Calculation Methodology for Flexible Arithmetic Processing

Abstract

A new operation model of flexible calculation that allows us to adjust the operation delay depending on the available time is presented. The operation method design uses look-up tables and progressive construction of the result. The increase in the operators' granularity opens up new possibilities in calculation methods and microprocessor design. This methodology, together with the advances in technology, enables the functions of an arithmetic unit to be implemented on the basis of techniques based on stored data that provide quality results and systematization in the implementation.

The proposed techniques are applied in the design of a multiplier operator. We report an evaluation of the architecture in area, delay and computation error, as well as a suitable implementation of an application example in FPGA to validate the design.

1. Introduction

In the last years a spectacular development of the scientific and commercial applications has taken place. Although calculation performance has grown to a considerable speed, there are a great number of applications that overflow the processing possibilities of the most advanced systems. In addition, other computational problems are difficult to fit into the rigid schemes of the calculation of conventional arithmetic architectures [1], [2], [3].

We can find several examples in which an intensive processing of data provided by peripheral takes place. In these cases, a strong coordination among sensors and the rest of system is necessarily produced. For example, in systems of mobile objects guidance, when the speed of the object is increased, the system has less time to process the information that is received from the sensors and to make decisions about its movement. In this application, a fast answer in appropiate time that allows decisions to be made at every moment may be advisable, with the drawback of less precision in the results.

For these applications it would be advantageous to have a calculation methodology allowing the design of high performance operators with control on the results and act on the even quality of the result and processing delay based on the specific computational requirements of each case. The progressive improvement in performance provided by advances in electronic technology justifies the search for new proposals that would probably have been prohibitive some time ago.

In this work, we present a flexible calculation model that supplies variable quality of the result based on the available time. The methodology must make good use of the operands' structure, provides strategies that contribute determinism in the response time and, at the same time, allows parallel designs.

We propose a flexible method of calculation of the arithmetical multiplication as an example of this methodology and its use in a realistic application.

This paper is structured in the following way: in section 2 we establish the operator's design principles and describe the architecture for the flexible arithmetic unit. Next, in section 3 the flexible multiplication algorithm is proposed based in that principles and evaluated under the scope of the area cost, delay and computation error. Finally, we present a suitable application example of the proposed method.

2. Design principles

The design objective is to conceive a calculation method of the mathematical functions that supplies to the processor with flexible adjustment in time and precision. The proposal consists of the combination of two techniques: using precalculated data in look-up tables and obtaining the result in a successive processing way.

The construction of operators based on the following design principles will depend on the characteristics of each function, and it is not applicable in all the cases. Nevertheless, in those operations in which it is possible take place advantages relative to the performance and flexibility of the processing.

2.1 Design based on Look-up tables

Traditionally, most elementary operators consider a bit to be the minimum unit of information that can be processed. They are called bit-to-bit operators. A new step in their evolution consists of increasing the granularity and taking a group of bits as the minimum unit of operation. For some functions, this calculation model offers a superior performance to the classic bit-to-bit operations and is therefore an advance in the design of high performance and quality operators.

In this study, the elementary operators that take a number of k bits as the minimum unit of information that can be processed are called k-operators. The fundamental idea consists of obtaining advantages in the design of the generic arithmetic operators by using k-operator elements in their construction.

Figure 1 shows schematically the functionality of a generic k-operator.

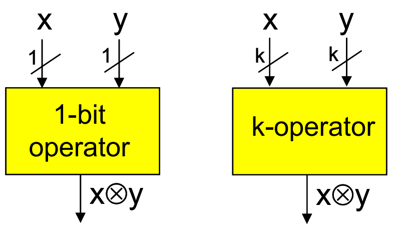


Figure 1: Operator comparison

The proposed design of the k-operators consists of using look-up tables (LUT —*Look Up Table*) to make the effective calculation. In that way, for any pair of blocks of k bits, a memory structure contains the direct result of the operation. These look-up tables must store all the results for k-size operands, so that it is only necessary to select the cell which contains the result. The operand's value itself is used to address the table [4]. The nature of the stored data will depend on the function to be calculated.

Figure 2 schematically shows a k-operator multiplier based on memory designs.

