Implementations and Comparisons of High-Speed Multipliers for Reconfigurable Devices

Tiago Patrocinio
Federal University of Piauí
Department of Computing
Teresina, Brazil
tiagodsp93@gmail.com

Sinésio Santos da Silva Neto
Federal University of Piauí
Department of Computing
Teresina, Brazil
sn.sinesio@gmail.com

Ivan Saraiva Silva
Federal University of Piauí
Department of Computing
Teresina, Brazil
ivan@ufpi.edu.br

ABSTRACT
Nowadays multipliers are essential components in integrated systems implementations. They have different architectures and implementations techniques that improve performance of the overall architecture. The purpose of this work is propose some architectures for implementing high-speed multiplier for reconfigurable devices. Architectures based on Booth Algorithm and Wallace Tree were developed and analyzed considering performance and area used in the reconfigurable device.

Categories and Subject Descriptors
B.2 [**Arithmetic and logic structure**]: Design Styles—pipeline; B.2 [**Arithmetic and logic structure**]: High-Speed arithmetic; B.2 [**Arithmetic and logic structure**]: General

General Terms
Algorithms, Measurement, Performance, Design.

Keywords
Hardware; arithmetic; multiplier; high-speed;

1. INTRODUCTION
Multipliers are key hardware blocks for performance of integrated systems such as microprocessors, graphics processors, co-processors, among others. Architectures and implementations of these components have a huge impact on logical and arithmetical units (ULAs), which can determine the final performance of the overall integrated system. Eventually efficient improvements on implementation and architectural organization of such components are necessary. Considering this, in this paper we developed a multiplier architecture based on Booth's Algorithm [1] and Wallace Tree [2], which allows obtain a efficient multiplier.

The rest of this paper is organized as follows. The second section presents Booth's Algorithm, Wallace Tree and Adders used to design the multiplier, discussing their implementation advantages. Third section shows performance results obtained from some multipliers designed from variations of their internal components. Section four presents conclusions and future works.

2. ARCHITECTURE COMPONENTS
The multiplier implementation presented in this paper uses the Booth's Algorithm to generate partial products and an Wallace Tree to reduce them. Along this section, the Booth Algorithm and Wallace Tree are briefly presented.

2.1 A. Booth's Algorithm
The Booth's Algorithm, also known as Radix-2 Booth's Algorithm, allows multiplication of positive and negative binary numbers in two's complement, commonly used in multiplier circuit for signed numbers. The version of the algorithm used in this paper is Radix-4 Booth's Algorithm [3]. This version was chosen considering its simplicity and easily deployment, among other advantages that will be discussed soon. The algorithm consists in codifying one multiplication operand, reducing the number of partial products by half, so reducing chip area. The adoption of this algorithm resulted in improvements in multiplication of long numbers.

As presented in Figure 1, Radix-4 Booth Algorithm encodes the multiplier operand, where a zero bit is appended in less significant bit. The operand is analyzed from the less significant to most significant bit, grouped in blocks of three bit, whose the most significant bit of one block overlaps the less significant of the next block.

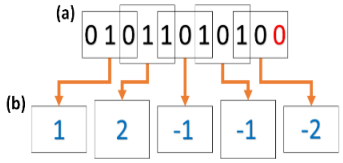

Figure 1. Three bits block division example of the multiplier operand with a zero bit appended in less significant bit (a). Result of codification (b).

Each block of three bits codifies according with Table 1, resulting in multiplication of the multiplicand operand by ±1, ±2 or 0, to generate the partial products, which will be summed posteriorly.

Table 1. Radix-4 partial product generation table.

Block	Partial Product
000	Multiplication of the multiplicand by 0
001	Multiplication of the multiplicand by 1
010	Multiplication of the multiplicand by 1
011	Multiplication of the multiplicand by 2
100	Multiplication of the multiplicand by -2
101	Multiplication of the multiplicand by -1
110	Multiplication of the multiplicand by -1
111	Multiplication of the multiplicand by 0

The improvements of Radix-4 algorithm from Radix-2 is in the recodification way, grouping blocks of three bits, allowing generation of N/2 partial products, where N is the length of operands. In contrast, the Radix-2 algorithm groups 2 Bits blocks, generating N partial products, not being recommended for multiplication of huge numbers.

Another approach is Radix-16 Booth's Algorithm [4], aiming reduce partial products of N-bits multiplication to N/4, therefore reducing subsequent operations.

Figure 2. Radix-16 codification blocks division.

The process codifies blocks of 5 bits, as shown in figure 2, and then analyses according with Table 2, which two combination of bits can generate same partial product.

Table 2. Radix-16 partial product generation table.

