

The Universal Multiplier Unit

Humberto Calderón and Stamatis Vassiliadis
 Computer Engineering Laboratory
 Faculty of Electrical Engineering, Mathematics and Computer Science
 Delft University of Technology
 Mekelweg 4, 2600 GA Delft, The Netherlands
 Phone: +31 (15)2783664 Fax: +31 (15)2784898
 E-mail: {H.Calderon|S.Vassiliadis}@ewi.tudelft.nl

Abstract— This paper presents a thoroughly detailed design of a new Universal Multiplier Unit capable of working with unsigned, signed and two's complement numbers. A review of different multiplier structures such as the unsigned array multiplier, Baugh-Wooley and Wallace-Booth Tree was also performed. A comparison of all these multipliers is achieved, and their performances were found out through the mapping of the algorithms into a Spartan IIE FPGA.

Finally, the Synthesis results showed that the overhead in silicon space and time delay necessary to produce a Universal Multiplier Unit are despicable compared with traditional multipliers, considering its functionality.

Keywords— Unsigned numbers, signed numbers, two's complements, basic multipliers, CSA, Array multiplier, Baugh-Wooley, Wallace-Booth, FPGA implementation of Multipliers.

I. INTRODUCTION

Number representation in computer machines is achieved with binary digits (bits) of finite length. The encoding approximates real, rational and integer numbers with the sequence of bits; these numbers are operated by the machine as if they were integers and their description is constructed in a weighted positional representation [1]. The three widely used representations for numbers in binary computers are: unsigned, signed and two's complement numbers.

A. Unsigned Number

Let r be the radix of a weighted positional system with $r \geq 2$ and the number N be represented in the digit set $0 \leq N \leq r-1$; then a positive number can be represented by (1)

$$N_{n-1}N_{n-2}\dots N_1N_0 \quad (1)$$

where N will be equivalent in decimal representation to:

$$N = N_{n-1}r^{n-1} + N_{n-2}r^{n-2} + \dots + N_0 = \sum_{i=0}^{n-1} N_i r^i \quad (2)$$

Negative numbers can be expressed by:

$$N_- = -N_{n-1}r^{n-1} - N_{n-2}r^{n-2} - \dots - N_0 = \sum_{i=0}^{n-1} -N_i r^i \quad (3)$$

Nevertheless, the representation for negative numbers presented in (3) is not convenient for machine coding purposes. Therefore, negative number representations have been encoded in several ways [2] like the sign plus magnitude and two's complement.

B. Sign Magnitude Number

Sign magnitude number uses the most significant bit (MSB) to encode the sign, which corresponds to the $n-1$ position of the string of bits, and the lower $n-2$ bits are used to represent the magnitude. Equation (4) considers a sign magnitude notation for a negative number “ $-N$ ” denoted by N_- . For radix 2 a negative number is represented with a one in the most significant bit position :

$$N_- = [(r-1), N_{n-2}, N_{n-1}, \dots, N_1, N_0] \quad (4)$$

On the other hand, the sign of positive numbers is represented with a zero in the MSB as depicted in 5:

$$N = [0, N_{n-2}, N_{n-1}, \dots, N_1, N_0] \quad (5)$$

From equations 4 and 5 it is clear that a positive version of a number N , represented in sign magnitude notation, differs from the negative version only by the sign bit and the magnitude of both can be represented by the absolute value of N :

$$|N| = \sum_{i=0}^{n-2} N_i r^i \quad (6)$$

C. Two's Complement Number

A complement representation is another way to encode signed numbers. In complement representation of radix 2 numbers, the encoding of a positive number uses a zero in the MSB, representing the sign, and the lower $n-2$ bits represent the magnitude. A negative number in this representation is obtained by negating the unsigned representation of the number and adding one as expressed in equation (7):

$$N_- = [(r-1), \overline{N}_{n-2}, \overline{N}_{n-1}, \dots, \overline{N}_1, \overline{N}_0 + 1] \quad (7)$$

Two's complement, sign magnitude and unsigned representations are the essential codifications used to develop new algorithms and hardware organizations by the Computer Arithmetic.

