COMPARISON AMONG BOOTH'S AND PEKMESTZI'S ALGORITHM FOR THE MULTIPLICATION OF TWO NUMBERS

R. Rojas-Laguna, I. E. Villalón Turrubiates[*], *S. Serrano-Arellano*[*], *E. Alvarado-Méndez, J. M. Estudillo-Ayala and O. Vite-Chávez*[+]

Universidad de Guanajuato.
Facultad de Ingeniería Mecánica, Eléctrica y Electrónica.
Maestría en Ingeniería Eléctrica, Opción: Instrumentación y Sistemas Digitales.
Prolongación Tampico No. 912, Tel.: 01 464 6480911 ext. 119, Col. Bellavista.
36730 Salamanca, Gto., México.

ABSTRACT

A comparison between two different methods of multiplication of two 8-bit numbers is presented. This methods are the Booth's algorithm and the algorithm proposed by Kiamal Z. Pekmestzi [1]. The general objective is to show the benefits and the advantages obtained if it's used one of this algorithms over the other. This multipliers have low circuit complexity permitting high-speed operations and the interconnections of the cells are regular. This is the reason why the results shown was obtained using VHDL realization on a FPGA XC4010XL by Xilinx.

1. INTRODUCTION

Multiplication is the most critical operation in every computational system. Innumerable schemes have been proposed for the realization of this operation. In the early multiplier schemes proposed by Hoffman [2], Burton and Noaks [3], De Mori [4] and Guilt [5] for positive numbers, and by Baugh and Wooley [6] and Hwang [7] for numbers in two's complement form, the effort was on implementations using iterative circuits with uniform interconnections pattern. Also, on the same base, schemes using Booth's algorithm [8], [9] were presented.

The Booth's algorithm serves two purposes:

1. Fast multiplication (when there are consecutive 0's or 1's in the multiplier).
2. Signed multiplication.

The Pekmestzi's multiplication algorithm is based on a different mechanism. At each step, one bit of the multiplier and one bit of the multiplicand are processed. So, the algorithm is symmetric; this means that multiplier and multiplicand can be interchanged.

The main objective of this investigation is to find which of those algorithms provides the best results under a VHDL implementation in a FPGA XC4010XL of Xilinx©, and with this, ensure the best option at the moment of its use on a certain application.

2. BOOTH'S ALGORITHM

The goal of the algorithm is to recode the multiplier to reduce the number of additions done. The key idea is that when we have a string of 1's in the multiplier, representing a string of multiplicand adds, we can get the same effect with a single add and subtract. This is because:

$$2^m + 2^{m-1} + \ldots + 2^{n+1} + 2^n = 2^{m+1} - 2^n$$

For example:

$$01110 = 10000 - 00010$$
$$14 = 16 - 2$$

We can implement the multiplier by looking at the least significant 2 bits of Q. If they are:

0	**0**	we shift only
0	**1**	we add M and shift
1	**0**	we subtract M and shift
1	**1**	we shift only

Let's see how this is done with an example: 9 * 14 (for n = 5):

M	A	Q	Qprev	
01001	00000	0111**0**	**0**	Load
01001	00000	0011**1**	**0**	Shift only
01001	11011	1001**1**	**1**	Subtract and shift
01001	11101	1100**1**	**1**	Shift only
01001	11110	1110**0**	**1**	Shift only
01001	00011	1111**0**	**0**	Add and shift

Booth's algorithm also works with negative numbers like 2 * -3:

M	A	Q	Qprev	
00010	00000	1110**1**	**0**	Load
00010	11111	0111**0**	**1**	Subtract and shift
00010	00000	1011**1**	**0**	Add and shift
00010	11111	0101**1**	**1**	Subtract and shift
00010	11111	1010**1**	**1**	Shift Only
00010	11111	1101**0**	**1**	Shift only

