure with CSA. Thus, the multiplier using CSA presents a smaller amount of logic elements for 8 and 16-bit wide multipliers, as shown in Table I.

TABLE I
AREA RESULTS FOR 8 AND 16-BIT 2'S COMPLEMENT ARRAY MULTIPLIERS.

Multipliers	Area - Number of Logic Elements		
	with RCA	with CSA	Difference(%)
8-bit	170	162	-4.7
16-bit	776	758	-2.3

B. Delay Results

The reduction of partial product lines is an important issue for performance improvements. As can be observed in Table II m=2 array multiplier using CSA presents a slightly delay improvement for both 8 and 16-bit architectures. This occurs because in the CSA operation all of the columns can be added in parallel without relying on the result of the previous column, creating a two output adder with a time delay that is independent of the size of its inputs.

TABLE II
DELAY RESULTS FOR 8 AND 16-BIT 2'S COMPLEMENT ARRAY MULTIPLIERS.

Multipliers	Delay(ns)		
	with RCA	with CSA	Difference(%)
8-bit	24.09	21.92	-9.0
16-bit	47.11	44.28	-6.0

C. Power Results

As observed in [19], the major sources of power dissipation in multipliers are spurious transitions and logic races that flow through the array. Abstracting implementation details, we can think that an Logic Element in a FPGA is composed of a look-up table (LUT), a register cell, and a multiplexer. Thus, we think that the multiplier using CSA presents a smaller power value, since this multiplier presents a smaller area and their blocks can be easier mapped onto the logic elements in FPGA. The smaller number of interconnections present in the multiplier with CSA is responsible for the reduction of a significant amount of glitching in the circuit which justifies such a large gain in power for our approach as observed in Table III and IV, for both 8 and 16-bit architectures, for random pattern and sinusoidal signals. Moreover, we have observed in this work that the architecture with CSA presents less logic depth due to the more balanced paths to the basic blocks that compose the architecture. This contributes for improvement in power reduction because of the less generation of useless transitions.

TABLE III

POWER RESULTS FOR 8-BIT 2'S COMPLEMENT ARRAY MULTIPLIERS.

Multipliers	Power(mW)- 8-bit multipliers		
	with RCA	with CSA	Difference(%)
Sinusoidal	107.3	87.2	-18.7
Random pattern	186.4	144.8	-22.3

TABLE IV

POWER RESULTS FOR 16-BIT 2'S COMPLEMENT ARRAY MULTIPLIERS.

Multipliers	Power(W)- 16-bit multipliers		
	with RCA	with CSA	Difference(%)
Sinusoidal	2.4	2.1	-12.5
Random pattern	3.6	2.8	-22.2

VII. CONCLUSIONS

We have presented an array architecture multiplier that operates on 2's complement numbers using CSAs in order to speed-up the partial product adder along the array. This multiplier was previously proposed in [1] and uses radix-2^m encoding. The structure of this array maintains the same level of regularity as the normal array multiplier, yet since each partial product handles m bits at a time, the number of levels in the array is reduced by a factor of m. In this work we have experimented the use of CSAs in a $m = 2$ binary array multiplier. We have presented results that show significant improvement in area, delay and power. The results demonstrate that because of our simpler architecture after using CSAs, 2's complement multiplication can be performed with more than 20% of power reduction compared to the multiplier with RCAs.

Although the significant results we have found for $m = 2$ multiplier, after using CSAs, we are convinced that with further work we can improve the modules for $m = 4$ and obtain much better results using radix-16. Such result was previously presented in [1], but using RCA in the partial product lines of the multiplier.

The regularity of the array multiplier architecture make it suitable for applying other reducing power techniques. Thus, as future work we hope test the use of pipelined approach combined with the more efficient CSA in the array multiplier, in order to reduce useless signal transitions that are propagated into the array and the critical path.

REFERENCES

[1] E. Costa, J. Monteiro, and S. Bampi. A New Architecture for Signed Radix 2^m Pure Array Multipliers. In *IEEE International Conference on Computer Design*, pages 112–117, 2002.

[2] G. Knagge. ASIC Design for Signal Processing. In $< http : //www.geoffknage.com/fyp >$, 2003.

[3] T. Kim, W. Jao, and S. Tjiang. Arithmetic Optimization using Carry-Save-Adders. In *35th Design Automation Conference*, pages 433–438, 1998.

[4] E. Mussol and J. Cortadella. Low-Power Array Multipliers with Transition-Retaining Barriers. In *PATMOS*, pages 227–235, 1995.

[5] C. Phillips and K. Hodor. Breaking the 10K FPGA Barrier Calls for an ASIC-Like Design Style. In *Integrated System Design*, 1996.

[6] Altera Corporation. *Programmable Logic Device Family*. Data Sheet, 2002.

[7] Altera Corporation. *Quartus II: System-on-a-Programmable Chip Design Software*. 2002.

[8] A. Sohn. *Computer Architecture - Introduction and Integer Addition*. Computer Science Department - New Jersey Institute of Technology, 2004.

[9] Imperial College of London. *Adder Circuits - Lecture 13*. $< http : //www.ee.imperial.ac.uk/hp/staff/dmb >$, 2004.

[10] C. Wallace. A Suggestion for a Fast Multiplier. *IEEE Transactions on Electronic Computers*, 13:14–17, 1964.

[11] L. Dadda. Some schemes for Parallel Multipliers. *Alta Frequenza*, 34:349–356, 1965.

[12] K. Hwang. *Computer Arithmetic - Principles, Architecture and Design*. School of Electrical Engineering, 1979.

[13] G. Bewick. *Fast Multiplication: Algorithms and Implementation*. Stanford University, 1994.

[14] V. Oklobdzija, D. Villeger, and S. Liu. A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithm Approach. *IEEE Transaction on Computers*, 45(3), 1996.

[15] B. Parhami. *Computer Arithmetic - Algorithms and Hardware Designs*. Oxford University Press, 1999.

[16] W. Gallagher and E. Swartzlander. High Radix Booth Multipliers Using Reduced Area Adder Trees. In *Twenty-Eighth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 545–549, 1994.

[17] B. Cherkauer and E. Friedman. A Hybrid Radix-4/Radix-8 Low Power, High Speed Multiplier Architecture for Wide Bit Widths. In *IEEE International Symposium on Circuits and Systems*, volume 4, pages 53–56, 1996.

[18] P. Seidel, L. McFearin, and D. Matula. Binary Multiplication Radix-32 and Radix-256. In *15th Symp. on Computer Arithmetic*, pages 23–32, 2001.

[19] T. Callaway and E. Swartzlander. Optimizing multipliers for WSI. In *Fifth Annual IEEE International Conference on Wafer Scale Integration*, pages 85–94, 1993.