On the Design of Floating Point Units for Interval Arithmetic

– PhD. Project –

PhD. Student: Eng. Alexandru Amăricăi
Scientific Advisor: Prof. Dr. Eng. Mircea Vlăduţiu

September 2007

ABSTRACT

Interval arithmetic has been proven a more reliable alternative to the conventional floating point arithmetic. A great number of mathematical methods that use interval arithmetic with applicability in a wide range of fields, like computer graphics, air traffic control, physics, have been developed. However, these methods are slow and inefficient on modern computers due to the lack of hardware support. Therefore, the main goal of this thesis is to design floating point units suitable for interval arithmetic, in order to exploit the full potential of interval methods. The designed units have to be similar in cost and performance with respect to conventional floating point units. Furthermore, because interval arithmetic is not meant to be a replacement of the conventional floating point arithmetic, but an extension of it, the proposed units have also to be suitable for conventional floating point arithmetic. Other approaches consisted on developing hardware support for interval arithmetic by incorporating existing conventional floating point units. My approach relies on the design of these units from almost zero, by taking into account algorithms for interval operations and the particularities for each floating point operation (normalization, rounding algorithms, exponent computation, etc). A very important aspect of the research is the analysis of the proposed circuits in terms of cost and performance. The cost will be estimated in terms of gate count. The performance will be estimated both in the latency of the circuits (measured in logic levels) and by estimating the performance of different interval methods on the proposed units. Thus, a benchmark type analysis is required. Because in interval arithmetic there is no known set of benchmarks programs (similar to SPEC-FPU), a benchmark proposal will also be made in this thesis.

Table of Contents:

1. Introduction

In the last decades the computational power of computers has increased almost exponential. Nowadays, computers that perform billions of arithmetic operations are common. However, the precision of the arithmetic is at a standstill. As exemplified in [21], in the 1960's the IBM S360/91 floating point format had a 64 bits representation, with 7 bits for exponent and 56 for mantissa; nowadays, the almost all computers use the IEEE 754/1985 double precision representation, with 11 bits for exponents and 52 bits for mantissa.

Floating point arithmetic is full of errors. The reasons for these types of errors are multiple: the impossibility of representing all the real numbers using a floating point format (simple numbers like 1/3 or $\sqrt{2}$), truncation errors, rounding errors etc. These errors seem very small. However, due to the great number of arithmetic operations, an accumulation of arithmetic errors can happen. This may lead to disastrous consequences, like the Dahran incident during the First Gulf War [11].

Therefore, in some fields of application, it is very important for monitoring and controlling the errors which occur in the floating point arithmetic. One way for monitoring these types of errors is by using interval arithmetic. Interval arithmetic does not increase the precision of floating point arithmetic, but it provides a measure of the accuracy of the arithmetic computations [21].

Interval arithmetic does not deal with a single floating point number, which is an approximation (less or greater accurate) of the desired real number, but deals with an interval, defined by two floating point numbers (the upper bound of the interval and the lower bound of the interval). The interval has the property that it surely contains the desired real number. The width of the interval is of measure of the accuracy for the arithmetic computations, or better said is a measure of the lack of accuracy [11]. Thus, interval arithmetic and interval mathematics provides methods which offer guaranties over the obtained results. A wide range of mathematical methods have been developed over the last four decades, like the Newton interval method for non linear equations [15], methods for systems of equations [5][27][28][29] with applicability in physics [11], computer graphics [20], air traffic control, robot control [5] etc.

The potential of interval mathematics cannot be fully exploited because interval methods are inefficient on modern computers. Interval arithmetic is much slower than conventional floating point arithmetic due to the reasons: on one hand, interval arithmetic comprises of at least two floating point operations; on the other the only rounding mode which is incorporated in the arithmetic instructions is the rounding towards nearest even, which is not used in interval arithmetic; for operations with rounding towards positive or negative infinity two instructions are needed (one for setting the rounding mode and the other for the operation) [13]. Thus, the main reason for the low performance of interval methods is the lack of hardware support, although hardware designs for different arithmetic operations have been proposed. This lack of support has two reasons, as presented in [11]: the uncertainty in demand and the lack of any kind of standard (similar in some aspects with the IEEE 754).

The main objective of the thesis is to propose new designs and circuits, which are both high performance and cost effective with respect to conventional floating point units, for interval arithmetic. In this way, hardware support for this type of arithmetic can be provided. Because interval arithmetic should be seen as an extension of conventional floating point arithmetic [13], the proposed units have also to be suitable for conventional floating point arithmetic. Thus, the main focus of the proposed thesis is on hardware design of floating point units for interval arithmetic.

