Time-Precision Flexible Arithmetic Unit

J. M. García Chamizo, J. Mora Pascual, H. Mora Mora, M. T. Signes Pont
Department of Computer Science Technology and Computation
University of Alicante, Spain
{juanma, jeronimo, hmora, teresa}@dtic.ua.es

Abstract[1].- **A new conception of flexible calculation that allows us to adjust an operation depending on the available time computation is presented. The proposed arithmetic unit is based on this principle. It contains a control operation module that determines the process time of each calculation.**

The operation method design uses precalculated data stored in look-up tables, which provide, above all, quality results and systematization in the implementation of low level primitives that set parameters for the processing time. We report an evaluation of the architecture in area, delay and computation error, as well as a suitable implementation in FPGA to validate the design.

Index Terms— **Arithmetic unit, Real-Time Systems, special architectures, table Look-up**

I. Introduction

There are a great number of applications that are difficult to fit into the rigid schemes of the calculation of conventional arithmetic architectures. For these applications it would be advantageous to have operators that provide control on the results and act on the even quality of the result and processing cost based on the specific computational requirements of each case [1], [2].

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, although at the expense of less precision in the results.

In this paper, we propose a calculation method of arithmetic operations. Its main characteristic is the variable quality of the result based on the available time. The algorithm is based on the use of strategies that contribute determinism to the response time and, at the same time, allow for parallel designs.

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

II. Design Principles

The objective is to obtain a calculation model that makes the processing time/precision more flexible. The proposed operation method consists of the combination of two techniques: obtaining the result in a successive processing way and using precalculated data in look-up tables.

- 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.
- The use precalculated data memories (LUT —*Look Up Table*) 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 proposed [3], [4]. In this approach, we use look-up table to process directly an elemental operation. Figure 1.

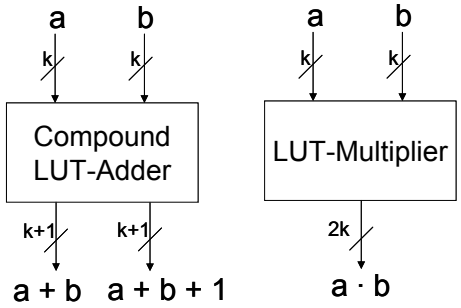

Figure 1: Table Look-Up based operators

The 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 that promote segmentation and a high level of parallelism. These construction possibilities provide robustness and flexibility to the operations [5].

Their disadvantage is the high area complexity for overlarge operands [3], nevertheless, the progressive improvement in performance provided by electronic technology justifies the search for new proposals that would probably have been prohibitive some time ago.

For performance reasons, it is assumed that the LUT memory is implemented in the circuitry of the Arithmetic Unit itself, thus reducing communication costs.

The projected arithmetic unit carry out two operations :

addition and multiplication. We are currently investigating also the adjustment to the flexible calculation of other operations (division, power and square root), that will be presented in following works.

The figure 2 illustrates the unit's design.

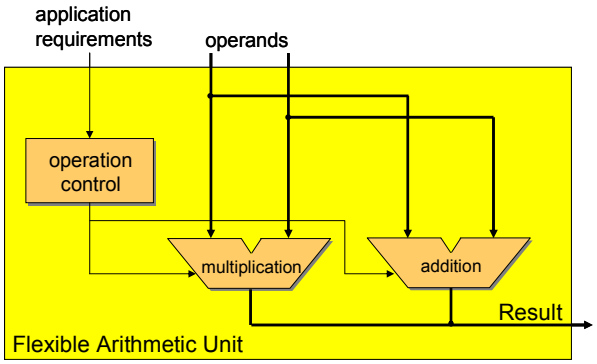

Figure 2: General structure of the arithmetic unit

The arithmetic unit must have an *operation control* that translates the *application requirements* into the number of processed stages.

III. ADDITION OPERATION

A. Addition method

The addition method is based on the carry-select adder scheme [6] and it is made up of the following steps:

1. *Fragmentation of operands into k-size blocks*: It is immediate from the original operands. For numbers of m bits (with m>k), we can divide the number into n blocks of k bits, so that n·k ≥ m.