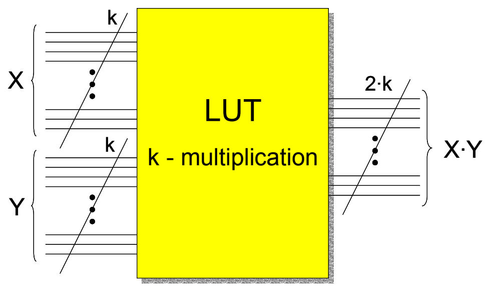


Figure 2: Multiplication k-operator

The implementation of the k-operators using look-up tables provides a greater density of VLSI integration than in the other combinational methods. In addition, LUTs have interesting characteristics relating to real-time processing: they work in a totally determinist way and they can incorporate error detection and correction mechanisms. The treatment of the operands in small blocks promotes segmentation and a high level of parallelism. These construction capabilities provide robustness and flexibility to the operations [5].

The area cost of the k-operators based on look-up tables grows exponentially with the operand length. Because of this, the use of spatial complexity is limited to general cases and affects the desirable value of k that maximizes performance. A compromise formula between the value of k and the size of the memory must be found. Nevertheless, advances in technology play a crucial role in performance improvement and in the reduction of the temporal delay.

The use of tables in the computation of functions is a well-known technique. In arithmetic literature, several implementations of elementary functions based on the use of look-up tables are analyzed [6], [7]. In this approach, we use look-up table to process directly an elemental operation. For performance reasons, it is assumed that the LUT memory is implemented in the circuitry of the Arithmetic Unit itself, thus reducing communication costs [8].

2.2. Successive result processing

Response quality is related to the number of calculated stages of the operations, and therefore, will be able to act on the time-quality-parallelism relationship. This approach forms a new architecture that will implicitly incorporate flexibility in order to adapt the duration of the calculation to time availability, which is the instrument for real-time management. This characteristic provides capabilities for successive refinement of the solution.

Smaller process time is translated in smaller precision in the results. This methodology cannot be generalized for all the operations nor does it offer reliability in all cases; each operation must be analyzed to verify the benefits.

3. Flexible Aritmetic Unit

The implementation objective is the conception of a flexible arithmetic unit whose operations are based on the previous design principles. The projected arithmetic unit carry out five operations: addition, multiplication, division and square root.

For the calculation management, the arithmetical unit has an operation control module that establishes the delay-precision of the operation and translates the application requirements into the number of processed stages. This operation control module consists of a combinational circuit that has the problem conditions in its inputs and a number of operation stages in its outputs, for example, a coder or multiplexor. Another design of this module may be a table look-up that stores the amount of stages to calculate in each situation.

The figure 3 illustrates the unit's design.

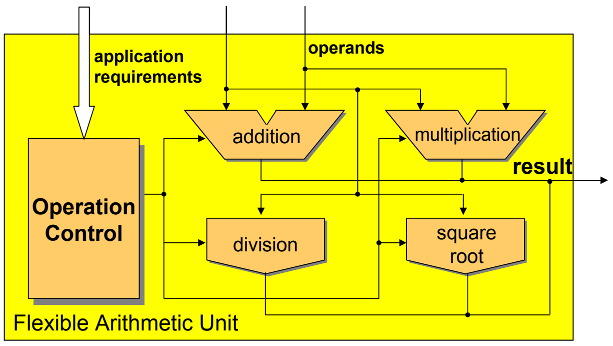


Figure 3: General structure of the arithmetic unit

In the rest of this work the multiplication operation based on these design principles is exposed. We are currently also investigating the adjustment to the flexible calculation of the all those operations. They will be presented in following works.

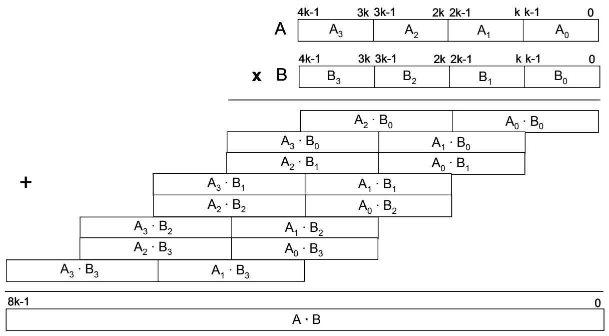
4. Multiplication operation

4.1 Algorithm

The calculation method is made up of the following steps [9]: generation of partial products, reduction in the number of partial products and final addition.

1. Generation of partial products: the partial products generation process is crucial to the operation's overall performance. Two aspects must be taken into account in its design: the generating circuit's complexity and the number of generated partial products. The first aspect is linked to the time taken in generating each partial product, whereas the second one affects the time invested in subsequently combining them to make up the final result. Both aspects are opposed, that is improving one it can mean to make worse the other.