This project proposal will present the main reasons why a successful research can be conducted in this engineering domain. This paper is structured on two main chapters. In the second chapter, the scientific background which will be the starting point for my research. This chapter will briefly present the main achievements and the latest trends in both interval and floating point arithmetic. A very detailed presentation will be done in the following PhD. reports, where the proposed solutions will be compared to both interval and floating point units. The third chapter will present the PhD. thesis outline. The structure of the reports and thesis is proposed. Furthermore, the targeted impact of this thesis is presented, in both potential contributions and desired publications. Also, the main activities and the milestones associated with them are depicted.

2. Background

Hardware designs for interval arithmetic have been proposed. However, these designs incorporate existing conventional floating point units, without modifying or adapting their internal structure for the algorithms and particularities of interval arithmetic. My approach in the research is designing the interval arithmetic units by creating new architectures for floating point units which take into account the algorithms and particularities for interval arithmetic. Therefore, in this chapter are presented algorithms and previous designs for interval arithmetic on one hand, and the main architectures and design issues for conventional floating point arithmetic, from where the inspiration is drawn, on the other hand.

2.1 Interval Arithmetic Algorithms and Designs

An interval X is defined by two floating point numbers, which constitute the bounds of the given interval: $X = [X_{lo}; X_{hi}]$ [13][15]. The four basic arithmetic operations between two intervals X, Y are given below [13][15][22]:

- addition: $[X_{lo}; X_{hi}] + [Y_{lo}; Y_{hi}] = [X_{lo} + Y_{lo}; X_{hi} + Y_{hi}]$
- subtraction: $[X_{lo}; X_{hi}] - [Y_{lo}; Y_{hi}] = [X_{lo} - Y_{hi}; X_{hi} - Y_{lo}]$
- multiplication: $[X_{lo}; X_{hi}] * [Y_{lo}; Y_{hi}] = \left[\max\left(\prod XY\right); \min\left(\prod XY\right)\right]$

  where $\prod XY$ is represented by the four products

  $X_{lo} * Y_{lo}; X_{hi} * Y_{lo}; X_{lo} * Y_{hi}; X_{hi} * Y_{hi}$
- division: $[X_{lo}; X_{hi}] / [Y_{lo}; Y_{hi}] = [X_{lo}; X_{hi}] * [1/Y_{hi}; 1/Y_{lo}]$

  undefined for $0 \in [Y_{lo}; Y_{hi}]$

Also, the set operations, like the hull, the intersection and the inclusion are used in interval arithmetic.

A very important aspect for interval arithmetic is the rounding mode used; thus, the rounding modes which are used are rounding towards negative infinity for the lower bound of the result (for example, in case of addition $X_{lo} + Y_{lo}$ will be rounded towards negative infinity) and rounding towards positive infinity for the upper bound of the result (for example, in case of addition $X_{hi} + Y_{hi}$ will be rounded towards positive infinity).

Several approaches have been proposed in order to provide hardware support for interval arithmetic. The ones presented in [21][22] are dedicated to variable precision floating point format, while the ones presented in [1][13][15][26] can be used for IEEE floating point format numbers. In [25] issues regarding special numbers and exception handling in interval arithmetic are presented.

Interval addition, subtraction and division require two floating point operations. The solutions proposed for these three interval operations ([13][15]) rely either on a single hardware unit (a floating point adder for addition/subtraction or a floating point divider for division) which implements both needed rounding modes, either on two floating point units (one for the lower bound of the result, while the other for the upper bound of the result) – as depicted in Fig. 2.1. In the first case, the performance of an interval operation is equal to the performance of two conventional floating point operations, while the cost is the same with respect to a conventional floating point unit. In the second case, the performance of an interval operation is the same as a conventional floating point operation, while the cost is almost double.

