Exploring Parallel Prefix Adders in Optimized Squared Array Multiplier

Morgana Macedo[†], Leandro Rocha[*], Guilherme Paim[*], Eduardo A. C. da Costa[†]
, Sergio Bampi[*]
[†]Graduate Program on Electronic Engineering and Computing - Catholic University of Pelotas (UCPel), Pelotas, Brazil
[*]Graduate Program on Microelectronics (PGMicro) - Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

***Abstract*—This work explores the use of Parallel Prefix adders (PPA) in an optimized squared array multiplier. The optimized squared multiplier was implemented so that only the required sum trees are used. It is possible because in this type of circuit some multiplications are duplicated, and thus some of them can be discarded. Therefore, the use of PPA reduces still more the complexity required by the sums of the partial product lines. All the architectures of this work were described in Very High Speed Integrated Circuits Hardware Description Language (VHDL), an synthesized using the ST 65nm cell library. Results show that the quadratic multiplier cells using the Kogge Stone adder save up to 3.5% in power dissipation and 9.4% in cell area when compared to other 16-bit PPA-based multipliers.**

Keywords—Squared multipliers, Parallel Prefix adders, VHDL.

I. Introduction

Multiplier circuits are useful in applications that use multiplication and accumulation (MAC) operations, as well as in Digital Signal Processing (DSP) algorithms. On the other hand, in some applications, such as logarithm [1] and exponential [2] functions implemented in hardware, the use of a squared multiplier is mandatory. Anyway, the multipliers are always the main factor that contributes to the power consumption. One of the reasons for this high power dissipation is the large number of partial product lines, where the sums are realized in parallel. In this work, we explore efficient parallel prefix adders (PPA) in the adders trees of the squared multiplier, named Brent-Kung [3], Kogge Stone [4], Han-Carlson [5], Ladner-Ficher [6], and Skanskly [7].

Through a careful analysis, we were able to reduce the complexity of the squared multiplier by using only the necessary additions in the partial product lines. This is possible because most of the multiplications operations use the same inputs, and thus are only realized one time. Therefore, the use of efficient PPAs in the partial product lines can become the square multipliers still more efficient. Using these parallel prefix adders to reduce the sum trees, it was possible to save 62.5% of the total logic used in the squared multiplier. The rest of this paper is organized as follows: Section II give a background on squared multipliers and Section III explains the architecture of the parallel prefix adder overview. The synthesis methodology and the experimental results are given in Section IV, and finally, Section V concludes the paper.

II. Squared Multiplier

Multiplication is a complex operation that can be performed in two main ways: through dedicated logic circuits, called multipliers, or by a sequence of sum and displacement instructions [8]. The most basic form of multiplication consists of forming the product of two unmarked binary numbers, this can be done through the traditional technique taught in primary school [9].

The quadratic arithmetic operation is a critical operation in computer systems as it is used in image compression algorithms, cryptography, digital signal processing, and so on. This operation can be performed using a traditional multiplier, but its hardware implementation can be optimized if a custom circuit is built. As both operands are equal, we can avoid the generation of many partial products, eliminating redundant bits, resulting in a simpler circuit with less circuit area, smaller critical path and reduced energy consumption.

Our work proposes an efficient algorithm to build custom squared multipliers through the optimization of the logic equations according to the literal reduction techniques. In the squared multiplier, many multiplications repeat, as we have two equal values that multiply, so many simplifications can be made.

The logic reduction is justified as the multiplication has two equal values, so, for example, the multiplication of A0.A0, it can is simplified to A0. Another simplification occurs when there is two terms A1.A0 to be summed, it is only necessary to store the value of A1.A0 and, then, to perform a left-shift, as shown in Fig. 1.

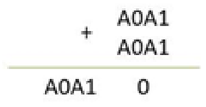

Fig. 1. Reduction of square multiplier logic

This technique of reducing the logic of the multiplier squared was discovered through a lot of study, where we realized that in the same way that we add 1 + 1 in binary we have 10 (2 in decimal), so we would need to consider only one bit and move it, of counter technique, because according to the number of similar logics that we count will be the form of the storage of the logic and the displacement.

