Design of a Radix-2^m Hybrid Array Multiplier Using Carry Save Adder

Marcelo Fonseca, Eduardo da Costa
UCPel, Pelotas, Brazil
mrf,ecosta@ucpel.tche.br

Sergio Bampi
UFRGS, P. Alegre, Brazil
bampi@inf.ufrgs.br

José Monteiro
IST/INESC, Lisboa, Portugal
jcm@inesc-id.pt

ABSTRACT

In this work, we present a design of a radix-2^m Hybrid array multiplier using Carry Save Adder (CSA) circuit in the partial product lines in order to speed-up the carry propagation along the array. The Hybrid multiplier architecture was previously presented in the literature using Ripple Carry Adders (RCA) in the partial product lines. In our work we present improvements in this multiplier by using a faster CSA along the circuit. The results we present show that the Hybrid architecture with CSA compares favorably in area, performance and power with the architecture with RCA. In this work we also compare the Hybrid multiplier against the Modified Booth multiplier, both using CSA. The results we have obtained show that after using CSA in the partial product lines, the Hybrid multiplier is significantly more efficient than the Modified Booth circuit. Power savings close to 25% are achievable. 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.

I. Introduction

One of the major speed enhancement techniques used in modern digital circuits is the ability to add numbers with minimal carry propagation [1]. Carry save adder is one of the most widely used schemes for fast arithmetic in industry [2]. 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.

In this work we use carry save adder in the partial product lines of a Hybrid multiplier in order to speed-up the carry propagation along the array. In this multiplier, it is used a new approach 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 multiplier uses an Hybrid operand encoding to obtain power efficient arithmetic modules that operate directly on these codes. 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 carry propagation adder.

The multipliers presented in this work were all implemented in Field Programmable Gate Array (FPGA) [3]. We use an Altera Stratix [4] device to compare the multipliers performance. The results we present show that the Hybrid multiplier is significantly more power efficient than the Modified Booth multiplier after using carry save adders in the partial product lines. Power savings up to 20% are achievable. This power reduction is mainly due to the less amount of switching activity and glitching transitions in the circuit.

This paper is organized as follows. The next section shows the design methodology adopted in this work. Section III makes an overview of relevant work related to fast multipliers. In Section IV we present the Hybrid array architecture with CSA. Performance comparisons between the multipliers are presented in Section V. Finally, in Section VI we conclude this paper, discussing the main contributions and future work.

II. Design Methodology

The multiplier circuits were implemented in the Quartus II Tool from Altera [5] and synthesized to EP1S10B672C6 device (Stratix). We have applyied carry save adders in the partial product lines of the array multiplier circuits in order to speed-up the carry propagation along the array. The basic idea of the CSA 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, as shown in the example of the Figure 1 which is presented in [6]. As can be observed in Figure 1, 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.

The design flow for the multipliers implementation is shown in Figure 2. After verifying the funcionality of the multipliers, the VHDL codes are re-simulated with anno-

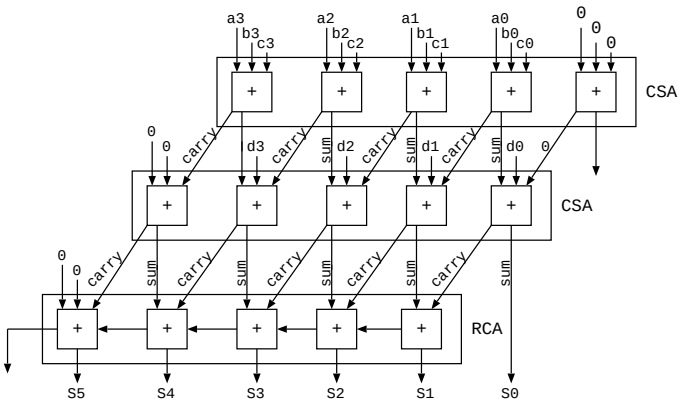

Fig. 1. Example of a carry save structure.

tated area, timing and power values. For the timing and power values, the same test vectors were used, verifying that the implementations were successfull. A file containing a set of sinusoidal and randomic vectors is used as input for the power consumption estimation.