[*] CONACyT Scholarship
[+] CONCyTEG Scholarship

3. PEKMESTZI'S ALGORITHM

Consider two positive integer numbers X and Y:

$$X = x_{n-1}x_{n-2}....x_0 = \sum_{j=0}^{n-1} x_j 2^j \quad (1)$$

$$Y = y_{n-1}y_{n-2}....y_0 = \sum_{j=0}^{n-1} y_j 2^j \quad (2)$$

We define X_{n-1} and Y_{n-1} as the numbers that remain after truncation of bits x_{n-1} and y_{n-1}. So:

$$X_{n-1} = x_{n-2}x_{n-3}....x_0 = \sum_{j=0}^{n-2} x_j 2^j \quad \text{and} \quad X = X_{n-1} + 2^{n-1}x_{n-1} \quad (3)$$

$$Y_{n-1} = y_{n-2}y_{n-3}....y_0 = \sum_{j=0}^{n-2} y_j 2^j \quad \text{and} \quad Y = Y_{n-1} + 2^{n-1}y_{n-1} \quad (4)$$

The product of the numbers X and Y, according to (3) and (4), can be computed as follows:

$$P = 2^{2n-2}x_{n-1}y_{n-1} + 2^{n-1}\{x_{n-1}Y_{n-1} + y_{n-1}X_{n-1}\} + X_{n-1}Y_{n-1} \quad (5)$$

Let us define $P_{n-1}=X_{n-1}Y_{n-1}$ and generally $P_j=X_jY_j$ (6), where X_j and Y_j are the numbers formed by the j least significant bits of X and Y, respectively. The product P_j can be computed by the following recursive equation:

$$P_j = X_jY_j = 2^{2j-2}x_{j-1}y_{j-1} + 2^{j-1}\{x_{j-1}Y_{j-1} + y_{j-1}X_{j-1}\} + P_{j-1} \quad (7)$$

According to the above relations, the product P=XY can be computed by the equation:

$$P = \sum_{j=0}^{n-1} x_j y_j 2^{2j} + \sum_{j=1}^{n-1}\{x_jY_j + y_jX_j\}2^j \quad (8)$$

The main computation concerns the addition of the terms:

$$Z_j = x_jY_j + y_jX_j \quad (9)$$

These terms must be added, properly weighted, and the product is given by the next relation:

$$P = \sum_{j=0}^{n-1} x_j y_j 2^{2j} + \sum_{j=1}^{n-1} Z_j 2^j \quad (10)$$

The values that quantity Z_j can take depend on the values of the logical variables x_j and y_j and are shown in table 1. The only case where Z_j requires computation is when the two bits of the multiplied numbers have value 1. At each step j, only s_j and c_{j+1} are new. The rest of the bits of S_j have been formed in the previous j-1 steps according to the relation:

$$S_j = S_{j-1} + x_{j-1} + y_{j-1} \quad (11)$$

The sum S=X+Y can be computed once. During the *j*th iteration of the algorithm, the j least significant bits of S with c_j as the most significant bit form the quantity S_j which, in turn, is used only if $x_j=y_j=1$. In this case, however, $s_{j+1}=c_j$ and, in the proposed algorithm, the quantity S_j can be computed equivalently by both of the following relations:

$$S_j = c_j s_{j-1}s_{j-2}....s_0 \quad \text{or} \quad S_j = s_j s_{j-1}s_{j-2}....s_0 \quad (12)$$

TABLE 1
VALUES OF Z_j TO DETERMINE THE PRODUCT P=XY

X_J	Y_J	Z_J
0	0	0
0	1	X_j
1	0	Y_j
1	1	$X_j+Y_j=S_j$

The proposed multiplexer-based parallel multiplier is shown in Fig. 1. The cells are described in Fig. 2, 3 and 4. The implementation of a two's complement multiplier based on the proposed technique can be done on an array similar to the array of Fig. 1. Let us consider two integer numbers X and Y in this form:

$$X = x_{n-1}x_{n-2}....x_0 = -2^{n-1}x_{n-1} + \sum_{j=0}^{n-2} x_j 2^j = -2^{n-1}x_{n-1} + X_{n-1} \quad (13)$$

$$Y = y_{n-1}y_{n-2}....y_0 = -2^{n-1}y_{n-1} + \sum_{j=0}^{n-2} y_j 2^j = -2^{n-1}y_{n-1} + Y_{n-1} \quad (14)$$