The technique proposed is based on LUT access with precalculated products. The method consists of fragmenting the numbers to be multiplied into k bit blocks and obtaining the product of each pair of blocks directly from the *LUT k-multiplication* table that shows figure 2 to form the whole of partial products. Figure 4 shows partial products for operands divided in 4 parts.



4k-1 3k 3k-1 2k 2k-1 k k-1 0
A A_3 A_2 A_1 A_0
x B B_3 B_2 B_1 B_0
$A_2 \cdot B_0$ $A_0 \cdot B_0$
$A_3 \cdot B_0$ $A_1 \cdot B_0$
$A_2 \cdot B_1$ $A_0 \cdot B_1$
+
$A_3 \cdot B_1$ $A_1 \cdot B_1$
$A_2 \cdot B_2$ $A_0 \cdot B_2$
$A_3 \cdot B_2$ $A_1 \cdot B_2$
$A_2 \cdot B_3$ $A_0 \cdot B_3$
$A_3 \cdot B_3$ $A_1 \cdot B_3$
8k-1 0
$A \cdot B$

Figure 4: Generation of partial products

The last partial product can be placed to the left of the first one. For n-bit size operands, we can express the number of partial products generated according to the expression $\lceil 2n/k \rceil$-1.

With this technique, a lower number of partial products are obtained compared with other known methods, as the following table shows.

n	Simple Generation	Booth2 [8]	LUT-Prod k=4	LUT-Prod k=8
8	8	5	3	1
16	16	9	7	3
32	32	17	15	7
64	64	33	31	15

Table 1: Quantity of partial products generated

2. Reduction in the number of partial products: the general way in which a high performance multiplier works consists of combining the partial products in order to reduce their number until a total of two is reached, which will be added in the last stage of the multiplier. There are several methods of reduction of partial products [9], [10]. In this work a Wallace-tree reduction is used based on 3:2 counters [11].

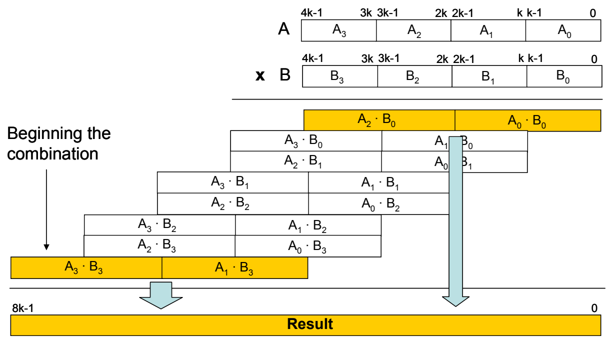
3. Final addition: It is implemented by anyone of the known addition methods [12].

4.2 Flexible multiplication

The partial product generation stage must be carried out completely due to the fact that, on the one hand, it is the starting point for the following steps, and on the other, because the cost of generating all the partial products is constant.

The proposed flexible method consists of beginning the combination of generated partial products from an initial point according to the time available. Since the application requirements already are contemplated in the partial product reduction stage, the addition of the result of the combination will be made completely.

Thus, for most restricted case, the first and the last partial product generated are ready to go on directly to the result. See figure 5.



4k-1 3k 3k-1 2k 2k-1 k k-1 0
A A_3 A_2 A_1 A_0
x B B_3 B_2 B_1 B_0
Beginning the combination
$A_2 \cdot B_0$ $A_0 \cdot B_0$
$A_3 \cdot B_0$ $A_1 \cdot B_0$
$A_2 \cdot B_1$ $A_0 \cdot B_1$
$A_3 \cdot B_1$ $A_1 \cdot B_1$
$A_2 \cdot B_2$ $A_0 \cdot B_2$
$A_3 \cdot B_2$ $A_1 \cdot B_2$
$A_2 \cdot B_3$ $A_0 \cdot B_3$
$A_3 \cdot B_3$ $A_1 \cdot B_3$
8k-1 0
Result

Figure 5: Result produced for the first selection

For other cases, partial products are combined and the final addition is carried out. See figure 6.

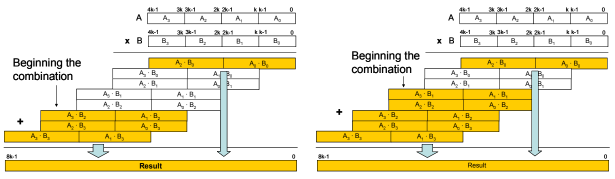


Figure 6: Result produced for the 2nd and 3rd selection

In this way, we obtain an imprecise result in less processing time than we need to carry out the complete operation of combination and final addition of partial products.