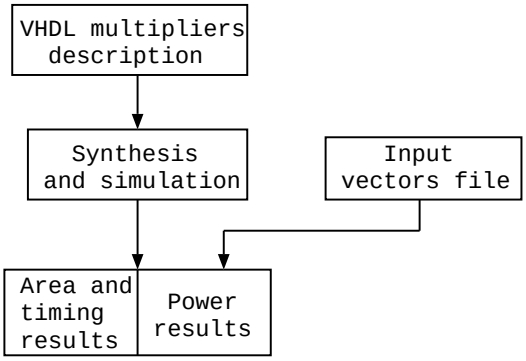

Fig. 2. Design flow for the multipliers implementation

III. Related Work on Fast Multipliers

Various arrangements for providing fast multiplying circuits for use in computer have been proposed [7], [8], [9], [10], [11], [12]. According to [9], probably the single most important advance in improving the speed of multipliers, pioneered by Wallace [7], 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 [8]. 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 suffer from a bad regularity. Moreover, the multiplying operation is performed bit by bit. In the Hybrid multiplier we are using in this work, it is possible to obtain significant power reduction. 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 [13], [14], [15]. In a previous work presented in literature [16], it was shown that the Hybrid multiplier is significantly more efficient than the Booth multiplier in the logic level, both using ripple carry adders in the partial product lines. Thus, in our work we are presenting performance improvements in the Hybrid multiplier by using carry save adders in the partial product terms rather than the coventional ripple carry adder in FPGA platform.

IV. Array Multipliers Using Carry Save Adder

In this section we summarize the methodology of the Hybrid operation for the generation of regular structures for arithmetic operators using signed radix-2^m representation. We also show the Hybrid and Booth multipliers using carry save adders in the partial product lines.

A. Hybrid Operation Overview

The idea of splitting the operands in groups of m-bits and encode each group using the Gray code can be used for operands that operate in 2's complement representation. Table I shows the 2's complement Hybrid encoding for 4-bit numbers and m=2.

TABLE I. 2's Complement Hybrid code representation for m=2.

Dec	Hyb	Dec	Hyb	Dec	Hyb	Dec	Hyb
0	0000	4	0100	-8	1100	-4	1000
1	0001	5	0101	-7	1101	-3	1001
2	0011	6	0111	-6	1111	-2	1011
3	0010	7	0110	-5	1110	-1	1010

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 [16]. This operation is illustrated in Figure 3.

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

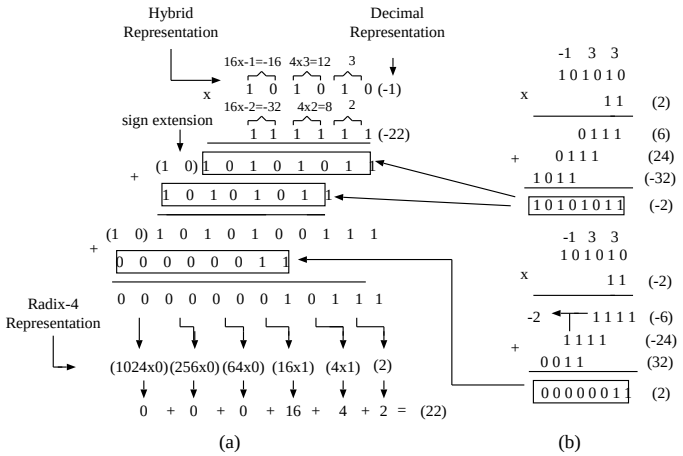

Fig. 3. Example of a 2's complement 6-bit wide radix-16 multiplication.

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. In Figure 3 we can see that thesign extension is represented in Hybrid manner (10 for a negative number).