D. Computer Arithmetic

The Computer Arithmetic looks for the improvement of the logic design and the development of new and more efficient algorithm to improve the performance in terms of time, area,

and power consumption of arithmetic operations, working with unsigned, signed magnitude and two's complement notations. The basic circuits for processing arithmetic operations are built from combinational elements that are interconnected by wires (e.g. switching transistors in FPGAs) [3]. Half Adder (HA) and Full Adder (FA) are two basic combinational circuits which are the fundamental base for processing arithmetic operations. With the above central core cells it is possible to construct a complete n bit adder and create different structures for processing the binary addition. The most utilized structure is the Ripple Carry Addition (RCA), which propagates a carry chain of $n-1$ for n bits of summands and it has a processing time of $\Theta(n)$. Another structure, called Carry-Lookahead Adder (CLA), improves the carry propagation problem through the use of a lookahead carry generator, which has the capacity of generating the carries in a parallel way, diminishing the bounding time to $\Theta(\log n)$. Finally, a structure of basic adders, called Carry Save Addition (CSA), is used to process multioperand addition with a deferred carry assimilation obtaining bounding time of $O(\log n)$. The addition is the base for the implementation of multiplication operations, which are used to process multimedia data like graphics, audio, image and video.

The discussion of this paper is organized as follows. Section II gives some background information about the most important structures and algorithms in multiplication. Section III presents the design of a Universal Multiplication Unit (UMU). The results and discussion of the FPGA implementation of the Universal Multiplication Unit and some important unsigned and signed multiplication structures are presented in section IV. Finally, the article is concluded in section V.

II. THE MULTIPLICATION

Let the multiplier and the multiplicand be denoted by X and Y :

$$X = X_{n-1}X_{n-2}\dots X_1X_0 \quad (8)$$

$$Y = Y_{n-1}Y_{n-2}\dots Y_1Y_0 \quad (9)$$

When applying the fundamental multiplication algorithm we will have to operate within sequential characteristics and the processing steps will be $n-1$ for a multiplier of n bits. The following equation (10) recursively expresses the operation:

$$P_{j+1} = (P_j + X_j Y) 2^{-1} \quad (10)$$

where partial product P_j is obtained from the product of the multiplier bit X_j and Y with $P_{(0)} = 0$. This sequential algorithm is prohibitive for some applications due to its inherently slow processing characteristic, and it cannot be used for applications like Computer Graphics, signal and image processing [4]. Many algorithms and structures have been created for multiplication processing in a non sequential way, and looking for the speeding up of the inherently multiplication delay were created important combinational structures like the array multiplier [5], which provides a regular structure using shortest wires when implemented in a

VLSI layout. Usually the array multiplier has three main stages: 1) Partial products generation, 2) Partial products addition with Carry Save Adders (CSA) and 3) Final adder with some carry propagation technique.

A. Carry save adder based multiplier

Carry-Save Adder has been created to add three or more operands, avoiding in this way the carry propagation for every operand added and optimizing the processing in terms of time. With this structure the addition of three n -bit numbers takes a constant additional amount of time. A CSA cell is presented in figure 1 and it corresponds to a Full Adder Cell.

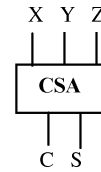


Figure 1 Carry Save Adder Cell

CSA cell accepts three operands and produces two outcomes; sometimes this is called a (3,2) counter [1][2]. Multi-operand addition implicit in multiplication operations is achieved with a CSA array or tree; figure 2 presents an implementation of a CSA array for the multiplication of two 5 bits unsigned numbers.