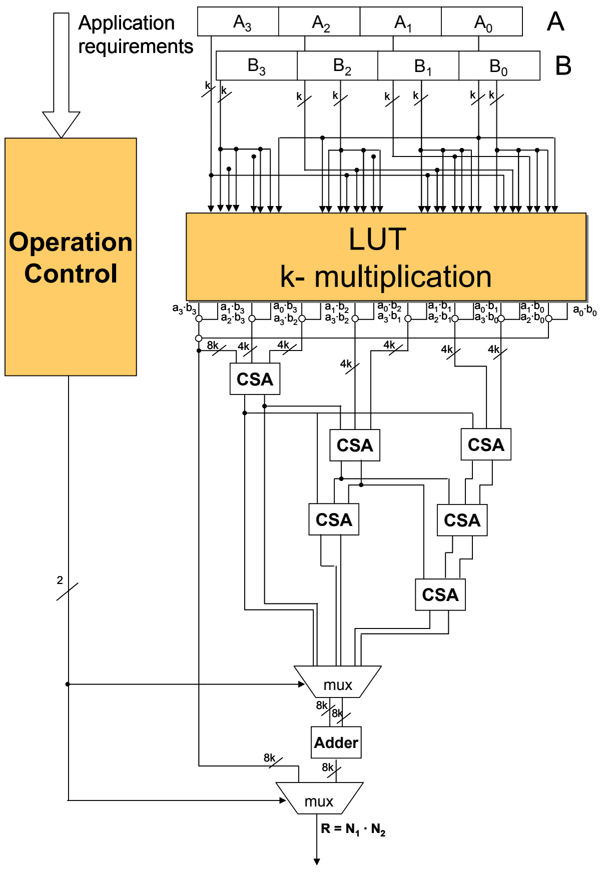
The amount of possibilities depends on the length of the operands and the operation control module. This last one transforms the requirements of the problem into a discreet position of combination beginning.

5. Architecture

This architecture is suitable for specific purpose applications where time restrictions are present.

5.1 Design

Figure 7 shows the proposed arithmetic unit architecture for multiplication operation. We assume, for example, the numbers are fragmented into 4 blocks of k bits.



Application requirements
A_3 A_2 A_1 A_0 A
B_3 B_2 B_1 B_0 B
k
Operation Control
LUT k- multiplication
$a_3 \cdot b_3$
$a_1 \cdot b_3$ $a_2 \cdot b_3$
$a_0 \cdot b_3$ $a_3 \cdot b_2$
$a_1 \cdot b_2$ $a_3 \cdot b_2$
$a_0 \cdot b_2$ $a_3 \cdot b_1$
$a_1 \cdot b_1$ $a_2 \cdot b_1$
$a_0 \cdot b_1$ $a_3 \cdot b_0$
$a_1 \cdot b_0$ $a_2 \cdot b_0$
$a_0 \cdot b_0$
8k
4k
CSA
2
mux
Adder
$R = N_1 \cdot N_2$

Figure 7: Block diagram of the proposed architecture

The main features of this architecture are:

- The *Operation Control* module consists of a combinational circuit that has application conditions in its inputs and a number of operation stages in its outputs.
- Access to the *LUT k-multiplication* provides the partial results for all the pairs of blocks.
- The partial product reduction is carried out by means of a tree scheme based on *Carry Save Adders*.
- Several results with different qualities of degree and delay are extracted from operation circuits. There are 4 execution paths that provide results with different precision degree with respect to the exact retult and with different time delay.
- The *Operation Control* circuit selects the partial result that best fits the conditions of the problem.

5.2 Path time

The path time of the proposed architecture is set by the delay of the slowest path in the circuit. Depending on the application requirements and the selected operation, it can be any of the main paths of the implementation. There are the same number of paths as possibilities of result selection. These paths, as shown in figure 7, are the following:

- Max(Op.Control, LUT-Multiplier) + mux
- Max(Op.Control, LUT-Multiplier) + CSA + Addition + 2mux
- Max(Op.Control, LUT-Multiplier) + 3CSA + Addition + 2mux
- Max(Op.Control, LUT-Multiplier) + 4CSA + Addition + 2mux

Apparently, the improvement in time of one incomplete operation is not very significant, nevertheless, when the amount of the multiplications to be made is elevated, the architecture acquires a greater relevance.