Note that for the Hybrid architecture, three types of modules are needed. `Type I` is the module for the operation of unsigned values. `Type II` module handles the m-bit partial product of an unsigned value with a 2's complement value. Finally, `Type III` module 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. The Hybrid multiplier architecture using these modules will be presented in the next section.

B. Hybrid Multiplier using CSA

The operation of a 2's complement Hybrid multiplication using CSA for the summation of partial product terms, for operators with $W = 6$ bits using radix-4($m = 2$), is illustrated in Figure 4. 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 4. 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 form, as shown in Figure 4. It should be observed that only the final recombination of the final carry and sum requires a ripple carry addition (RCA). In the RCA operation, the two most significat bits should be not considered. To make the array more regular we use a sign extension technique where two extra bits are used in the partial product. Note that we extend the signal as 10 that represents a negative number (-1) in the Hybrid operation.

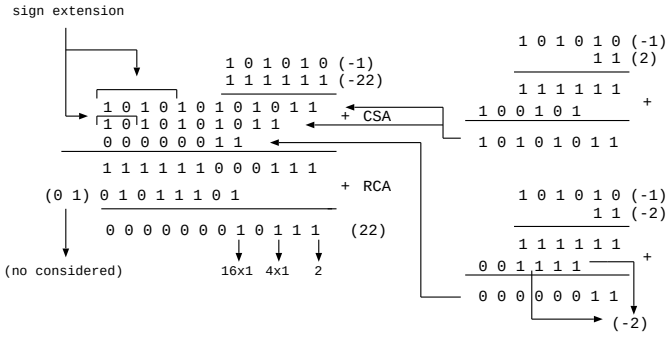

Fig. 4. Example of a 6-bit wide 2's complement Hybrid multiplication using CSA.

In the Hybrid multiplier using CSA, ripple carry modules are only required for the final addition of the adder tree, as can be observed in Figure 5 for a 8-bit wide example. As can be observed in this figure, 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 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 ripple carry addition is only needed in the final step to reduce the 2 numbers to a single, final product.

C. Booth Algorithm Overview

The radix-4 Booth's algorithm (also called Modified Booth) has been presented in [17]. In this architecture it is possible to reduce the number of partial products by encoding the two's complement multiplier. In the circuit the control signals (0,+X,+2X,-X and -2X) are generated from the multiplier operand for each group of 3-b as shown in the example of Fig. 6 for a 8 bits wide operation. A multiplexer produces the partial product according to the encoded control signal. Note that the partial product terms are sign extended up. To make the array more regular a sign extension technique is used where it is used two extra bits in the partial product. It should be observed that the basic Booth cells are not simple adders as in the Hybrid array multiplier, but must perform addition-subtraction-no operation and controlled left-shift of the bits on the multiplicand. Besides taking more area, this complexity also makes it more difficult to increase the radix value in the Booth architecture.