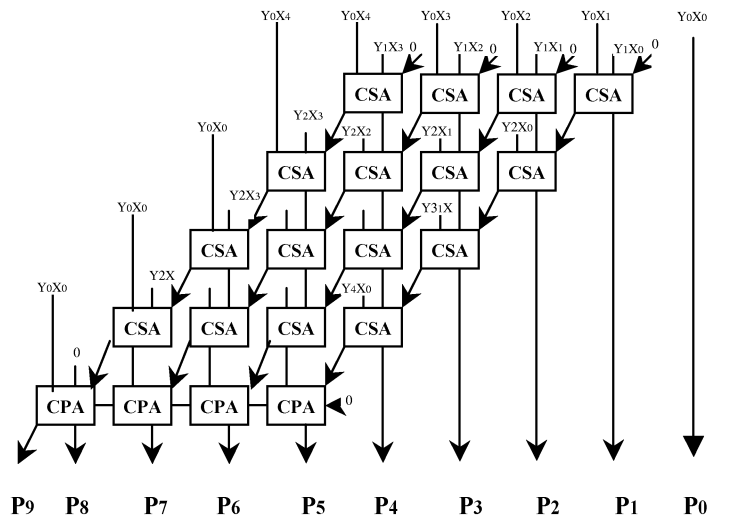


Figure 2 CSA Tree for Unsigned multiplication.

Beyond this basic structure, different algorithms and organizations have been developed for the processing of unsigned, signed magnitude and two's complement numbers. Pezaris [6] proposed an array structure for processing signed numbers, an optimized processing of signed numbers is achieved with the Booth recoding [7].

B. Booth recoding

It has been observed that it is not necessary to compute all the bits of a multiplier, diminishing with this approach the number of partial products to be processed. Therefore, a string of consecutive 1s in multiplier X can be replaced by the

addition of the next position of the last one in the string and the subtraction of the least significant one in the scanned string. Equation 11 depicts this observation.

$$2^j + 2^{j-1} + 2^{j-2} + \dots + 2^{i+1} + 2^i = 2^{j+1} - 2^i \quad (11)$$

Booth proposed this kind of recoding five decades ago and describes a simple algorithm for the multiplication of two signed numbers without the necessary logic correction of other algorithms [1] [2]. The algorithm walks along the bits of the multiplier, creating a new partial product using the previous X_{i-1} bit scanned and the actual one X_i ; as is depicted in Table 1. The simplifications introduced by this encoding into the multiplier will help to diminish the processing time of the multiplication.

Table 1 Booth encoding

X_i	X_{i+1}	Comments
0	0	Shift the sum of partial products one bit to the right
0	1	Shift the sum of partial products one bit to the right
1	0	Add the multiplicand Y to the sum of partial products and then shift the result one bit to the right.
1	1	Subtract the multiplicand Y from the existing sum of partial products and then shift the result one bit to the right

However, this algorithm presents drawbacks due to its inefficient processing in presence of isolated 1's, and the variable number of addition-subtraction and shift operations required [1].

C. MacSorley encoding

An improvement of the Booth algorithm was presented by MacSorley [8]. The modification takes into account three bits (radix 4) instead of two, accelerating in this way the processing time. The recoding used by MacSorley proposal is presented in table 2 and it is also expressed in equations 12 to 15.

Table 2 MacSorley encoding

X_{i+2}	X_{i+1}	X_i	Add to Partial Product
0	0	0	+0Y
0	0	1	+1Y
0	1	0	+1Y
0	1	1	+2Y
1	0	0	-2Y
1	0	1	-1Y
1	1	0	-1Y
1	1	1	-0Y

$$X = -X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i \quad (12)$$

$$= -\sum_{i=0}^{n-1} X_i 2^i + 2 \sum_{i=0}^{n-2} X_i 2^i \quad (13)$$

$$= \sum_{i=0}^{n-1} (-X_i + X_{i-1}) 2^i \quad (14)$$

Then, for an n even number the equation can be rearranged as follows:

$$= \sum_{i=0}^{n/2-1} (-2 X_{2i+1} + X_{2i} + X_{2i-1}) 2^{2i} \quad (15)$$

The following three steps can describe the MacSorley algorithm:

1. Extend the sign position when it is necessary to have an even word.
2. Append a 0 to the right of the LSB in the multiplier.
3. According to the scanned bits, the partial product will be 0, +Y, -Y, +2Y or -2Y.