The quadrature unit requires half of the partial products, as we can combine using the equivalences ($A_iA_j + A_iA_j = 2A_iA_j$) which can be represented by the addition of (A_iA_j) with a left shift.Reducing depth that can be defined as the number of partial products to be added together in each column.

In figure 2 we have a reduction of the logical depth and also of the critical path, where we can observe that the two terms A1A0 are reduced to only one term with a left shift,

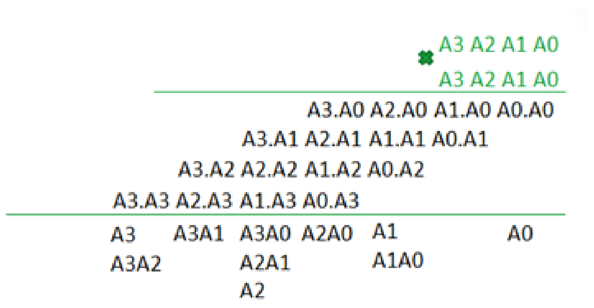

Fig. 2. Reducing the critical path of the square multiplier

the property of A0A0 can be reduced to A0, since we have a AND gate of equal values, so if both values are zero the output is zero, and if both are one the output will be at logical level one.

Our technique compared to general purpose multipliers for the calculation of squared, reduces the number of logical synthesis by 62.5%, that is, it reduces redundant logic.

III. Parallel Prefix Adders Overview

The most basic circuit to perform the addition of two operands is the Ripple Carry Adder (RCA). This adder is composed by a cascade of full adder blocks. Although simple structure forms the RCA, it does not present such a good performance when compared with other structures shown in the literature.

Various fast adder circuits have been proposed in the literature to speed-up the carry calculation because this term represents a bottleneck for the performance in an RCA structure. When high performance is demanded, Parallel Prefix Adders (PPA) appear as the best choice [10]. They rely on the use of simple cells and keeps a regular connection between them. The prefix structures allow several tradeoffs between i) the number of cells used, ii) the number of required logic levels, and iii) the cells fan-out [10].

A parallel prefix circuit computes N outputs from N inputs using an associative operator. Most prefix computation is precomputed from intermediate variables of the inputs. The prefix network combines these intermediate variables to form the prefixes. The outputs are post computed from the inputs and the prefixes. In the parallel-prefix adders the sum is divided into three main steps: i) Pre-computation: the intermediate generate and propagate pairs (g, p) are composed by AND and exclusive-OR; ii) Prefix: all the signs calculated in the first step are processed by a prefix network, using a prefix cell operator; iii) Post-computation: using the intermediate signals propagate and the carry signals a sum result can be found by using another exclusive-OR gate. Fig. 3 shows the way of calculating the prefix.

An ideal prefix network will be composed of: i) log2(n) stage of logic, ii) fan-out never exceeding two at each stage, and iii) no more than one horizontal track of wire at each stage. Many parallel-prefix networks have been described in the literature to calculate the carry signals by taking into account the mentioned characteristics.

The PPA have similarities between them because all of them are divided into the same three steps. The difference between them is the area, the number of computations and the delay. The delay value is verified through logarithmic functions, and thus these adders are also called logarithmic delay adders. In the pre-processing stage, the carry signals required for the composition of the signals that enable the subsequent steps are calculated. In this part, propagate functions are used, where carry propagation values are generated. The carry propagation occurs when one of the two input signals corresponds to bit 1, as shown in (equation 1).