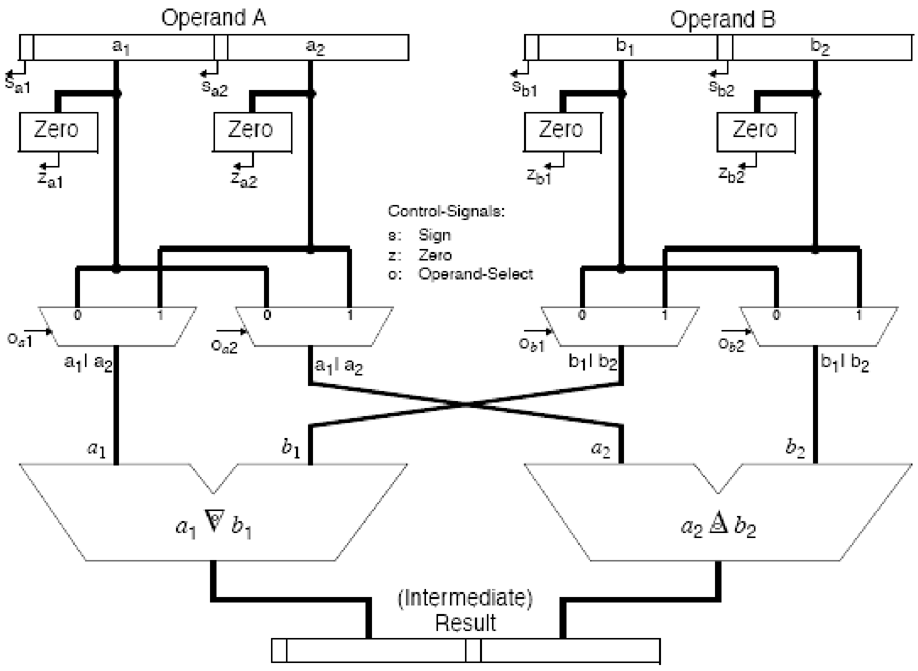


Fig. 2.1 – Hardware support for interval operations with two floating point units as depicted in [13]

Interval multiplication is the most complex from all four basic interval arithmetic operations. It requires four floating point multiplications and six floating point comparisons, which means a total of ten floating point operations (ten operations if we do not count rounding). As presented in [18], in conventional floating point arithmetic the multiplications count almost 40% from all arithmetic instructions. Therefore, it is expected that in interval arithmetic, the multiplication to be a very important and frequent operation. Thus, the latency of the interval multiplication (which consists of ten floating point operations) is unacceptable. Different approaches in order to improve the performance have been proposed. One approach used in [13][15][26] is based on the reduction of the number of operation needed based on the signs of the two operands. Thus, depending on the signs of the four floating point numbers, nine cases for interval multiplication are obtained (Table 2.1). In eight cases, only two floating point multiplications are needed; while in the ninth case (when the two operands contain zero) four multiplications and two comparisons are needed. (Regarding the interval division, this operation is undefined when the second interval contains zero; thus, using the sign examination approach, only six cases are obtained, in all of them only two operations being required.)

Table 2.1 – Nine cases for interval multiplication [15]

Case	$[X_{lo}, X_{hi}]$	$[Y_{lo}, Y_{hi}]$	**Result**
1	$X_{lo} > 0$	$Y_{lo} > 0$	$[X_{lo} * Y_{lo}, X_{hi} * Y_{hi}]$
2	$X_{lo} > 0$	$Y_{hi} < 0$	$[X_{hi} * Y_{lo}, X_{lo} * Y_{hi}]$
3	$X_{lo} > 0$	$Y_{lo} < 0 < Y_{hi}$	$[X_{hi} * Y_{lo}, X_{hi} * Y_{hi}]$
4	$X_{hi} < 0$	$Y_{lo} > 0$	$[X_{lo} * Y_{hi}, X_{hi} * Y_{lo}]$
5	$X_{hi} < 0$	$Y_{hi} < 0$	$[X_{hi} * Y_{hi}, X_{lo} * Y_{lo}]$
6	$X_{hi} < 0$	$Y_{lo} < 0 < Y_{hi}$	$[X_{lo} * Y_{hi}, X_{lo} * Y_{lo}]$
7	$X_{lo} < 0 < X_{hi}$	$Y_{lo} > 0$	$[X_{lo} * Y_{hi}, X_{hi} * Y_{hi}]$
8	$X_{lo} < 0 < X_{hi}$	$Y_{hi} < 0$	$[X_{hi} * Y_{lo}, X_{lo} * Y_{lo}]$
9	$X_{lo} < 0 < X_{hi}$	$Y_{lo} < 0 < Y_{hi}$	$[\min(X_{lo} * Y_{hi}; X_{hi} * Y_{lo}), \max(X_{lo} * Y_{lo}; X_{hi} * Y_{hi})]$

In the sign examining approach, a significant improvement in the average performance is obtained. Also, an improvement in the worst case performance is also obtained. However, this approach is very difficult to implement on pipeline structures, mainly due to the different number of operations required in each case [15]. The hardware designs based on sign examination makes use of one or two multipliers and one or two comparators. As it can be observed in

Table 2.1 an interval multiplication cannot be performed without a floating point comparator.

Another approach for interval multiplication is presented in [15] and uses and algorithm, which although at a first glance looks difficult, is suitable for pipelined structures. This type of algorithm makes use of one or two floating point multipliers (the one multiplier variant is presented in Fig. 2.2) and of two comparators.