X_{n-1} and Y_{n-1} are the numbers that remain after truncation of the bits x_{n-1} and y_{n-1} from X and Y, respectively. The product P of the numbers X and Y, according to (13) and (14), is equal to:

$$P = XY = 2^{2n-2}x_{n-1}y_{n-1} - 2^{n-1}\{x_{n-1}Y_{n-1} + y_{n-1}X_{n-1}\} + X_{n-1}Y_{n-1} \quad (15)$$

The above relation differs from the corresponding (5) for positive numbers only in the sign of the term $Z_{n-1} = x_{n-1}Y_{n-1} + y_{n-1}X_{n-1}$. Now, the above term must be subtracted instead of added. The algorithm for all other terms remains unchanged and the product P=XY can be computed by the next equation:

$$P = \sum_{j=0}^{n-1} x_j y_j 2^{2j} + \sum_{j=1}^{n-2} Z_j 2^j - Z_{n-1}2^{n-1} \quad (16)$$

Quantity Z_{n-1} is generated of the left boundary cells and it is subtracted by inverting the S_{out} bits of these cells. Also, the two additive inputs of the next to the leftmost top cells must be set to one. The final array implementing the multiplication of numbers in two's complement form is shown in Fig. 5.

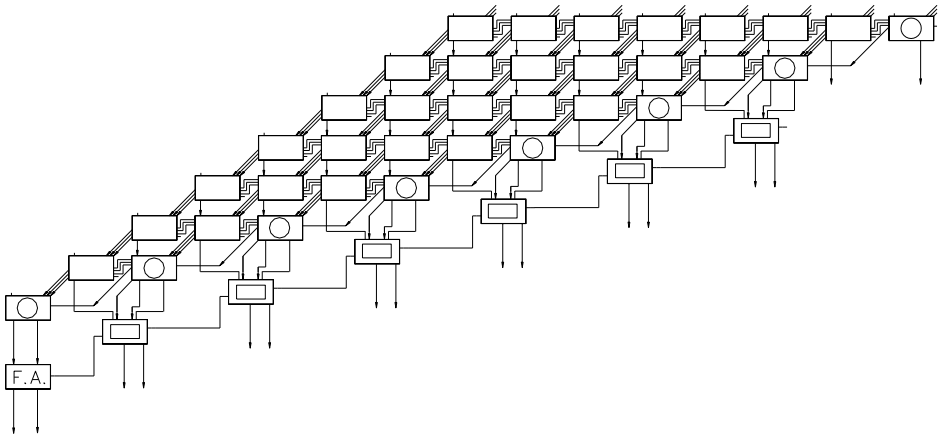

Fig. 1. General diagram used by the proposed algorithm to calculate the product between two numbers

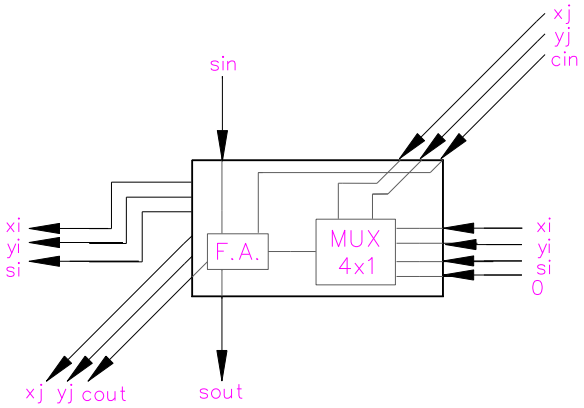

Fig. 2. Diagram of the rectangular cells (not filled from Fig. 1)

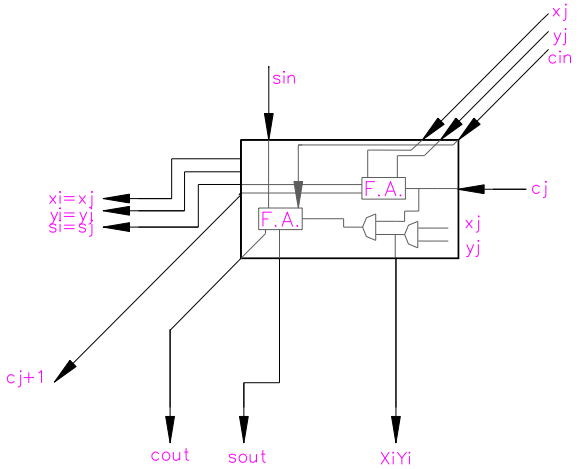

Fig. 3. Diagram of the rectangular cells with a circle inside (see Fig. 1)