6. Evaluation of the proposed architecture

In this section, we present estimates of the area costs, execution time and error computation of the architecture proposed in the previous section. The power consumption of the circuit is not important for this research and is therefore not dealt with in this paper. It will be studied in depth in the event that it can be implemented in a chip.

6.1 Area estimations

The model we use for the area estimations is taken from [7], [13] and [14] as well as of our own estimations and experiments.The unit used is the size of a complex gate τ_a, since the area of the compound LUT-multiplication, selection circuits, adder and multiplexors are easily expressed in this unit.

The main contributions to the area of the architecture come from the compound LUT k-multiplication. The area of the other components of the circuit is small when compared to the area of the LUT, therefore, the estimation is focused only on the LUT k-multiplication.

Data storage imposes severe restrictions on k block size. Table 2 shows the cost for the most common sizes.

k	Size (bits)
4	256 B
6	6 KB
8	128 KB

k	Size (τ_a)
4	$\approx 70\ \tau_a$
6	$\approx 1441\ \tau_a$
8	$\approx 20478\ \tau_a$

Table 2: LUT k-multiplication size

As we can see, the area cost increases exponentially with the k value. Therefore, we have to achieve a balance between the memory required and the complexity of the circuit. The model assumes a 40 τ_a/Kbit rate for tables addressed by words of up to 6-bits long, a 35 τ_a/Kbit rate for 7-11 input bit tables, a 30 τ_a/Kbit rate for 12-13 input bit tables and 25 τ_a/Kbit one for 14-16 input bit tables.

As shown in the previous table, the amount of area is much greater than in conventional multiplier designs based on simple combinational circuits, nevertheless, this architecture is still suitable for applications in which the size of the circuit is not a problem.

6.2 Delay estimations

Delays in a complete operation calculus are divided into *access to the LUT k-operator* and *combination of partial results*.

Let τ_t be the delay of a complex gate, such as one full-adder. According to [7], [14] analysis[1] we assume a delay of about $T_{LUT} = 3.5\tau_t$ for 8 input bit tables, $T_{LUT} = 5\tau_t$ for 12-13 input tables and $T_{LUT} = 6.5\tau_t$ for 16 input bit tables. We suppose a concurrent access of all the operand's blocks in a multiport memory.

The delay estimations for each module of the propose architecture are:

- *Access time to the LUT k-multiplication to obtain the precalculated partial products*. This time will only be determined by memory access time T_{LUT}.
- *Reduction of partial products:* Let T_{CSA} be the time taken in the Carry Select Adder that reduces 3 partial results to 2. According to [9] design, a reduction step consists of two complex gates τ_t, so $T_{CSA} = 2\ \tau_t$.
- *Final Addition*: We consider anyone addition method. The standar adder delay depends on the operand's length. According [12], addition delay is $T_{Adder} = \tau_t \lg n$.

The comparison of the propose methodology with other algorithms is focused in the first part of the operation. Due to the fact that the different methods generate a different quantity of partial products, in order to compare them homogenously, we reduced the number of products generated by including stages of 3:2 counters.

Table 3 shows the results obtained in terms of costs of complex gate levels and LUT access time. The table access model is shown as LUTP-x, in which the index indicates the k size. We assumed an arbitrary operand size.

Method	Generation delay	Reduction stages (3:2)	Total delay
Simple	0.5 τ_T	5	10.5 τ_T
Booth2 [8]	2 τ_T	3	8 τ_T
LUTP-4	3.5 τ_T	2	7.5 τ_T
LUTP-6	5 τ_T	1	7 τ_T
LUTP-8	6.5 τ_T	0	6.5 τ_T

Table 3: Homogenous comparison between the partial product generation methods

[1] Implementation using a family of standard gates from the AMS 0.35 μm CMOS library

The LUT k-multication module performance plays a fundamental role in the calculation time. Therefore, the look-up table is proposed as part of the arithmetic unit itself in order to accelerate its access. These table sizes are easily handled with the rest of the arithmetic unit by today's high speed, high density memories and it may already be possible to implement the entire hardware on a single chip.

In addition, the circuit delays depend on the technology used and on the implementation itself. In order to prove this, the LUT k-multiplication have been implemented in VHDL and tested on FPGA[2]. The LUT implementation corresponds to the design presented in [15], [16], and has been integrated into the selection circuit. Table 4 shows the results obtained after the synthesis and simulation of each k-multiplication for some k wordlength in homogenous comparison.