$p = X_{lo} * Y_{lo}$		
$q = X_{lo} * Y_{hi}$		
$r = X_{hi} * Y_{lo}$	$m = \min(p, q)$	$M = \max(p, q)$
$t = X_{hi} * Y_{hi}$	$m = \min(m, r)$	$M = \max(M, r)$
	$m = \min(m, t)$	$M = \max(M, t)$
Rounding	$Z_{lo} = RNI(m)$	$Z_{hi} = RPI(M)$

Fig 2.2 – Interval multiplication algorithm presented in [15]

Very important for interval arithmetic are the set operations, like the hull, inclusion or intersection (which is frequently used in Newton's interval method [15]). Therefore, hardware designs for such operations have been proposed, such as the ones in [1][13]. These designs are based on the floating point comparator and implement the algorithms for interval set operations, like the ones presented in Fig. 2.3.

$Z_{lo} = \max(X_{lo}, Y_{lo})$	$Z_{lo} = \min(X_{lo}, Y_{lo})$
$Z_{hi} = \min(X_{hi}, Y_{hi})$	$Z_{hi} = \max(X_{hi}, Y_{hi})$
If $z_{lo} < z_{hi}$ then	R= $[z_{lo}, z_{hi}]$
R= $[z_{lo}, z_{hi}]$	
else R= $\varnothing$	
a)	b)

Fig 2.3 – Interval intersection (a) and hull (b) algorithms as presented in [1]

As we can see, all these designs make use of floating point units without modifying their internal structure for the specificities and algorithms of interval arithmetic. My approach for this PhD. is to modify the internal structure of the floating point units in order to obtain the best performance and cost for interval arithmetic operations. Thus, an overview of the main floating point units is absolutely necessary.

2.2 Floating Point Arithmetic

2.2.1. IEEE 754 standard

The IEEE 754 standard was developed in the mid 1980's and it was a successful attempt to provide a unitary floating point number system to be used in all computers. The main idea behind the developing of the IEEE 754 standard was that the same program, which contains floating point operations, with the same inputs, that is working on two different computers, to produce the same result [9]. The standard presents a number format to be used for the floating point numbers, specifies operations that have to be done, rounding modes, special values and exceptions [9].

A floating point number using the format specified by IEEE 754 standard is composed from the sign bit, exponent bits and mantissa bits [30]. The value is computed using the formula:

$$N = (-1)^s * 2^{e-bias} * 1.m$$

where *s* represents the sign bit, *e* the exponents bits and *m* the mantissa bits, while the *bias* represents the biased number of the exponent (the exponent is represented in a biased form). The bit 1 from the mantissa representation (1.*m*) is called the hidden bit [30].

The IEEE 754 standard specifies four formats for floating point numbers [30]: single precision, double precision, single extended and double extended. The single difference between these formats is represented by the number of bits used to represent the numbers. In Table 2.2 the parameters for the four formats are given.

The IEEE 754 standard also specifies four rounding modes for the floating point operations [30]: rounding towards nearest even, rounding towards positive infinity, rounding towards negative infinity, rounding towards zero. Although, four rounding mode are adopted by the standard, the only implicit rounding mode adopted both in processor design and programming languages is rounding towards nearest even. For any other rounding mode, an instruction for setting the rounding mode has to be used before any operation [13].

Table 2.2 IEEE 754 format parameters [9]

	Single Precision	Single Extended	Double Precision	Double Extended
Sign bits	1	1	1	1
Exponent Bits	8	11	11	15
Bias	127	1023	1023	16383
Mantissa Bits	23	31	52	63
Format Width	32	43	64	79

Also in IEEE 754 standard special values had been specified. These values describe special situations that can appear during computations and their role is to prevent program halting when is not necessary [9]. The special values are encoded in the IEEE format and they are easy to detect. The special values are: zero (positive and negative), infinity, denormalized and NaN. In table 2.3 are presented al the special values that appear in the IEEE 754 standard.

Table 2.3 Special values specified by the IEEE 754 standard [9]

Exponent	Magnitude	Represents
E=0	M=0	+/- 0
E=0	M<>0	Denormalized number
0<E<=Emax		Floating point number
E=Emax+1	M=0	+/- ∞
E=Emax+1	M<>0	NaN

Along with special values, the IEEE 754 standard also defines the mechanism of exceptions. This mechanism is used in case of exceptional conditions do appear [9]. Five classes of exceptions are defined by the standard: overflow, underflow, division by zero, invalid and inexact [30]. The standard also recommends the use of trap handlers in case one of these exceptions does appear. However, it is not mandatory to implement trap handlers, in this case for each exception a specific value being returned (see Table 2.4).