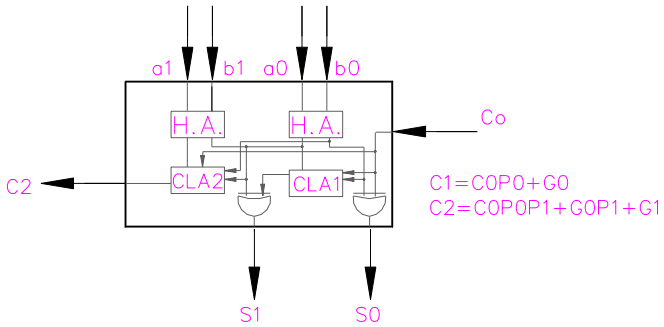

Fig. 4. Diagram of the rectangular cells with a rectangle inside (see Fig. 1)

4. DESCRIPTION IN VHDL CODE

The Booth's algorithm proposed in the section 2, as well as the Pekmestzi's algorithm for positive numbers and in complement at two proposed in the section 3, are now coded in the descriptive language VHDL. These three algorithms are coded in combinational logic, whose main characteristics are:

1. The exit functions depends only on the state of the entrances.
2. It doesn't contain memory elements.
3. It has two level realization of logical gates.

5. SYNTHESIS AND SIMULATION

Once compiled the three codes, we proceeds to carry out the simulation of the same ones. For it, the software Foundation by Xilinx© is used, carrying out the synthesis in a FPGA XC4010XL also by Xilinx©.

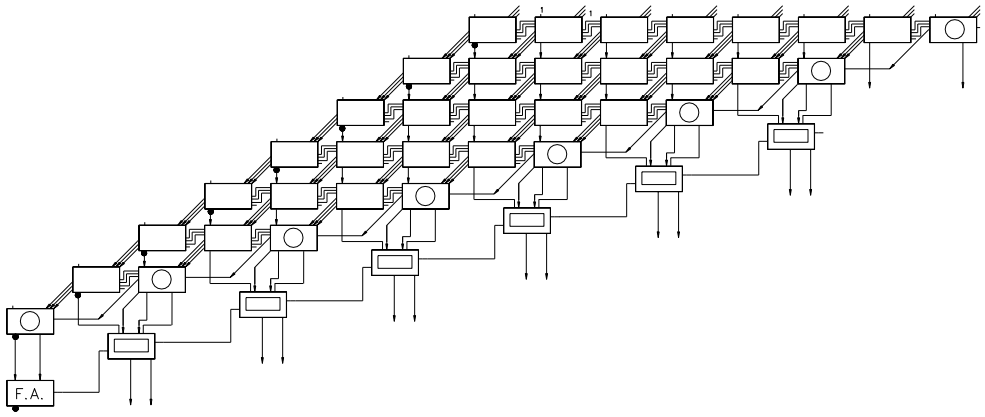

Fig. 5. General diagram of the proposed multiplexer-based two's complement parallel multiplier