Method	Generation delay	Reduction stages (3:2)	Total delay
Simple	0.573	4.704	5.277
Booth2 [8]	2.181	2.352	4.533
LUTP-4	2.754	2.352	5.106
LUTP-6	2.946	1.176	4.149
LUTP-8	3.132	0	3.132

Table 4: Delays of partial product generation in homogenous comparison (ns)

The previous results demonstrate that the proposed k-operator design presents a delay similar or better to the conventional designs for this particular implementation, and they show the technology's high degree of dependency on performance.

The total delay of the complete operation for several operand sizes n is shown in the following table in term of complex gate delay:

n	Generation delay, k=8	Reduction of partial product	Final addition	Total delay
16	6.5 τ_t	2 τ_t	4 τ_t	12.5 τ_t
32	6.5 τ_t	8 τ_t	5 τ_t	19.5 τ_t
64	6.5 τ_t	12 τ_t	6 τ_t	24.5 τ_t

Table 5: Complex gate delay of multiplication

Complex gate delay will take different values according to the implementation technology used.

6.3. Error computation analysis in flexible multiplication

With the objective of testing the error computation while ignoring the correct combination of all the sequence of partial results, an exhaustive set of experiments has been made to prove the method for all cases. The experiments have been carried out by means

[2] See appendix A.

of a simulation in a C programming environment[3]. They are both in individual operations and in sequences of successive operations.

The profile of the experiments is the following: A LUT 8-multiplication, operand size n = 64 bits and operands are rational numbers at the interval [0, 1). To process the complete number 8 stages are needed.

6.3.1. Independent multiplications

This test consists of calculating the average error rate in 10^7 multiplications of two random rational numbers. Figure 8 shows a graph of error evolution in stages. The dotted line indicates the limit of precision for numbers represented according to the IEEE 754 double precision standard [17].

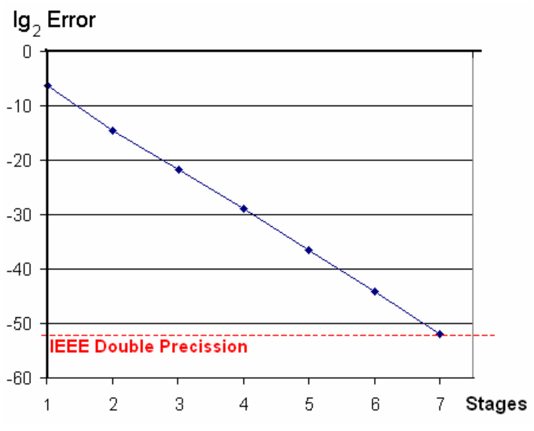


Figure 8: Evolution of the error average in independent multiplications

The following figure shows the error dispersion. It is located around the interval that depends on the amount of processed stages.

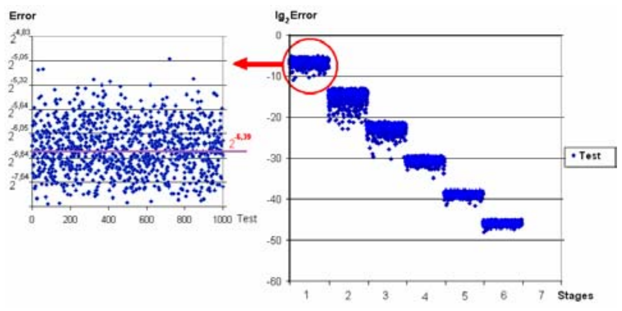


Figure 9: Error dispersion in incomplete operations

6.3.1. Successive multiplications

This test is aimed at empirically analyzing error propagation while processing inaccurate values consecutively. The error average is calculated in 1,000 sets of 1,000 successive operations of random rational numbers within the interval [0,1) for each of the operation's loops. The numbers are generated at a positive interval, so they do not compensate positive errors with negative ones in the successive operations. The graph in figure 10 shows the results obtained in this test:

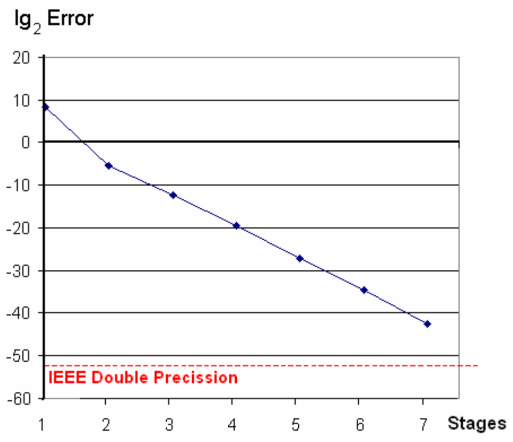


Figure 10: Evolution of the error average in succesive multiplications

The previous figures graph the decreasing tendency of the error with the number of operation stages. With the sufficient stages the wished precision is reached with a saving of time with respect to the complete operation.