2. *Addition of the corresponding pairs of blocks*. The partial additions are obtained directly from a look-up table containing the precalculated results: *LUT-Adder*. The processing of the carry is directly made by obtaining the sum and its successor from a *Compound LUT-Adder*. Figure 1. By designing multiple memory access routes, simultaneous access can be gained without the need for several memory chips.

3. Ordered concatenation of the partial additions taking the carry logics into account: The selection of each block is a function of the carry bit of the preceding block selected according to the algorithm carry-select adder [6]. The compound LUT-Adder is used to consider the carry in a direct way, since adding the carry to a block is the same as obtaining its successor in the LUT. The method considers the tree concatenation of the partial additions.

B. Flexible addition

The addition operation based on Look-Up Tables offers predictability of the response times. The basic idea consists of only performing the sum on blocks for which available time exists. Therefore, this design has real-time properties. Thus, depending on the time available, the system will adapt the quality of response. According to the increase in the number of iterations, the error rate will decrease.

The flexible adder design is based on the previous algorithm with the special feature that only part of the blocks obtained from the operands are combined, according to the time availability.

For a tree concatenation scheme, figure 3, the process is carried out from bottom to top, acting on all the blocks of partial solutions concurrently per level. The total number of bits from the result increases exponentially with the number of steps.

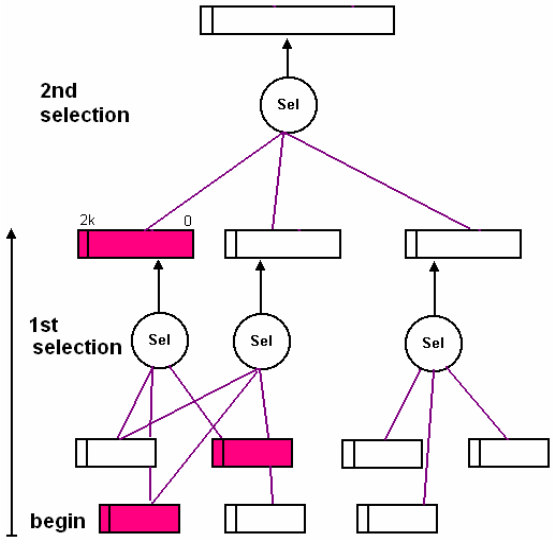

Figure 3: Tree selection

IV. MULTIPLICATION OPERATION

A. Multiplication method

The calculation method is made up of the following steps [7]:

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.

The technique presented is based on LUT access with precalculated products. The method consists of fragmenting the numbers to be multiplied into *k* bit blocks, obtaining the product of each pair of blocks directly from the LUT table to form the whole of partial products. See figure 1. Figure 4 shows partial products for operands divided in 4 parts.

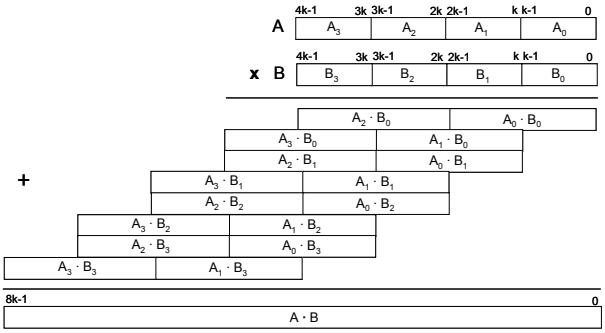

Figure 4. Generation of partial products

The last partial product can be placed to the left of the first one. We can express the number of partial products generated according to the expression $\lceil 2m/k \rceil - 1$.

This is shown in the following table.

TABLE I: QUANTITY OF PARTIAL PRODUCTS GENERATED

m	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

With this technique, a lower number of partial products are obtained compared with other known methods.

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. There are several methods of reduction of partial products [7], [9]. In this work a Wallace-tree reduction is used based on 3:2 counters [10].

3. Final addition: It is implemented by anyone of the known addition methods. [6], [12]

B. 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. The addition of the result of the combination will be made completely. Since the application requirements already are contemplated in the partial product reduction stage.

Thus, for the last partial product generated. It is ready to go on to the result. Figure 5.