Table 2.4 – Exceptions in IEEE 754 and their returned values [9]

Exception	Returned Value
Overflow	$+/-\infty$
Underflow	0
Divide by zero	$+/-\infty$
Invalid	NaN
Inexact	Round(result)

2.2.2. Floating Point Addition

Floating point additions/subtractions count more than 55% of all floating point operations [18], thus making the floating point adder the most frequently used floating point unit. In order to add to IEEE 754 floating point numbers, given by sign, exponent and mantissa, several steps must be followed [18]:

1. Exponent difference.
2. Right shifting the significand of the smaller operand with the result of the absolute value of the exponents' difference.
3. Significand addition/subtraction.
4. Result conversion in case that the significand is negative, after being yielded in step 3.
5. Leading zero detection in order to determine the number of left shifting positions needed in normalization step.
6. Normalization of the significand and updating the exponent of the result.
7. Rounding.

This algorithm requires two full length shifters (in step 2 and 6), three large carry propagate adders (in steps 3, 4 and 7) and one leading zero counter (in step 5). Therefore, a very low performance is obtained by implementing this algorithm.

In order to increase the performance of this crucial operation, the double path floating point adder is used. First described in [8], many types of this kind of floating point adder have been developed, like the ones in [10][17][18][23][24]. The double path adder is built on the following assumptions:

1) When the exponent difference is greater than 1, then in case of an effective subtraction the maximum number of leading zeros is one, so in the normalization step only one-position left shift might be requires. Furthermore, there is no need for the leading zero detection. Also, in case of

any type of effective addition (no matter of the exponent difference) there is no possibility of leading zero's appearance. In this case, a large full length right shifter is required in order to align the two mantissas. This case is known as the FAR path.

2) When the exponents' difference is 0 or 1, then only a one position right shift might be needed in step 2. However, in case of effective subtraction, there is the possibility of appearance of a large number of leading zeros. This case is known as the CLOSE path. Also, on this path, instead of counting leading zeros after the subtraction of the two mantissas was performed, a leading zero prediction is performed in parallel with the mantissa addition [2][19]

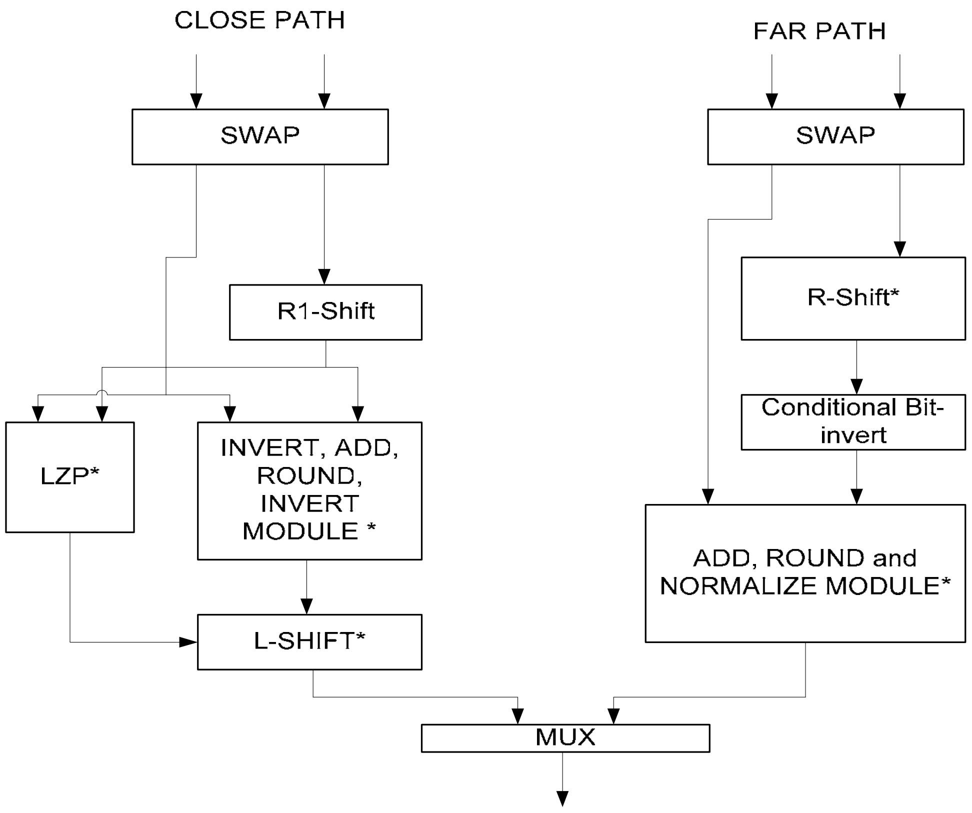


Fig. 2.5 – The double path adder architecture [6]

Furthermore, both in FAR and CLOSE path a compound adder (which has as result the sum of two numbers and the incremented sum – sum+1) is used to add the aligned mantissas [18]. In this way are computed all the possible results which may be obtained after rounding. Thus, the rounding step is reduced to only a simple selection. Furthermore, the mantissas are swapped based on the exponent difference so in case of a subtraction, the result to be always positive [18]. Thus, the result conversion (2's complementing) is not any more needed. The general architecture of the mantissa computational path in the double path adder is depicted in Fig 2.5. The selection criteria for the path selection can vary from different types of double path adder: in [8][18] the exponents' difference is used, while in [10][17][24] the CLOSE path is used only for effective subtraction when the exponents' difference is 0 or 1.