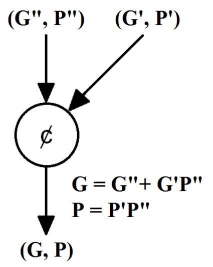

Fig. 3. Operator for the prefix calculation.

$$P_i = A_i \oplus B_i \tag{1}$$

In (1), A and B are the input signals composed of a logic gate XOR. In this way, if a Full-Adder circuit is considered, then Full-Adder cells are used to implement an RCA. An identical structure is presented to implement a PPA adder, where the sum values are exposed to capture the Carry value that will need to be generated for the next cell. Therefore, we can understand the propagation and generation carry functions. The carry generation function is given in (equation 2).

$$G_i = A_i \cdot B_i \tag{2}$$

In (equation 2), A and B are the input signals composed of a logic gate AND that only allows the carry generation when the two input values have a logical level 1. When the carry propagation and carry generation equations are analyzed, it is noticed that they generate input vectors according to the need of bits necessary for their application. But since these equations are performed in parallel, they do not add up a considerable computing delay, and the area cost reaches the bit size needed at the input.

In the stage of prefix computation, the carries are grouped according to the adders configuration regarding lower cost, power, and delay. According to the adders configuration, the prefix computation groups both values directly from the input with values that were computed in the pre-processing stage. The delay is increased by the configuration that has the highest critical path like the adders which process more than two inputs. The carry propagation function is given in (equation 3) and carry generation function is given in (equation 4), where P_i; P_{i+1}; G_i; P_{i+1} and G_{i+1} are the functions that were already acquired in the pre-process stage.

$$P_{i,i+1} = P_i \cdot P_{i+1} \tag{3}$$

$$G = (G_i \cdot P_{i+1}) + G_{i+1} \tag{4}$$

In the post-processing stage, the carry values that compose the output, are grouped. They are reached through the last adder configuration that is arranged by a solution that solves the problem of the correct carry propagation for each input bit. The final sum configuration is structured by an XOR function that captures the values coming from the final disposition of the carry. The function is shown in (equation 5), where P is the propagate function and C is the last carry signal that receives the values of the hierarchical structure of each adder.

$$S_{i+1} = P_{i+1} \oplus C_i \tag{5}$$

The decision to evaluate different adders in the sum tree of the square multipliers is because the choice of the most accurate precise adder can improve the power-performance relationship in such architectures.

In the parallel prefix adders, the carries are all calculated in parallel. The type of PPA adder depends on the way of organizing the tree of generating and propagating operators. Table I shows the parameters of the area and delay for the PPA adders mentioned above, according to the number of bits (n).

TABLE I. DELAY AND AREA OF n-BIT WIDTH PPAS

Adder	Delay	Area
Brent-Kung	$2\log_2(n) - 2$	$2n - 2 - \log_2(n)$
Kogge-Stone	$\log_2(n)$	$n\log_2(n) - (n-1)$
Han-Carlson	$\log_2(n) + 1$	$\frac{n}{2}\log_2(n)$
Ladner-Fischer	$2\log_2(n) - 1$	$2n - 2 - \log_2(n)$
Sklansky	$\log_2(n)$	$\frac{n}{2}\log_2(n)$

The sum tree of the Brent Kung adder [3] computes prefixes for 2-bit groups and is used to find prefixes for 4-bit groups, which are used to find prefixes for 8-bit groups and so on. forward, until we describe the sum tree with the amount of bits we want. We need to store the computation from the prefix computation, since these must be available to calculate the final carry. The fanout is limited to two cells per logical stage.

The Sklansky adder also called the divide adder to conquer [7] reduces the delay of the calculation of the intermediate prefixes, however this occurs at the cost of a fanout that doubles at each level. These high fanouts cause low performance of this adding cell.

The Kogge Stone advisor [4] is an optimal mix between efficiency and fanout, but the tree contains more spread propagation and carry cells, although this may not impact the area if the adder layout is in a regular grid, but this will increase energy consumption.

The Han Carlson adder [5] is a hybrid compound between Kogge Stone and Brent Kung adders. To get an optimal mix between power, power dissipation, fanout, and area, however in terms of acceleration it still loses to the Kogge Stone as we can analyze in [11].