Bit Block	Partial Product	Bit Block	Partial Product
00000, 11111	0	11101, 11110	-1 * MPLD
00001, 00010	+1 * MPLD	11011, 11100	-2 * MPLD
00011, 00100	+2 * MPLD	11001, 11010	-3 * MPLD
00101, 00110	+3 * MPLD	10111, 11000	-4 * MPLD
00111, 01000	+4 * MPLD	10101, 10110	-5 * MPLD
01001, 01010	+5 * MPLD	10011, 10100	-6 * MPLD
01011, 01100	+6 * MPLD	10001, 10010	-7 * MPLD
01101, 01110	+7 * MPLD	10000	-8 * MPLD
01111	+8 * MPLD		

Figure 3 shows the general diagram of Booth's Algorithm used to design the multiplier.

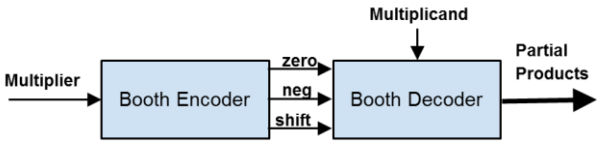

Figure 3. General structure architecture diagram of Booth's Algorithm.

2.2 Wallace Tree

Multiplication circuits need to perform successive additions of partial products, generating dependencies between operations. The Wallace Tree offers an efficient structure for implement parallel faster additions of the partial products. It avoids carry propagation, requiring only an addition at the end of tree to obtain the result.

Compressors [5] are the basic blocks used in the implementation of the Wallace Tree. The use of compressors contributes to decrease the amount of additions in a multiplication. In this paper 4:2 Compressors [5] are used, reducing groups of four partial products. 4:2 Compressors have five inputs A, B, C, D and CIN to generate three outputs SUM, CARRY and COUT, as shown in figure 4.

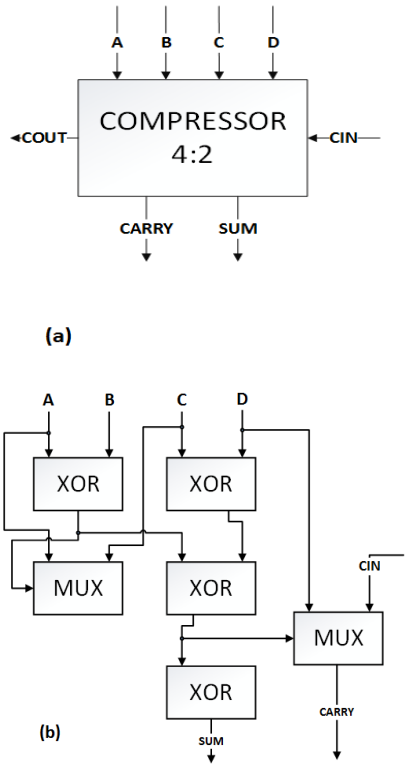

Figure 4. 4:2 Compressor structure (a). Logic architecture (b).

As shown in figure 5, the organization of the Wallace Tree with 4:2 compressors groups four partial products in parallel to perform the compression at each level of the tree. The output of each compressor becomes an input at next level, until all partial products become reduced to two inputs for addition at the end of tree.

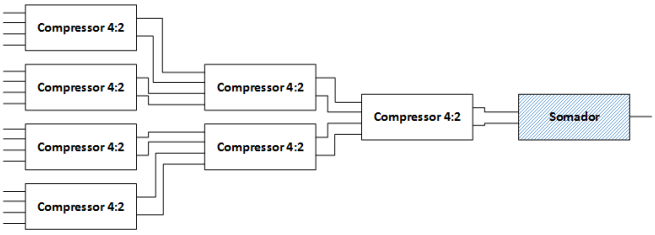

Figure 5. Wallace Tree structure with 4:2 compressors.

The implementation of the Wallace Tree in pipelined way [6] aims to improve the frequency of operation. Moreover it allows performing successive faster multiplications, adding registers between tree's levels.

2.3 Adders

Efficient implementation of the adder induces huge impact in the multiplier performance. A traditional adder is the Carry-Propagate Adder [7]. It has a simple architecture, but offers inefficient performance, due carry propagation from less significant bit to most significant.

A bit more complex adder's architecture as Carry Look Ahead Adder [7] (figure 6) allows forwarding carry due the addition of extra signals that calculates the Generation and Propagation signals (figure 7) [7]. The Carry Look Ahead Adder reduces the critical path and improves the operation speed.