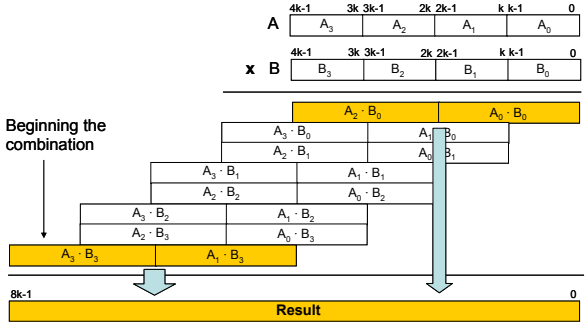

Figure 5. Result produced for the first selection

For other case, partial products are combined and the final addition is carried out. Figure 6.

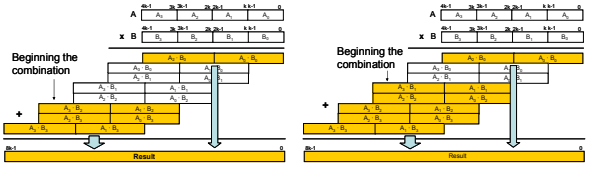

Figure 6. Result produced for the 2nd and3rd selection

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

V. ARCHITECTURE

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

A. Design

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

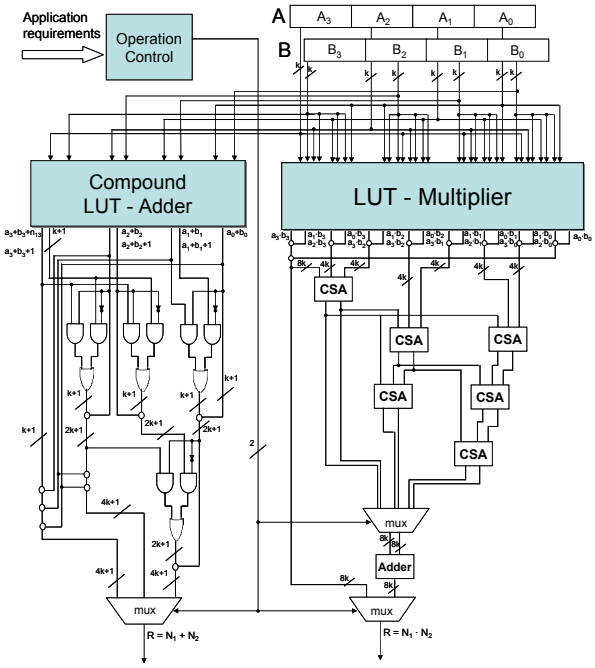

Figure 7: Block diagram of the proposed architecture

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, for example, a coder, multiplexor or table circuit.
- Access to the Compound LUT-Adder and LUT-Multiplier provides the partial results for all the pairs of blocks.
- The selection circuit in addition has a simple design since the effective sum is carried out in the memory with precalculated results: the tree selection combines the partial results.
- The partial product reduction is carried out by means of a tree scheme based on Carry Save Adders (CSA) or 3:2 counters.
- The operation control circuit selects the partial result that best fits the conditions of the problem. Several results with different qualities of degree and delay are extracted from operation circuits.

B. 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:

Addition:

- Max(Op.Ctr.,Comp.LUT-Adder) + mux
- Max(Op.Ctr.,Comp.LUT-Adder)+and+ or + mux
- Max(Op_Ctr.,Comp.LUT-Adder)+ 2and+ 2or + mux

Multiplication:

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

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

VI. Evaluation of the 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.

A. Area estimations

The main contributions to the area of the architecture come from the Tables Look-Up. The areas of the selection and reduction circuits are small when compared to the area of the LUTs, thus, the estimation is focused only on the LUT operators. The model we use for the area estimations is taken from [3], [4] and [11].The unit used is the size of a complex gate τ_a, since the area of the LUTs and other components are easily expressed in this unit.

Data storage imposes severe restrictions on k block size. 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. Tables II to V show the area cost in terms for the most common sizes.

Table II: LUT-Adder size (bits)

k	Size (bits)
4	160 B
6	3.5 KB
8	72 KB

Table III: LUT-Adder size (τ_A)

k	Size (τ_a)
4	$\approx 44\ \tau_a$
6	$\approx 840\ \tau_a$
8	$\approx 11519\ \tau_a$

Table IV: LUT-Multiplier size (bits)

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

Table V: LUT-Multiplier size (τ_A)

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

As shown in the previous tables, the amount of area is much greater than in conventional operators 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.