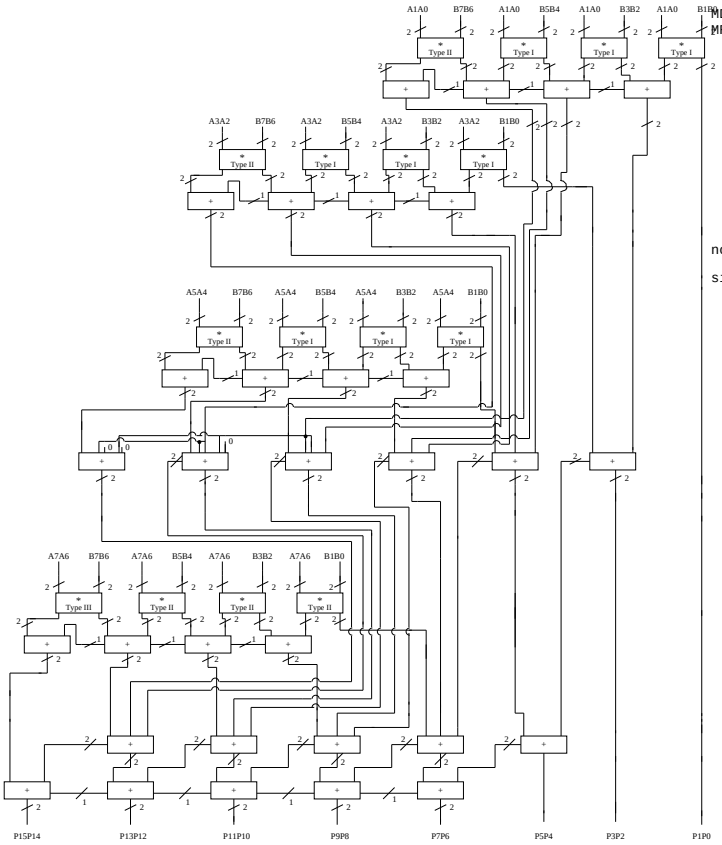

Fig. 5. Example of a 8-bit wide 2's complement radix-4 Hybrid array multiplier using CSA.

```
MD  1 0 1 1 0 1 1 1   (-73)        1 0 1 1 0 1 1 1 0  (2*MD)
MR  0 1 0 1 1 0 1 0(0) (+90)       0 0 1 0 0 1 0 0 1  (-MD)
                                   0 1 0 0 1 0 0 1 0  (-2*MD)

          1 0 0 ->  0 0 0 0 0 0 0 0 0  (PP)
        1 0 1       0 1 0 0 1 0 0 1 0
      0 1 1   ->  0 0 0 1 0 0 1 0 0 1 0 (shift)
    0 1 0         0 0 1 0 0 1 0 0 1
                ->  0 0 0 0 1 1 0 1 1 0 1 1 0 (shift)
                    1 0 1 1 0 1 1 1 0
                ->  1 1 1 1 0 0 0 1 0 0 1 0 1 1 0 (shift)
                    1 1 0 1 1 0 1 1 1
(-6570)         1 1 1 1 0 0 1 1 0 0 1 0 1 0 1 1 0 (shift)
```

Fig. 6. Example of a 8-bit wide Modified Booth multiplication.

D. Modified Booth Multiplier Using CSA

In the Modified Booth algorithm using carry save adders, it is possible to perform the addition of 3 operands simultaneously. In the example shown in Figure 7, for a 8-bit wide multiplication, as a result of the addition of the three first operands, partial sum and carry terms are separately generated. These terms are combined with the fourth operand in order to perform a new carry save addition. The final result is obtained in a ripple carry operation which is performed by the addition of the two last partial sum and carry terms generated in the previous carry save addition. A sign extension technique is used in order to keep the regularity of the operation.

```
MD  1 0 1 1 0 1 1 1     (-73)      1 0 1 1 0 1 1 1 0  (2*MD)
MR  0 1 0 1 1 0 1 0 (0) (+90)      0 0 1 0 0 1 0 0 1  (-MD)
                                   0 1 0 0 1 0 0 1 0  (-2*MD)

          1 0 0  ->  (0)(0)(0)(0)0 1 0 0 1 0 0 1 0
                CSA +(0)(0) 0  0 1 0 0 1 0 0 1
        1 0 1        1  0  1  1 0 1 1 1 0
      0 1 1    ->  (1)(1)(1) 1  0  1  1 1 0 1 0 1 0 1 1 0    (sum)
                CSA + (0) 0  0  0  0  0  0 1 0 1 0 0          (carry)
    0 1 0           (1) 1  1  0  1  1  0 1 1 1               (fourth operand)

                RCA   +  1  0  0  1  0  0  0 1 0 1 0 1 0 1 1 0    (sum)
not considered          ( 0 1) 0  1  0  1  0  1  0 1 0 0 0 0      (carry)
sign extension          1 1  1  1  0  0  1  1 0 0 1 0 1 0 1 1 0  (-6570)
```

Fig. 7. Example of a 8-bit wide Modified Booth multiplication using CSA.