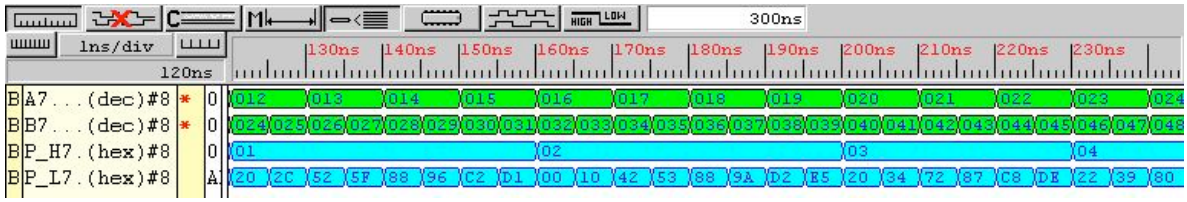

Fig. 6. Positive numbers multiplication using Booth's Algorithm

First, we are going to simulate only positive numbers multiplications. Taking A and B as our inputs numbers (8 bits). The Fig. 6 shown the results using Booth's algorithm. We can see that the results are shown in hexadecimal numbers, and is divided in two parts (low and high). The final number is the concatenation of these numbers. The Fig. 7 shown the results using Pekmestzi's algorithm. It's clear that the results are shown directly by the output P, in decimal numbers. Now, to show the two's complement multiplication, let's use an example. Let us consider $A=-19_{10}$ and $B=+22_{10}$. Now, the number A in two's complement is $A=11101101_2$, and $B=00010110_2$. The multiplication in decimal of these numbers is $P=-418_{10}$, or $P=1111111001011110_2$. Changing to hexadecimal, the product is $P=FE5E_{16}$. The Fig. 8 shown the simulation using Booth's algorithm, where the result is shown divided in two (high and low) again, but the concatenation of these values is the expected result. The Fig. 9 shown the simulation using Pekmestzi's algorithm. The result is direct and is equal to the expected value.

6. COMPARISON AMONG THE TWO ALGORITHMS

Now, we proceed to compare the obtained results of the simulations of both algorithms. Of this comparison, we obtain the following remarks:

1. Both algorithms carry out the multiplication of positive numbers and numbers in two's complement.

2. Both algorithms can be coded in the descriptor language VHDL in combinational logic.

3. The operation speed is appropriate.

However, certain advantages and disadvantages exist when using one of these codes instead of the other one:

1. For the Pekmestzi's algorithm, it is necessary a code for positive numbers multiplications and other similar but with certain differences for the multiplication with numbers in two's complement, what takes us to have two extensive codes. For the Booth's algorithm, we have a single and smallest code that carries out both operations.

2. The number of logical gates on the FPGA XC4010XL is 10,000. The Pekmestzi's code for positive numbers (only) uses a total of 1195 equivalent gates for design (11.95%). The Pekmestzi's algorithm for numbers in two's complement uses a total of 1236 equivalent gates for design (12.36%). The Booth's algorithm uses a total of 1287 equivalent gates for design (12.87%).

3. The VHDL code for the Pekmestzi's algorithm is of around 300 lines, while for the Booth's algorithm decreases to only 100.

4. The Pekmestzi's algorithm gives us an alone number as a result. However, the Booth's algorithm gives us two numbers that should be concatenated to obtain the result.