D. Wallace Tree

Wallace [9] introduced another structure for multiplication: the traditional array was changed to a tree structure; this new organization looks for the implementation of a faster addition with minimum delay in the propagation. This CSA tree reduces n-bit operands to 2 bit operands and when used in multiplication improves the column of adding partial products, combining at the end, all partial products into two vectors: the carry and the sum. This outcome of the tree is then combined using a conventional adder [10]. An example of this structure used in the addition of nine numbers is shown in figure 3.

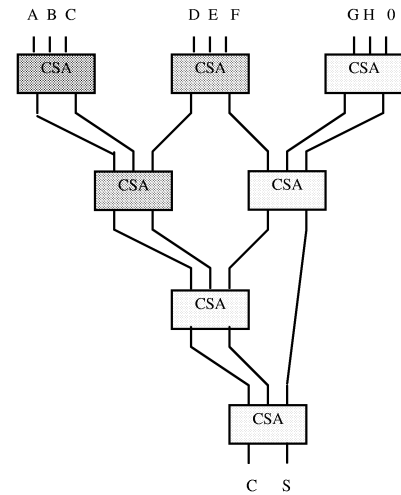


Figure 3 Wallace Tree

E. Wallace-Booth Multiplier

When a Wallace tree is used to add the partial products generated by the Booth or Modified Booth encoders, the resultant structure is called Wallace-Booth Multiplier [11]. A block diagram of this low delay organization is presented in figure 4.

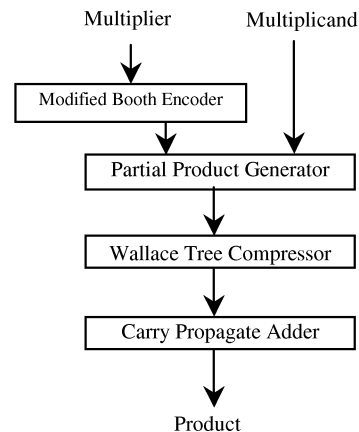


Figure 4 Block Diagram of Wallace-Booth Multiplication

III. THE NEW UNIVERSAL MULTIPLICATION UNIT

The main difference between the traditional multiplication units and the proposed configurable Universal Multiplication Unit, is the capacity of processing unsigned, signed magnitude and two's complement numbers with a central core array, optimizing in this way the used resources.

Assume the multiplication of two numbers, X and Y, encoded in two's complement representation for a binary number of length n. Equations 16 and 17 depict the extended representations of a regular binary number in one bit.

$$X = -X_n 2^n + \sum_{i=0}^{n-1} X_i 2^i \quad (16)$$

$$Y = -Y_n 2^n + \sum_{j=0}^{n-1} Y_j 2^j \quad (17)$$

The product X*Y will produce four Main Multiplication Products (MMP) depicted by the equations 18-21.

$$MMP_1 = X_n Y_n 2^{2n} \quad (18)$$

$$MMP_2 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j 2^{i+j} \quad (19)$$

$$MMP_3 = 2^n \left(-2^n + \sum_{i=0}^{n-1} X_i Y_n 2^i \right) \quad (20)$$

$$MMP_4 = 2^n \left(-2^n + \sum_{j=0}^{n-1} Y_j X_n 2^j \right) \quad (21)$$

Using the well known two's complement property expressed in equation 22:

$$-\sum_{u=0}^{n-1} X_u 2^u = -2^n + \sum_{u=0}^{n-1} \overline{X_u} 2^u + 1 \quad (22)$$