2.2.3. Floating Point Multiplication

Floating point multiplication is one of the most simple floating point operations and consists of an exclusive or between the signs, an exponent addition, and a significands multiplication. The significand multiplication is basically an unsigned integer multiplication. The most appropriate multipliers for the significand multiplication are the tree multipliers [6], due to their high performance. The typical structure of a tree multiplier [6] consists of:

1. A partial product generation circuit which acts like an encoding scheme. This unit implements one of the integer multiplication algorithms, like Robertson, Booth or Modified Booth. Based on the two input numbers, the scheme generates several partial products, depending on the chosen algorithm and the chosen radix.
2. The partial product reduction tree, which can be a Wallace tree or a binary tree, reduces the partial products resulted after the encoding scheme into two final partial products. This unit is usually built from carry-save adders.
3. The final propagate adder, which sums up the final partial products in order to generate the result of the multiplication. In general, if the input number has an *m* bits size, then the final propagate adder has a $2 \cdot m$ bits size.

However, because of operating with IEEE 754 floating point numbers, several challenges appear [7]:

1. The significands are numbers in the [1,2) interval. Consequently, the result is a number in the [1,4) interval. Therefore, a normalization step (a one position right shift of the significand, followed by an increment of the exponent) may be needed.
2. After the mantissas multiplication, a double size mantissa will result. Thus, a rounding step is needed. This rounding step may require a plus one addition to the significand of the result – thus a large carry propagate adder.

Therefore, performance degradation can be observed, mainly due to the rounding step. Thus, several methods for latency reduction in the rounding step were developed, such as the ES algorithm, the YZ algorithm or the QTF algorithm [7].

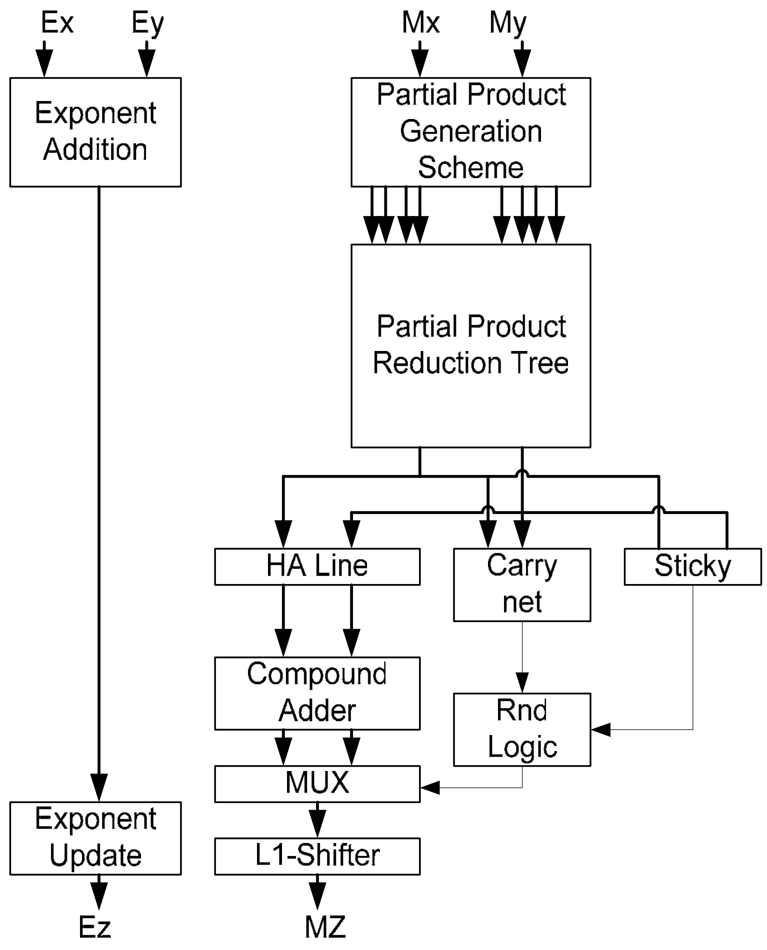


Fig. 2.6 – Overall architecture of the floating point multiplier [6]

All these methods rely on splitting the two final partial products into two halves [6]. The most significant halves of the final partial products are fed to a compound adder in order to compute all the possible results which may occur after rounding. The least significant halves of the two final partial products are used to compute the sticky bits needed for rounding and the carry (that would normally result if the addition of the whole partial products would have taken place). This way, the rounding step would be reduced to a selection between the two possible results, using a simple multiplexer. Thus, a significant increase is obtained because the carry propagate adder in the final stage of the three multiplier is replaced by a half size compound adder and the rounding step is reduced to a selection (using a multiplexer instead of a large carry propagate adder) [6].