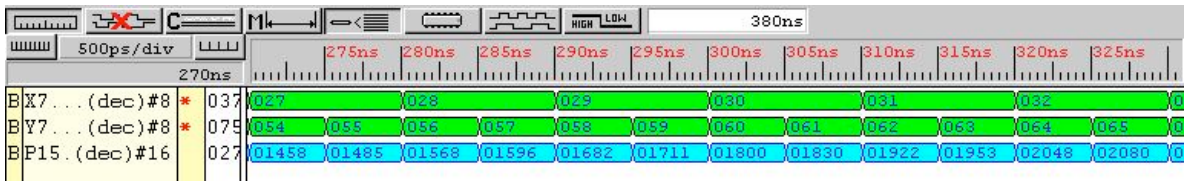

Fig. 7. Positive numbers multiplication using Pekmestzi's Algorithm

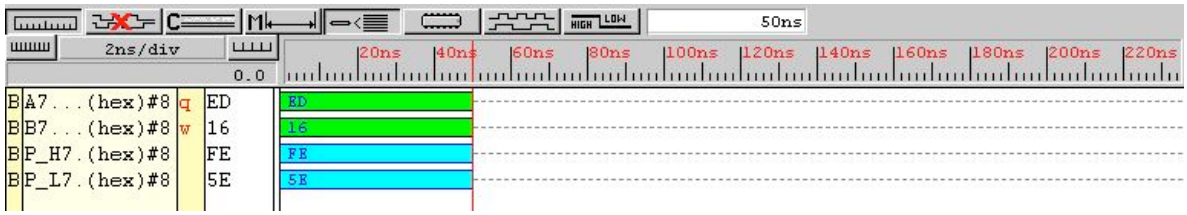

Fig. 8. Two's complement multiplier using Booth's Algorithm

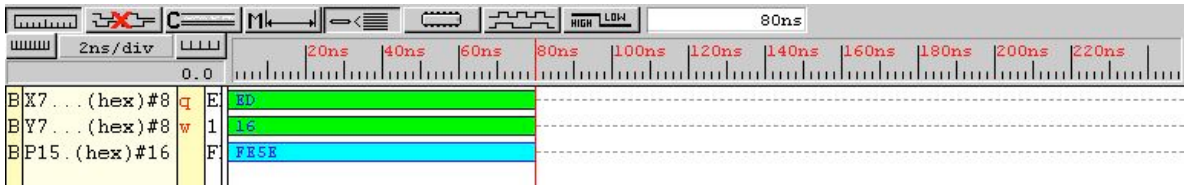

Fig. 9. Two's complement multiplier using Pekmestzi's Algorithm

7. CONCLUSION

The multiplication is one of the most critical operations in every computational systems. In this work two different algorithms are shown for the calculation of multiplications between positive numbers and two's complement numbers, which are the Booth's algorithm and the Pekmestzi's algorithm.

Once we carries out the comparison between both methods, can conclude that these algorithms complete their multiplicative work indeed. However, some advantages exist of using one of them instead of the other one, like it was described in section 6.

If is needed to use this multiplier in a complex circuit, it is convenient to use the Booth's algorithm, because the code is less extensive and it carries out both operations. However, if only multiplications are going to be carried out, the Pekmestzi's algorithm is effective because in spite of being extensive in code is simple of to understand and to use.

Something very important is the number of gates used for design. It is possible to see that the Pekmestzi's algorithm for positive numbers (only) is the best choice, because it uses less quantity of gates. However, the algorithm has limitations (no negative numbers). In the end, the selection of the best option will depend on the application in which will be used.

As a personal opinion, the Booth's algorithm can provide good results on a small code and using a very similar number of gates to those that are used by the Pekmestzi's algorithms. So, Booth's algorithm is a very good choice for application problems.

It is necessary to point out that the Foundation software by Xilinx© did not allow to carry out a single code that carried out both operations in the case of the Pekmestzi's algorithm (due to the extensive code). Also, it didn't allow to carry out the direct concatenation of the exit in the case of the Booth's algorithm.

8. ACKNOWLEDGEMENTS

This work was supported by CONACyT under projects J35313-A, J32018-A, J35303-E and the project "Apoyo a Actividades Académicas de los Progrmas de Postgrado de Excelencia de CONACyT".

This project was also supported by Xilinx©, providing the software and hardware (FPGA XC4010XL) for the realization, under their "University Program" for support.

9. REFERENCES

[1] K. Z. Pekmestzi, "Multiplexer-Based Array Multipliers", *IEEE Trans. Computers,* vol. 48, no. 1, pp. 15-23, Jan. 1999.

[2] J. Hoffman, G. Lacaze, P. Csillag, "Iterative Logical Network for Parallel Multiplication", *Electronics Letters,* vol. 4, p. 178, 1986.

[3] P. Burton, D. R. Noaks, "High-Speed Iterative Multiplier", *Electronics Letters,* vol. 4, p. 262, 1968.

[4] R. De Mori, "Suggestion for an IC Fast Parallel Multiplier", *Electronics Letters,* vol. 5, pp. 50-51, Feb. 1969.

[5] H. Guilt, "Fully Iterative Fast Array for Binary Multiplication", *Electronics Letters,* vol. 5, p. 269, 1969.

[6] R. Baugh, B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm", *IEEE Trans. Computers,* vol. 22, no. 12, pp. 1045-1059, Dec. 1973.

[7] K. Hwang, "Global and Modular Two's Complement Array Multipliers", *IEEE Trans. Computers,* vol. 28, no. 4, pp. 300-306, Apr. 1979.

[8] A. Booth, "A Signed Binary Multiplication Technique", *Quarterly J. Mechanics of Applied Math,* vol. 4, pp. 236-240, 1951.

[9] L. MacSorley, "High Speed Arithmetic in Binary Computers", *Proc. IRE,* vol. 49, Jan. 1961.

[10] http://ivs.cs.uni-magdeburg.de/EuK/Lehre/booth.html, "Booth's Algorithm".