The Ladner Fischer adder [6] has a regularity between the terms of Sklansky and Brent Kung adders, although its architecture is very similar to that proposed by [7], this adder calculates the prefix for odd numbers and uses one more stage to curl even positions.

IV. RESULTS

The multiplier architectures were described in VHDL HDL (Hardware Description Language) and synthesized using Cadence Genus™ Synthesis tool with varying frequency targets – according to the input bit-width – using the low-power ST 65nm commercial standard cell library with a 1.0V supply voltage. The synthesis for the 8-bit circuit versions had a operating frequency target of 200 MHz, whereas the 16-bit versions had a 100 MHz target.

As we can analyze in table II the power results follow the expected 8-bit quadratic multiplier that obtained better power dissipation with the use of the PW Skansky adder, this is due to the fact that for 8-bit results the PPAs are not at their full regularity and demonstrating their full functionality. In 16-bit multipliers, PPA adders already demonstrate better functionality and distribution of carry propagation and generation cells. Thus, as we analyzed in [11] the adder that obtained the best result in power dissipation was the PPA Kogge Stone adder, due to the regularity in the formation of its prefix computation.

In table III, we have the estimation of the area results where again we have the best quadratic multiplier area result in 8-bit module with the use of the Sklansky PPA adder. However, the reliability of the results is denoted by the integrity of the prefix computation of the adders that the larger the number of analyzed bits, the greater the layout of the carry generation and propagation cells. According to [4], the multiplier that obtained the best area was the one using the PPA Kogge Stone adder, still according to [4] although the Kogge Stone tree contains more propagation cells, and carry generation, this may not impact area results if the adder layout is in a regular grid, however this will increase power consumption. This statement can be reinforced by analyzing the results of [11], where for the Kogge Stone adder to gain in area we had an increase of 7.4% in energy consumption.

V. CONCLUSIONS

In the squared multiplier we obtained a reduction of 62.5% of the total hardware used, where when we analyzed a quadratic multiplier of 4 bits we were able to reduce the logic to six units of the partial products, that before the reduction showed results for sixteen partial products, which are acquired by multiplying coefficients. This reduction was obtained by analyzing a general purpose multiplier and the optimized quadratic multiplier structure.

The work still in progress, but which has already provided us with a relative improvement is the cubic multipliers where we were able to reduce by 50% the partial products and 67% the number of coefficients to be added, this architecture as well as the quadratic architecture to be used in the generators of harmonics need to be highly verified, so still new versions of squared and cubic multipliers will be discussed, as well as, the greater approach will be in the exploration of efficient adders in the sum tree of the multipliers, in order to obtain low-power results.

REFERENCES

[1] J. Lai, "Hardware Implementation of the Logarithm Function - using Improved Parabolic Synthesis," in *Department of Electrical and Information Technology, Faculty of Engineering, LTH, Master of Science Thesis*, 2013.

[2] A. Shaik, "Hardware Implementation of the Exponential Function Using Taylor Series and Linear Interpolation," in *Department of Electrical and Information Technology, Faculty of Engineering, LTH, Master of Science Thesis*, 2014.

[3] R. Brent and H. Kung, "A regular Layout for Parallel Adders," *IEEE Trans. Computer*, 1982.

TABLE II. Total power dissipation synthesis results.