7. Application example

This section describes a simple example that illustrates the specific application of the proposed architecture.

The experiment is located in the *Specialized Processing Architectures and Communications Systems* research groups of the *University of Alicante, Spain*[6]. One of the most interesting lines of research consists of the development of a *Real-Time processor* that considers the temporal restrictions in the low level of the architecture [4]. It is usefulness in many interest applications: calculation of trajectories for moving bodies, guidance and positioning systems, high frequency communications etc.…To process this successfully, the flexibility and determinism of the calculations plays a fundamental role in the correct working of the processor.

Let there be an object that moves according to a vector position $\vec{s}$. The application consists of calculating the scalar product of this vector in relation to a reference vector $\vec{r}$.

$\vec{r} = (r_x, r_y, r_z)$; $\vec{s} = (s_x, s_y, s_z)$;

$$\vec{r} \cdot \vec{s} = r_x s_x + r_y s_y + r_z s_z \qquad (1)$$

[3] C++ Builder 5.0 Professional. Borland software corporation. http://www.borland.com

[6] http://www.ua.es/i2rc/

We proposed the following architecture to resolve the expresion vector multiplication (1):

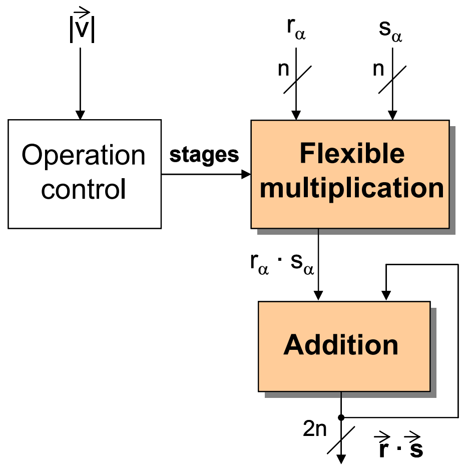


Figure 11: Architecture for the calculation of the scalar product

The product of the individual components is made sequentially by flexible multiplication operator In the time of LUT k-multiplication, the operation control module decides the number of stages that will be made in the reduction step of multiplication operations. The criterion is based on the speed of the moving object. At greater speed, fewer partial results are selected.

For example, we assume: $x_i \in [0, 1)$, $|\vec{v}| \in [0, 150]$, n = 32 bits, k = 8.

Table 6 shows the number of selection stages, the time-saving that takes place in the three multiplications and the computation error. Simulation is made in FPGA for a set of 1000 series of operations.

$\lvert\vec{v}\rvert$	selection stages	delay (ns)	Saving of time (ns)	|Error|
[0, 32)	4	28.88	0,0	$2^{-30,91}$
[32, 64)	3	23.23	5.65 (19.5%)	$2^{-22,97}$
[64, 96)	2	16.07	12.81 (44.3%)	$2^{-14,82}$
[96, 150]	1	14.02	14.6 (50.8%)	$2^{-6,89}$

Table 6: Average experimental results

The simulation results demonstrate that this technique saves considerable time in cases in which a fast response is necessary. Error is maintained within the acceptable margins. Although the value of the scalar product is not obtained with absolute precision, the result can be sufficient to make a decision.

8. Conclusions

The following conclusions have been drawn from the research described in this paper:

- The use of precalculated results in stored logic allows the construction of fast operators comparable to existing methods and lays the foundations for the design of high performance architectures. At the same time, the new calculation method offer inherent advantages due to memory structure: flexibility, robustness, parallelism and reusability. The implementation example of a simple operation based on these ideas demonstrates these improvements.
- The partial product generation by means of LUT k-multiplication equals or improves other common methods of generation. In addition, technological improvements in manufacture or in communication with the selection circuit will tend to reduce LUT access time and, therefore, total operation time.
- The operator's flexible behavior, which produces more and more precise results as the number of iterations increase, is suitable for the construction of systems with temporal/precision restrictions, in which result quality is exchanged for response determinism and speed.
- Finally, the error analysis carried out shows that the algorithm provides limited results in multiplication operations, even in cases in which successive calculations are made with imprecise operands.