Equations 20 and 21 can be rewritten to yield:

$$MMP_{3M} = -2^{2n} + \sum_{i=0}^{n-1} \overline{X_i} Y_n 2^{i+n} + 2^n \quad (23)$$

$$MMP_{4M} = -2^{2n} + \sum_{j=0}^{n-1} \overline{Y_j} X_n 2^{j+n} + 2^n \quad (24)$$

Finally, grouping the four multiplication products, equation 25 is obtained, which is the representation of the P = X*Y product.

$$P = X_n Y_n 2^{2n} + \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j 2^{i+j} + \sum_{i=0}^{n-1} \overline{X_i} Y_n 2^{i+n} + \sum_{j=0}^{n-1} \overline{Y_j} X_n 2^{j+n} - 2^{n+1} + 2^{2n+1} \quad (25)$$

Based on this result a Universal Multiplier Unit is created. An example for n = 4 is showed in figure 5.

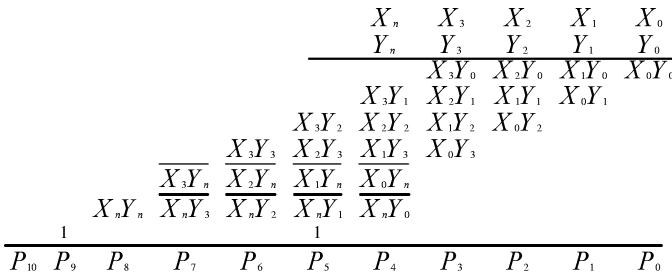


Figure 5 Matrix of Universal Multiplier

The extended multiplication expressed by (25) can be used to process unsigned, signed magnitude and two's complement

numbers by the Universal Multiplication Unit. Therefore, if the UMU receives two numbers X and Y of length n:

$$X_{n-1} X_{n-2} X_{n-3} \cdots X_2 X_1 X_0 \quad (26)$$

$$Y_{n-1} Y_{n-2} Y_{n-3} \cdots Y_2 Y_1 Y_0 \quad (27)$$

And, for these data inputs, X_{n-1} and Y_{n-1} become the MSBs when working with unsigned numbers, but represent the sign bit when working with signed magnitude and two's complement representations. Therefore, it is necessary to correctly assign, via the Universal Control signals (UC), the different terms expressed in (25) for the processing of multiplications with unsigned, sign magnitude and two's complement representations; this functionality is achieved through the use of multiplexers. A block diagram of the Universal Multiplier Unit is shown in figure 6.

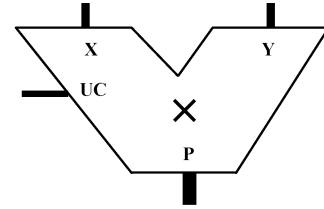


Figure 6 Block Diagram of Universal Multiplier.

A. Unsigned Number processing

The processing of unsigned numbers is achieved in a UMU by filling with zeros the positions corresponding to the extra bits X_n and Y_n . As stated in equations 18 to 21 the product will be:

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j 2^{i+j} + 2^{2n} + 2^{2n} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j 2^{i+j} \quad (28)$$

Equation (28) represents the outcome for the first $2n$ least significant bits. At position $2n+1$ the $1 \otimes 1$ will be zero therefore the result is correct.

B. Signed magnitude processing

The processing of signed magnitude numbers in a UMU is accomplished by filling the extra bits X_n , Y_n and also the X_{n-1} , Y_{n-1} bits with zeros; the result is depicted by the following equation:

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j 2^{i+j} + 2^{2n} + 2^{2n} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j 2^{i+j} \quad (29)$$

A correction should be made at the outcome's position $2n$ in order to obtain the correct sign of the multiplication. This is implemented with the " $X_{n-1} \otimes Y_{n-1}$ " operation.