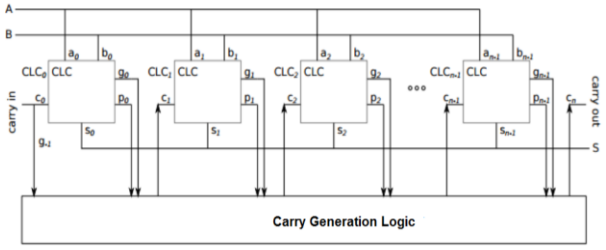

Figure 6. Carry Look Ahead Adder architecture

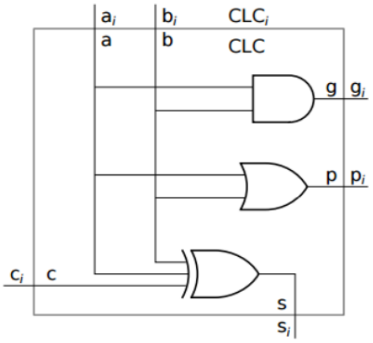

Figure 7. Carry Look ahead Cell (CLC) with generation and propagation carry signals.

3. Architectures, Results and Comparison

To design the multiplier, various architectural organizations are available. Each one has significant impact in the overall performance. This paper presents some architectures, their performance comparison and discuss about their benefits.

The multipliers architectures designed in this paper have Wallace Tree with and without pipeline, both using Radix-4 and Radix-16 Booth`s Algorithm to generate partial products. Each type of architecture implements different type of adders in Wallace Tree. In addition of Carry-Propagate Adder and Carry Look Ahead Adder, was included the Altera Adder. Figure 8 shows the general architecture diagram.

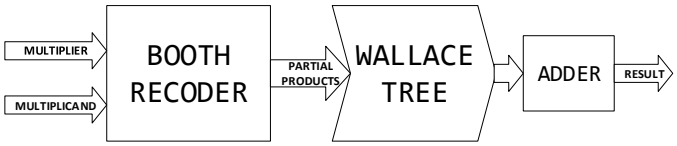

Figure 8. Multiplier general structure diagram.

Table 3 shows obtained results from 32 Bits Multipliers with Radix-4 Booth's Algorithm. Results also include the Altera Multiplier only for performance comparison.

Pipelined multipliers have high frequency, but needs more cycles to produce results. Considering the need of successive multiplications, results can be producer successively after the first product. This fact is nullified when few operations are requested in a determined period of time. Architectures without pipeline are slower, but present result in one cycle. The of such implementation presents advantages over the pipelined one if the number of operations do not exceeds the pipeline time to produce the same number of results. Figure 9 compares implementations with and without pipeline showing the number of successive operation from which using pipeline is advantageous.

Table 3. Frequency and chip area comparison.

Architecture	Wallace Tree	Adder	Frequency		Chip Area
			Fmax (Slow 1200mV 85C Model)	Fmax (Slow 1200mV 0C Model)	
Developed in this study	Without Pipeline	Carry-Propagate Adder	37.54 MHz	41.72 MHz	2,468 LE
		Carry Look Ahead Adder	52.44 MHz	58.10 MHz	2,416 LE
		Altera Adder	90.52 MHz	102.16 MHz	2,655 LE
	With Pipeline	Carry-Propagate Adder	56.82 MHz	64.02 MHz	2,524 LE
		Carry Look Ahead Adder	149.37 MHz	167.95 MHz	2,432 LE
		Altera Adder	205.63 MHz	235.13 MHz	2,325 LE
Altera Multiplier	-	-	79.33 MHz	89.40 MHz	1,409 LE

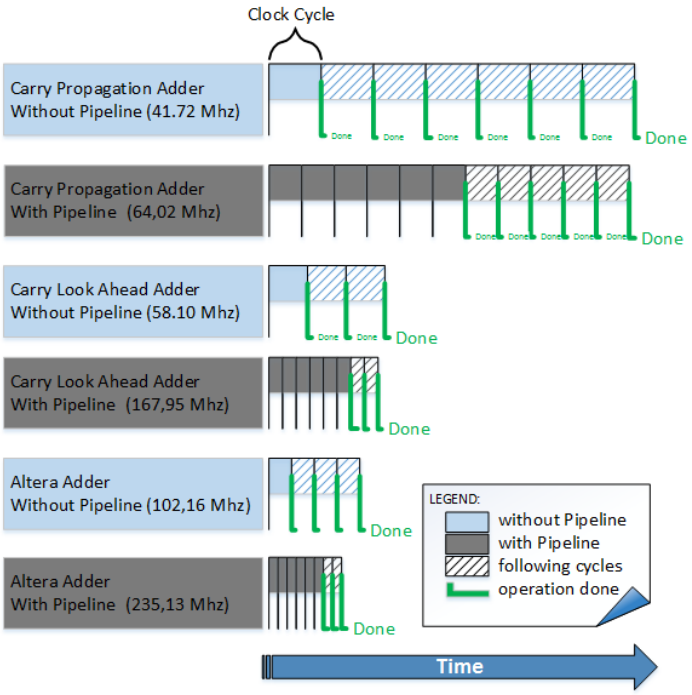

Figure 9. Cycles representation of the multipliers applied successive operations.