9. Acknowledgments

We want to thank *Comunications Systems* research group of *Computer Technology and Architecture Department* of *University of Alicante* for providing positioning system model as well as for discussions on methodology calculation.

This work is being backed by grant DPI2002-04434-C04-01 from the *Ministerio de Ciencia y Tecnología* of the Spanish Government.

10. References

[1] W. A. Halang, K. M. Sacha. "Real-Time Systems. Implementation of Industrial Computerised Process Automation." *World Scientific Publishing Co*. 1992.

[2] G. C. Butazzo. "Hard Real-Time Computing Systems. Predictable Scheduling Algoritms and Applications." *Kluwer Academic Publishers*. 1997.

[3] F. Pujol, F.J. Ferrández, J.M. García. "A new Method for Position Location in Random Media", *8th International Conference on Electromagnetics of Complex Media*, Lisboa (Portugal), 2000.

[4] J.M. Mora Pascual, "Real-Time Floating Point Arithmetic Unit", PhD Thesis, University of Alicante, 2001.

[5] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", *Oxford University Press*, 2000.

[6] W.F. Wong, E. Goto, "Fast Hardware-Based Algorithms for Elementary Function Computations Using Rectangular Multipliers", *IEEE Transaction on Computers*, vol. 43 (4), 278-294. 1994.

[7] M.D. Ercegovac, T. Lang, J-M. Muller, A Tisserand, "Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers", *IEEE Transaction on Computers*, vol 49, 628-636, 2000.

[8] S. Carr. "Memory Hierarchy Management". *PhD Thesis*, Rice University. 1993.

[9] G.W. Bewick. “Fast multiplication: Algorithms and implementation”. *PhD Thesis*. Dept. of Electrical Engineering, Stanford University. 1994.

[10] V.G. Oklobdzija, D. Villeger, S.S. Liu. “A method for speed optimised partial product reduction and generation of fast parallel multipliers using an algorithmic approach”. *IEEE Transactions on Computers*. Vol. 45, no. 3, 1996.

[11] C.S. Wallace. “A Suggestion for a Fast Multiplier”. *IEEE Trans. Computers,* vol. 13, no 2. 1964.

[12] R. Zimmermann. “Binary Adder Architectures for Cell-Based VLSI and their Synthesis.” *PhD Thesis, Swiss Federal Institute of Technology*. 1997.

[13] W.F. Wong, E. Goto, “Fast Hardware-Based Algorithms for Elementary Function Computations Using Rectangular Multipliers”, *IEEE Transaction on Computers*, vol. 43 (4), 278-294. 1994.

[14] J-A. Piñeiro, M.D. Ercegovac, J.D. Bruguera. “High-Radix Logarithm with Selection by Rounding”, *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP’02)*, 2002.

[15] S.J.E. Wilton, N.P. Jouppi. “An Enhanced Access and Cycle Time Model for On-Chip Caches”. *Digital Western Research Laboratory*. 1994.

[16] H. Nambu, K. Kanetani, K. Higeta, M. Usami, T. Kusonoki, K. Yamaguchi, N. Homma. “A 1.8 ns Access, 5550 Mhz 4.5 Mb CMOS SRAM”. *IEEE ISSCC*. 1998.

[17] American National Standards Institute and Institute of Electrical and Electronic Engineers. “IEEE Standard for Binary Floating-Point Arithmetic”. *ANSI/IEEE Standard 754*. 1985.

Appendix A: Xilinx FPGA

Hardware design and simulation techniques and their subsequent implementation in reconfigurable systems such as FPGAs, enable valid designs to be made and a high productivity in development to be obtained. The results of this study are focused on designs implemented on standard FPGAs, specifically on the Xilinx family of FPGAs. These devices have had a wide acceptance and are used by the scientific community as a whole for processing results.

In spite of the fact that the speed of these devices is not very high (up to 50 MHz), by choosing them we aim to establish a homogeneous basis for comparing the addition algorithms and draw conclusions from this comparison. Using one of these devices, the simulation and testing of the proposed design have been carried out with definition and simulation free software *Xilinx WebPack 3.2XE*n found at http://www.xilinx.com.

On the development platform, the architecture proposed in VHDL code and that of other known architectures, has been developed and the results obtained by the software itself on the execution times have been compared.