B. Delay estimations

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

Let τ_t be the delay of a complex gate, such as one full-adder. According to [4], [11] analysis[2] 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.

Addition:

- *Access time to the LUT-Addition in order to obtain the precalculated results*. This time will only be determined by memory access time T_{LUT}.
- *Selection of the blocks that make up the result*. Total selection time is obtained by taking into account that all the selections at one tree level are carried out in parallel and that the total number of tree levels is $\lg_2 n$. Let T_{sel} be the time taken in the selection on one tree level, so the expression for the total selection time is $T_{sel} \cdot \lg_2 n$. A selection step consists of two single gates: (and, or), or one complex gate τ_t, so $T_{sel} = 1\ \tau_t$. For operands of $m = n \cdot k$ bits, the proposed algorithm calculates the addition in: $T_{add} = T_{LUT} + T_{sel} \cdot \lg_2(m/k)$ time units.

We perform a comparison of the proposed architecture with other known adder algorithms.

In the first place, in terms of the expression of the asymptotic temporal complexity, those adders have a growth in the delay equal to the proposed design. The table VI shows the temporal complexity of adder designs [12], where m is the length in bits of the operands.

Table VI: Asymptotic Temporal complexity

Adder	Architecture	Asymptotic Temporal complexity	
CLA	Carry Lookahead Adder standard	$4\lg_2 m$	O(lg m)
PPA-SK	Parallel-Prefix Adder Sklansky	$2\lg_2 m + 4$	O(lg m)
PPA-BK	Parallel-Prefix Adder Brent-Kung	$4\lg_2 m$	O(lg m)
COSA	Conditional-Sum Adder	$2\lg_2 m + 2$	O(lg m)
TLA-4	k=4 LUT adder	$T_{TLU} + 2\lg_2 m - 4$	O(lg (m/4))
TLA-8	k=8 LUT adder	$T_{TLU} + 2\lg_2 m - 6$	O(lg (m/8))

The addition algorithms differ in the constants that modify the general cost expression. In the TLA algorithm, the compound LUT-Adder performance plays a fundamental role in the final calculation time.

In addition, the circuit delays depend on the technology used and on the implementation itself. In order to prove this, the TLA-4 and TLA-8 have been implemented in VHDL and tested on FPGA in comparison with the implementation provided by [13]. The LUT implementation corresponds to the design presented in [14], [15] and has been integrated into the selection circuit.

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

Table VII shows the results obtained after the synthesis and simulation of each adder for some number wordlength, including k wordlength. COSA codification is not available in [13]. We do not implement it to get objectivity in the results.

TABLE VII: DELAYS IN THE CALCULATION OF THE SUM (NS)

Adder	bits (m)				
	8	16	32	64	128
CLA	11,1	25,5	54,0	110,9	224,8
PPA-SK	9,4	20,4	34,5	37,5	48,6
PPA-BK	9,4	13,7	19,8	24,3	32,7
TLA-4	3,3	4,8	7,3	13,1	23,2
TLA-8	3,1	3,7	5,2	7,7	13,5

The previous results demonstrate that the proposed adder 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.

Multiplication:

- *Access time to the LUT-Multiplier 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 [7] design, a reduction step consists of two complex gates τ_t, so $T_{CSA} = 2\ \tau_t$.
- *Final Addition*: We consider anyone addition method of table 6. The cost depends on the operand's length: $T_{Adder} = O(\lg m)$.

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 VIII 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.

TABLE VIII: HOMOGENOUS COMPARISON BETWEEN THE PARTIAL PRODUCT GENERATION METHODS

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 9 shows the results obtained after the synthesis and simulation in FPGA.

TABLE IX: DELAYS OF PARTIAL PRODUCT GENERATION IN HOMOGENOUS COMPARISON (NS)

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

C. Error computation analysis

With the objective of testing the error computation while ignoring the correct concatenation of all the sequence of partial results, an exhaustive set of experiments has been made to prove the method for all cases. The experiments are both in individual operations and in sequences of successive operations.

The profile of the experiments is the following: A compound LUT-Adder with k = 8; number length m = 48 bits; operands are rational numbers at the interval [0, 1).

- *Independent operations* test consists of calculating the average error rate in 10^7 operations of two random rational numbers.
- *Successive operations* 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.