Figure 8 shows the Modified Booth multiplier architecture using carry save adders in the partial product lines. As can be observed in Figure 8, for a 8-bit architecture, 4 operand circuits are necessary to calculate the partial product terms. These circuits are composed by an encoder and a multiplexer which produces the multiplicand term according to the 3-bit in the multiplier term (MR). The product terms are shifted by 3 adders (2 carry save and 1 ripple carry) which are also used to produce the final result. The carry save adders are responsible for adding 3 terms of the partial product simultaneously. The ripple carry adder is responsible for adding the sum and carry terms which are separately generated in the previous carry save addition. .

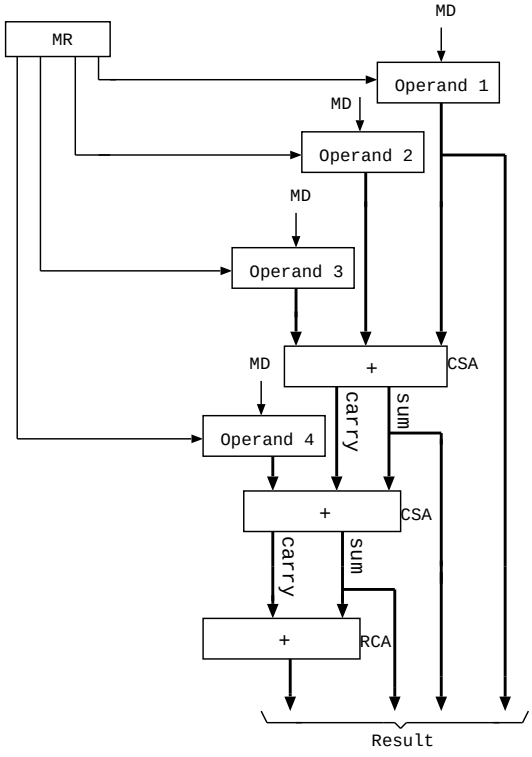

Fig. 8. Example of a 8-bit wide Modified Booth multiplication.

V. Performance Comparisons

In this section we present results for W=16-bit Hybrid and Modified Booth multiplier architectures. The multipliers are compared after using RCA and CSA along the array. Area, delay and power results were obtained in the Quartus-II environment. The multiplier circuits were synthesized to Stratix device from Altera. 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 stadby 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.

A. Area Results

Although the multipliers using RCA and CSA present the same number of partial produc lines, 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. Thus, the multipliers using CSA present a smaler amount of logic elements for 16-bit wide multipliers, as shown in Table II.

TABLE II. Area results for 16-bit 2's complement array multipliers.

Multipliers	Area - Number of Logic Elements		
	with RCA	with CSA	Difference(%) CSA vs.RCA
Booth	513	491	-4.4
Hybrid	666	607	-9.7
Difference(%) Hybrid vs.Booth	+22.9	+19.1	–

Although the Hybrid multiplier presents area reduction when using CSA in the partial product lines, this multiplier presents higher area value as compared against the Modified Booth multiplier, as can be observed in Table II. In fact, this occurs because in the Hybrid multiplier the partial product lines operate on group of m bits and the basic multiplier elements which composes the modules for the product terms are slightly more complex.

B. Delay Results

The reduction of partial product lines is an important issue for performance improvements. As can be observed in Table III the array multipliers using CSA presents a slightly delay improvement compared against the architectures using RCA. 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.

As can also be observed in Table III, the Modified Booth architecture present a slightly improvement compared against the Hybrid multiplier. In fact, in Booth architecture, the partial product selector is composed by a multiplexer that produces the partial product according to the encoded control signal. On the other hand, 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, although the recoding and partial product selector logic used at each level in the Booth algorithm produces a larger number of interconnections and longer delay per row, this algorithm presents a smaller delay value, since their multiplexers can be easier mapped onto the logic elements in FPGA.