C. Two's Complement Processing

The processing of two's complement representation by the UMU is achieved copying the X_{n-1} , Y_{n-1} bits into the extra X_n , Y_n bits respectively. Table 3 presents a summary of the multiplexer's correct assignment at the input and the outcome of the UMU.

Table 3 Universal Multiplier Control (UC)

UC1	UC0	Input	Output
0	0	Unsigned $X_n \leftarrow 0$ $Y_n \leftarrow 0$	Unsigned Take the $2n-1$ LSBs
0	1	Signed $X_n = X_{n-1} = 0$ $Y_n = Y_{n-1} = 0$	Signed Take the $2n-1$ LSBs and fill the sign position with $P_{2n} \leftarrow X_{n-1} \otimes Y_{n-1}$
1	X	Two's Complement $X_n \leftarrow X_{n-1}$ $Y_n \leftarrow Y_{n-1}$	Two's Complement Take the $2n$ LSBs.

IV. ANALYSIS AND EXPERIMENTAL RESULTS

The Xilinx ISE 6.1 logic design environment [12] was used to implement several multipliers with a word length of 16 bits into the chosen unit device XC2S200E [13]. The mapped multipliers include: the CSA Unsigned Array, Baugh-Wooley, Wallace-Booth and the Universal Multiplier Unit. Since this paper is focused in performance evaluations of each structure reviewed against the UMU organization, the synthesis results presented include number and percentage of used slices in the target device, as well as the number of LUT [14] and IOBs. The summary of resources used in the FPGA for the implementation of the studied structures is depicted in table 4.

Table 4 Silicon use in different multipliers and UMU

Unit	# Slices	# LUT	# IOBs
Unsigned	284 12%	495 10 %	64 43%
Baugh & Wooley	330 14%	578 12%	65 44%
Wallace-Booth	366 15 %	641 13%	64 43 %
UMU	334 14%	584 12%	67 45%

The results obtained by the different structures mapped onto the FPGA device in terms of speed, including the maximum combinational path delay, denominated gate delay (logic delay), and the routing delay (wire delay) are illustrated in table 5.

Table 5 Time delays in different multipliers and UMU

Unit	Logic Delay	Wire Delay	Total Delay	Levels of Logic
Unsigned	17.735 ns 34.4%	33.813ns 65.6%	51.548ns 100%	32
Baugh & Wooley	18.153ns 33.7%	35.730ns 66.3%	53.883ns 100%	33
Wallace-Booth	13.973 ns 33.5%	27.702ns 62.5 %	41.675ns 100 %	23
UMU	18.989ns 32.9%	38.650ns 67.1%	57.639ns 100%	35

As expected from the theoretical background [1] [2] [11] [15], the obtained results presented on table 5 shown a Wallace-Booth multiplier as the fastest structure studied.

Compared with the other structures, the UMU presents an additional delay of 7 and 12 % against the Baugh&Wooley and Unsigned multiplier respectively. In addition, the results

presented in table 4, show that the UMU consumes an additional 18 % of LUT resources compared with the unsigned implementation, practically the same amount compared with the Baugh and Wooley and a 9 % less than the Wallace Booth implementation.

A simple but significant Area-Time product presented by the UMU structure and the traditional multipliers is depicted in table 6

Table 6 Basic Area-Time product for different Structures

Unit	Time Delay	# Slices	Figure # Slices*Time Delay
Unsigned	51.548 ns	284	1.0
Baugh & Wooley	53.833ns	330	1.21
Wallace-Booth	41.675 ns	366	1.04
UMU	57.639 ns	334	1.31

The comparison base is the unsigned multiplier, which corresponds to the better Area-Time figure off the four structures. It could be brought up to observation that the UMU's extra delays introduced by the multiplexer for the processing of unsigned, sign magnitude and two's complement representation are considered to be an acceptable overhead.