2.2.4. Multiply-Add Fused

In a wide range of applications, like the signal processing, matrix multiplication, computer graphics, the most frequent operation is the multiply-add (multiplication followed by addition – *A*+*B***C*) [6]. Therefore, it is very favorable that this operation should be implemented as a single instruction and executed by a specialized hardware unit. There are two main reasons for implementing this operation with a single specialized hardware unit, rather than using a multiplier and an adder [6][12]:

1. The performance is higher.
2. There is only one rounding, rather than two. Therefore, a major reduction of rounding errors can be obtained.

In order to multiply-add three numbers (*A*+*B***C*) several steps have to be followed [3]:

1. Add the exponents of the multiplying numbers (*B* and *C*) and subtract the exponent of A in order to determine the amount of alignment shift for A.
2. "Multiplication" of B and C in order to produce a carry save representation (two final partial products form) – in term of a tree multiplier only the encoding scheme and the reduction tree is used.
3. Bit inversion (in case of an effective subtraction) and alignment of A. This is done in parallel with the multiplication.
4. Addition of A with the two final partial products obtained from B*C. This is done using a carry-save adder line.
5. Final addition using a large carry propagate adder. In parallel a leading zero prediction is done.
6. Normalization and rounding.

The general architecture of a floating point multiply-add fused unit is presented in Fig. 2.7. As presented in Fig. 2.7 a multiply-add fused has components specific to both floating point adders (like the alignment shifter, leading zero predictor and normalization shifter) and floating point multipliers (like the carry-save adders based tree).