Addition:

The error evolution is shown in figure 8, on sequential and tree schemes of partial results concatenation.

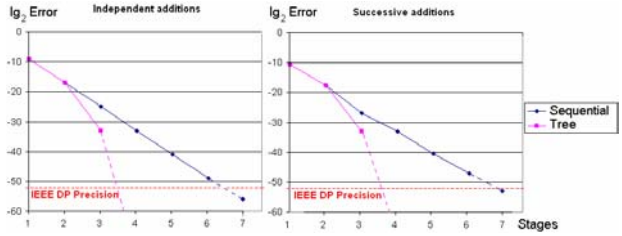

Figure 8: Evolution of the error average in flexible addition

Multiplication:

The error evolution is shown in figure 9.

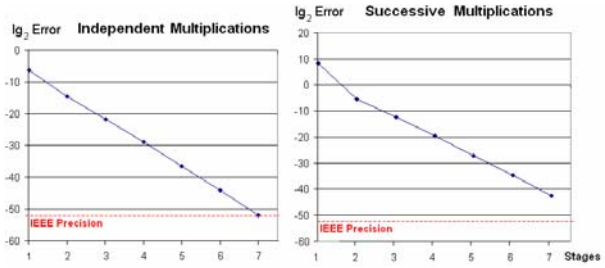

Figure 9: Evolution of the error average in flexible multiplication

VII APPLICATION EXAMPLE

The experiment is located in the *Specialized Processing Architectures* research group of the *University of Alicante, Spain*. 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 [16]. 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.

This section describes a simple example that illustrates the specific application of the proposed architecture. 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 $\vec{r}$.

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

We proposed the following architecture to resolve the expression (1):

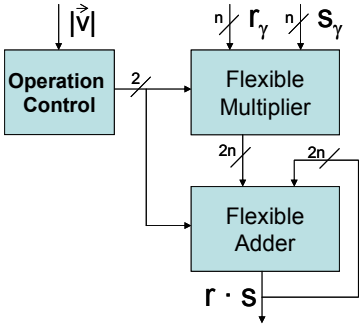

Figure 8. Architecture for the vector scalar product

The multiplication of the components is made sequentially. The first addition is made in a segmentation way at the multiplication time. The operation control module decides the number of stages that will be made in the multiplication and sum operations. The criterion is based on the speed of the moving object. At greater speed, fewer partial results are selected.

We assume: $x_i, y_i, z_i \in [0, 1]$, $n = 32$ bits, $k = 8$, object speed. $\in [0, 150]$.

Table 5 shows the number of multiplication and addition stages, the time-saving that takes place and the computation error. Simulation is made in FPGA for a set of 1000 series of experiments.

TABLE X: EXPERIMENTAL RESULTS (TIME IN NANOSECONDS)

speed	stages	delay	Saving of time	|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%)	$2^{-6,89}$

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 movement decision.

VIII. CONCLUSIONS

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

- The use of precalculated results in stored logic permits the construction of fast operators comparable to existing methods and lays the foundations for the design of high performance architectures. Technological improvements in manufacture or in communication with the selection circuit will tend to reduce T_{LUT} access time and, therefore, total operation time.
- The operation 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. Developed methodology for the operators, due to its features of high performance and obtaining imprecise calculations with limited and decreasing error, can be used in the development of other arithmetic operations with temporal restrictions.
- Finally, the error analysis carried out shows that the algorithm provides limited results, even in cases in which successive calculations are made with imprecise operands.

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

[4] 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, 2000.

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

[6] J. Sklansky. "Conditional-sum addition logic." *IRE Transactions on Electronic Computers*, vol. 9, 1960.

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

[8] Y. Wen-Chang, J. Chein-Wei. „High-Speed Booth Encoded Parallel Multiplier Design". *IEEE Transaction on Computers*, Vol. 49, no 7, pp 692-701. 2000.

[9] 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.

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

[11] J-A.. Piñeiro, J.D. Bruguera, J.M. Muller. "Faithful powering computation using table look-up and a fused accumulation tree". *ARITH'15*, 2001.

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

[13] R, Zimmermann. "VHDL Library of Arithmetic Units", *Forum of Design Languajes*, Lausanne, September 1998.

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

[15] 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.

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