V. CONCLUSIONS AND FUTURE RESEARCH

The Universal Multiplication Unit for the processing of unsigned, signed numbers and two's complement has been presented. In this approach the two's complement product of two numbers with an augmented position is used to process other notations as is depicted in equation (25), and resultant increment of silicon and delay is tolerable.

Currently, computer architectures consider several representations for data processing, specially when processing multimedia applications [4], and due this necessity, the design of arithmetic units capable of operating with different notations is an important issue for nowadays processors. Furthermore, new paradigms, like reconfigurable computing, are guiding to the design and use of flexible Custom Computing Units, like the Universal Multiplication Unit. The inherent potentiality of a UMU will increase over the traditional multipliers when used as a Custom Computing Unit on a Reconfigurable Processor like MOLEN [17]. The partial reconfiguration paradigm will improve the area-time performance; therefore, handling multiple types of operands, harnesses the potential utilization of this unit in several applications, paying a poor silicon and delay overhead.

References

- [1] Behrooz Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press Inc, 2000 ISBN 0-19-512583-5
- [2] Israel Koren, Computer Arithmetic Algorithms, A. K. Peters, Ltd, 2002, ISBN: 1-56881-160-8
- [3] V. Betz and J. Rose, "Circuit Design, Transistor Sizing and Wire Layout of FPGA Interconnect," IEEE Custom Integrated Circuits Conference, San Diego, CA, May 1999, pp. 171 - 174.

- [4] H. Calderón, S. Vassiliadis, Computer Graphics and the MOLEN Paradigm: A Survey, Proceedings of ProRISC, pp. 23-36, Veldhoven, The Netherlands, November 2003
- [5] S. Vassiliadis, J. Hoekstra and H. T. Chiu, Array multiplication scheme using (p, 2) counters and pre-addition, Electronic Letters, Vol. 31 No. 8, pp 619-620 April 1995.
- [6] Pezaris, S. D., A 40-ns 17 bit-Array Multiplier, IEEE Transactions on Computers, Vol. 20, pp. 442-447, April 1971.
- [7] Andrew D. Booth, A signed Binary Multiplication Technique, The Quarterly J. Mechanics and Applied Mathematics, vol. 4, pp. 236-240, 1951.
- [8] O. L. MacSorley, High-speed Arithmetic in binary computers, Proceedings IER, 99, 67-91, (January 1961).
- [9] C. S. Wallace, "A Suggestion for a fast multiplier", IEEE Transactions on Electronic Computers, Vol. EC-13, pp. 14-17, February 1964.
- [10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Introduction to Algorithms, MIT Press, 1989, USA. ISBN 0-262-03141-8
- [11] Mori, J., et al., "A 10-ns 54- x 54-bit parallel structured full array multiplier with 0.5- μ m CMOS technology," IEEE J. Solid-State Circuits, Vol. 26 pp. 600-606, April, 1991.
- [12] Synthesis and Verification Design Guide, Xilinx Inc., http://www.xilinx.com/support/sw_manuals/xilinx6/download/
- [13] Spartan-IIIE 1.8V FPA Family: Functional Description, DS077-2 V2.0, Xilinx Inc., November 12, 2002
- [14] Stephen Brown and Zvonko Vranesic, Fundamental of Digital Logic with VHDL design, MCGraw-Hill, 2000 ISBN 0-07-012591-0
- [15] Lakshmanan, Masuri Othman, Alauddin Mohd Ali, High Performance Parallel Multiplier using Wallace-Booth algorithm, Journal of Microelectronics, ICSE 2002 Malaysia.
- [16] C.R. Baugh and B.A. Wooley, "A Two's complement parallel array multiplication algorithm", IEEE, Transactions on Computers, vol. 22, pp. 1045--1047, December 1973.
- [17] Stephan Wong, S. Vassiliadis, S.D. Cotofana, Future Directions of Programmable and Reconfigurable Embedded Processors, Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation, pp. 235-257, January 2004