Table 4 shows the number of multiplications without pipeline recommended until exceeds the pipelined time, representing advantages regarding the number of results.

Table 4. Recommended successive operations number without pipeline for each architecture.

Adder Type	Recommended Multiplications (Without Pipeline)
Carry-Propagate Adder	6
Carry Look Ahead Adder	2
Altera Adder	3

The Table 5 show frequency and chip area of pipelined multipliers with Radix-4 and Radix-16 Booth's Algorithm.

Table 5. Performance data of 32-Bits Radix-4 and Radix-16 Multipliers.

Architecture	Booth's Algorithm	Adder	Frequency	Chip Area
Multiplier with Pipeline	Radix-4	Carry-Propagate Adder	64.02 MHz	2524 LE
		Carry Look Ahead Adder	167.95 MHz	2432 LE
		Altera Adder	235.13 MHz	2325 LE
	Radix-16	Carry-Propagate Adder	89.45 MHz	4614 LE
		Carry Look Ahead Adder	174.43 MHz	4717 LE
		Altera Adder	255.98 MHz	4557 LE

Table 6 shows the total cycles of each multiplier affected by Booth's Algorithm approached in this study and its Wallace Tree level amount affected by partial products reduction.

Table 6. Wallace Tree levels and necessary cycles of Radix-4 and Radix-16 Booth's Algorithm Multipliers

	32-Bits Multiplier	
	Wallace Tree levels	Necessary Cycles
Radix-4 Booth Algorithm	3	6
Radix-16 Booth Algorithm	2	5

As shown in table 7, switch Radix-4 to Radix-16 improves frequency, even for a little increment. However, chip area nearly doubles due to increase complexity of the logic in Radix-16 codification.

4. CONCUSION

In this paper faster multipliers were implemented, performance comparisons were presented and the impacts of the components architecture to improve performance were discussed.

The results presented that pipelined multipliers are most effectives when subjected to successive operations. Allowing high amount of result outputs in less time. Also presented that multipliers without pipeline have acceptable performance by cycle, being a feasible solution to application that do not need successive operations.

As shown in comparisons, pipelined multipliers with alternatives Booth's Algorithm implementation have an improvement in performance. However having as trade-off an impact in chip area due the complexity of partial products generation.

In future works this study will assist deployment of high-speed multipliers in many core processor architectures, aiming the improvement and assess its performance in various applications.

Table 7. Radix-4 to Radix-16 Comparison (Pipelined Multiplier).

Radix-4 to Radix-16 Comparison (Pipelined Multiplier).				
Adder Type	Frequency Gain	Frequency Gain (%)	Chip Area increase	Chip Area increase (%)
Carry-Propagate Adder	25.43 MHz	39,72%	2090 LE	82,81%
Carry Look Ahead Adder	6.48 MHz	3,86%	2285 LE	93,96%
Altera Adder	20.85 MHz	8,87%	2232 LE	96,00%

5. REFERENCES

[1] Collin, A. Andrew Booth's Computers at Birkbeck College. Resurrection, Issue 5, Spring 1993. London: Computer Conservation Society.

[2] Wallace, C. S. A suggestion for a fast multiplier, IEEE Trans. on Electronic Comp. EC-13(1): 14-17 (1964)

[3] Surendran, E. K. L. and Anthony, P. R. Implementation of fast multiplier using modified Radix-4 booth algorithm with redundant binary adder for low energy applications. IEEE 2014 First International Conference on Computational Systems and Communications, p.266-271, 2014.

[4] Pohane, G. and Sharma, S. Review Paper on High Speed Parallel Multiplier – Accumulator (MAC) Based on Radix-4 Modified Booth Algorithm. International Journal of Application or Innovation in Engineering & Management (IJAIEM) vol. 3, pp. 86-95, Nov. 2014.

[5] Tonfat, J., Reis, R. Low Power 3-2 and 4-2 Adder Compressors Implemented Using ASTRAN. IEEE Third Latin American Symposium on Circuits and Systems (LASCAS), 2012.

[6] Pang, K.F. Architectures for pipelined Wallace tree multiplier-accumulators. IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1990.

[7] Weste, N. and Harris, D. CMOS VLSI Design: A Circuits and Systems Perspective. Boston: Pearson Education, 2011. Ed.4, p.429-261.