Circuit Version	Power Dissipation (μW)			Total Power Variation (%)				
8-bit @ 200MHz	Leakage	Dynamic	Total	**Brent-Kung**	**Han-Carlson**	**Kogge-Stone**	**Ladner-Fischer**	**Sklansky**
Brent-Kung	0.5	318.9	319.5	-	4.2	3.8	3.7	4.3
Han-Carlson	0.5	306.2	306.7	-4.0	-	-0.3	-0.4	0.1
Kogge-Stone	0.5	307.1	307.6	-3.7	0.3	-	-0.1	0.5
Ladner-Fischer	0.5	307.5	308.0	-3.6	0.4	0.1	-	0.6
Sklansky	0.4	305.7	306.2	-4.2	-0.1	-0.5	-0.6	-
Circuit Version	Power Dissipation (μW)			Total Power Variation (%)				
16-bit @ 100MHz	Leakage	Dynamic	Total	**Brent-Kung**	**Han-Carlson**	**Kogge-Stone**	**Ladner-Fischer**	**Sklansky**
Brent-Kung	1.6	270.2	271.8	-	-0.4	3.2	1.5	1.8
Han-Carlson	1.6	271.3	272.9	0.4	-	3.6	1.9	2.2
Kogge-Stone	1.5	261.8	263.4	-3.1	-3.5	-	-1.6	-1.3
Ladner-Fischer	1.6	266.1	267.8	-1.5	-1.9	1.7	-	0.3
Sklansky	1.6	265.3	266.9	-1.8	-2.2	1.3	-0.3	-

TABLE III. Circuit area, critical path delay, and maximum frequency synthesis results.

Circuit Version	Gate Count	Cell Area	Area Variation (%)					C-Path Delay	Max. Frequency
8-bit @ 200MHz	(kgates)	(μm^2)	**Brent-Kung**	**Han-Carlson**	**Kogge-Stone**	**Ladner-Fischer**	**Sklansky**	(ps)	(MHz)
Brent-Kung	160	723.3	-	10.5	10.4	9.7	11.7	4999.2	200.0
Han-Carlson	132	654.7	-9.5	-	-0.1	-0.7	1.1	4999.0	200.0
Kogge-Stone	131	655.2	-9.4	0.1	-	-0.6	1.2	4993.8	200.2
Ladner-Fischer	134	659.4	-8.8	0.7	0.6	-	1.8	4992.0	200.3
Sklansky	129	647.4	-10.5	-1.1	-1.2	-1.8	-	4982.4	200.7
Circuit Version	Gate Count	Cell Area	Area Variation (%)					C-Path Delay	Max. Frequency
16-bit @ 100MHz	(kgates)	(μm^2)	**Brent-Kung**	**Han-Carlson**	**Kogge-Stone**	**Ladner-Fischer**	**Sklansky**	(ps)	(MHz)
Brent-Kung	509	2301.5	-	-1.8	1.2	0.1	0.7	9995.3	100.0
Han-Carlson	519	2344.1	1.9	-	3.1	2.0	2.5	9999.9	100.0
Kogge-Stone	474	2273.4	-1.2	-3.0	-	-1.1	-0.5	9998.3	100.0
Ladner-Fischer	502	2298.9	-0.1	-1.9	1.1	-	0.6	9996.7	100.0
Sklansky	501	2285.9	-0.7	-2.5	0.5	-0.6	-	9999.8	100.0

[4] P. M. Kogge and H. S. Stone, "Parallel Algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, 1973.

[5] M. Han, "Approximate computing: An emerging paradigma for energy-efficient design," *18th IEEE European Test Symposium (ETS)*, 2013.

[6] R. Ladner and M. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, 1980.

[7] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic Computers*, 1960.

[8] A. B. Sassi and J. N. S. Junior, "Projeto de uma ULA de inteiros e de baixo consumo em tecnologia CMOS," *Dissertao apresentada escola de Engenharia de So Carlos da Universidade de So Paulo como parte dos requisitos para obteno do ttulo de mestre em ciłncias, Programa de Engenharia Eltrica*, 2013.

[9] N. Weste and D. M. Harris, "CMOS VLSI Design a circuits and system perspective," *fourth edition, Pearson*, 2011.

[10] A. Beaumont-Smith and C.-C. Lim, "Parallel prefix adder design," in *IEEE Symposium on Computer Arithmetic*, 2001, pp. 218–225.

[11] M. da Rosa, B. Silveira, L. Soares, C. Diniz, and E. Costa, "Exploriting the Use of Parallel Prefix Adder Topologies into Aproximate Adder Circuits," *24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2017.