TABLE III. Delay results for 16-bit 2's complement array multipliers.

Multipliers	Delay(ns)		
	with RCA	with CSA	Difference(%) CSA vs.RCA
Booth	44.7	38.0	-17.6
Hybrid	51.3	44.0	-16.6
Difference(%) Hybrid vs.Booth	+12.8	+13.6	–

C. Power Results

The Hybrid and Booth multipliers using CSA present a smaller power value, as shown in Table IV and Table V. This occurs because these multipliers present a smaller area, when compared against the multipliers using RCA, and their blocks present a reduction in the switching activity since they can be easier mapped onto the logic elements in FPGA.

As observed in [18], the major sources of power dissipation in multipliers are spurious transitions and logic races that flow through the array. The smaller number of interconnections present in the Hybrid 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 when compared against Booth multiplier, as observed in Table IV and V, for 16-bit architectures using random pattern and sinusoidal signals. Moreover, although the radix-4 Booth multiplier presents a quite rectangular architecture, the regular structure presented by the m=2 Hybrid array multiplier makes him suitable for power reduction. Thus, the combination of regularity characteristic and more balanced paths to the basic blocks that compose the Hybrid multiplier with CSA, contributes for improvement in power reduction in this architecture because of the less generation of useless transistions.

TABLE IV. Power results for the array multipliers with sinusoidal vector.

Multipliers	Power(W)- sinusoidal vector		
	with RCA	with CSA	Difference(%) CSA vs.RCA
Booth	3.8	3.0	-26.6
Hybrid	3.1	2.4	-29.1
Difference(%) Hybrid vs.Booth	-22.6	-25.0	–

TABLE V. Power results for the array multipliers with randomic vector.

Multipliers	Power(W)- random vector		
	with RCA	with CSA	Difference(%) CSA vs.RCA
Booth	5,3	4.3	-23.2
Hybrid	4.3	3.6	-19.4
Difference(%) Hybrid vs.Booth	-23.2	-19.4	–

VI. Conclusions

We have presented a radix-2^m Hybrid array multiplier that operates on 2's complement numbers using carry save adders in order to speed-up the partial product adder along the array. We have shown that the use of CSAs in the partial product lines of the multipliers produces significant improvement in area, delay and power consumption when compared against the multipliers using RCA. In a similar manner we have used CSAs in the well-known Booth architecture. The results demonstrate that although the less area and higher performance presented by the Booth multiplier using CSA, 2's complement Hybrid multiplication can be performed with 25% of the power of a radix-4 Booth multiplier. These results are only based on FPGA platforms (Stratix family). According to our results, increasing the radix can improve the efficiency up to a certain point. Although the best results we have found were for $m = 2$, we are convinced that with further work we can improve the modules for $m = 4$ and obtain much better results using a radix-16 Hybrid multiplier using CSA. Such higher order radices are more difficult to implement with the Booth architecture.

References

[1] G. Knagge. *ASIC Design for Signal Processing*. The University of Newcastle, Australia, 2002.

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

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

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

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

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

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

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

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

[10] 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.

[11] J. Hidalgo and et al. A Radix-8 Multiplier Unit Design for Specific Purpose. In *XIII Conference of Design of Circuits and Integrated Systems*, 1998.

[12] D. Jones. *Fast Arithmetic - Lecture Notes for Spring 2004*. The University of IOWA - Department of Computer Science, 2004.

[13] 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.

[14] 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.

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

[16] E. Costa, S. Bampi, and J. Monteiro. A New Architecture for 2's Complement Gray Encoded Array Multiplier. In *15th SBCCI*, pages 14–19, 2002.

[17] I. Khater, A. Bellaouar, and M. Elmasry. Circuit Techniques for CMOS Low-Power High-Performance Multipliers. *IEEE Journal of Solid-State Circuits*, 31:1535–1546, 1996.

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