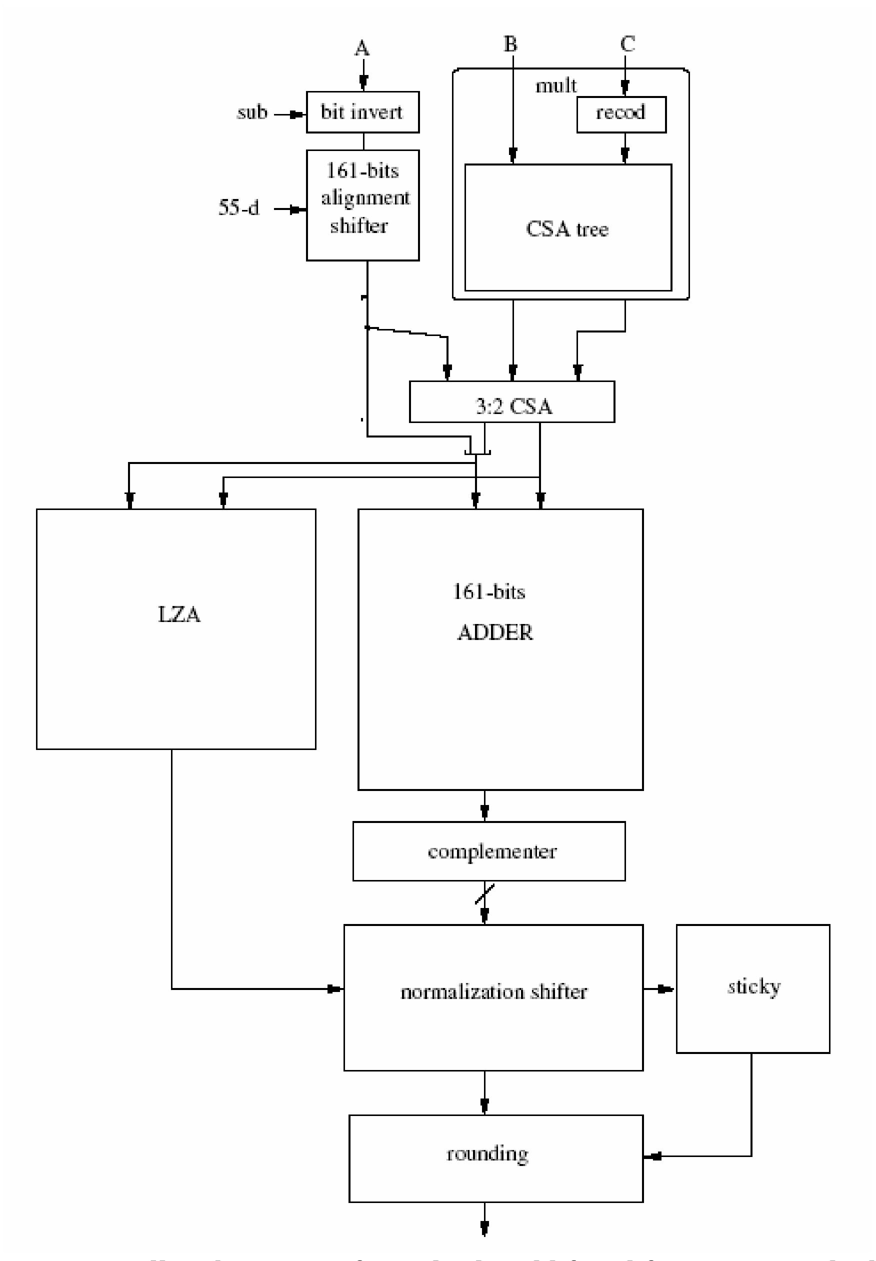


Fig. 2.7 Overall architecture of a multiply-add fused for IEEE 754 double precision format numbers [3]

Different improvements have been made to the overall architecture presented in Fig. 2.7, such as separating the two final operands (the inputs of the final carry propagate) in two parts (with the most significand being inputs for a smaller compound adder, while the least significand used for rounding bits computation – similar to the final stages of a floating point multiplier) [3], or using two computation paths based on the result in step 1 (similar to a double path adder) [4].

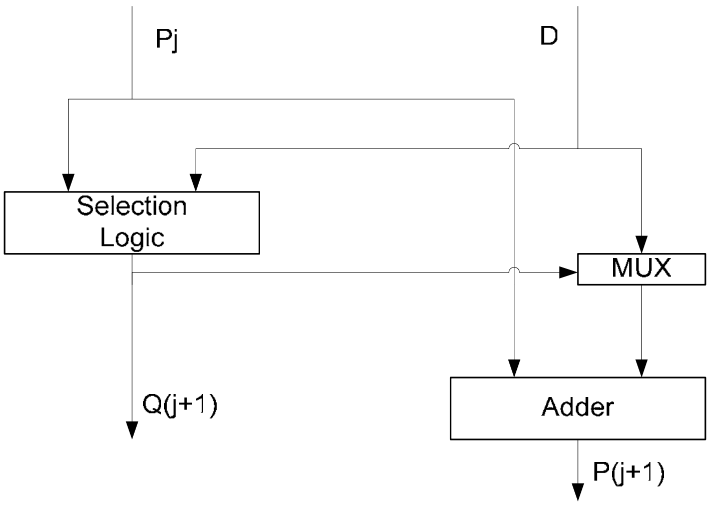
2.2.5 Floating Point Division

Floating point division is apparently one of the most simple floating point operations, being very similar to the floating point multiplication. It consists of an exclusive or between the signs, an exponent subtraction and a mantissa division, which is basically an integer division. However, the division requires a large number of clock cycles, five times or more than multiplication [16].

According to [16][18], the division techniques can be classified into four major classes: digit recurrence, functional iteration, very high radix, table look-up. However, almost all practical implementations use a combination of all these techniques, rather than a single particular class [18].

The digit recurrence techniques include restoring division, non-restoring division and SRT division [16]. The digit recurrence techniques use the addition as the main operation and after each iteration a fixed number of digits are retired and partial remainder is obtained [18]. The most used technique is the SRT because of its high performance. According to [14], the SRT division is characterized by a normalized divisor, a redundant symmetric quotient digit set and a possible redundant representation for remainder. The main choices I case of a SRT division are the radix, the redundant quotient digit set and the representation of the remainder [18]. The basic stage for SRT dividers is presented in Fig. 2.8. In case of the floating point division, a major issue it is represented by rounding: in the case of digit recurrence algorithm the rounding is performed after the computation of several extra quotient digits and examining the final partial remainder.

The functional iteration division uses multiplication, and not addition, as the main operation. The advantage of the functional iteration division is that after each iteration the result converges quadratically [18]. They are four major techniques of functional iteration [16]: polynomial approximation, rational approximation, Newton-Raphson and series expansion (the most used being Goldschmidt algorithm). Functional iteration is very used for floating point division, mainly due to the fact that division is a low frequently used operation and it can be easily performed on fast floating point multipliers (the multiplier is shared between multiplication and division) [18].



Pj – partial remainder after iteration j
P(j+1) – partial remainder after iteration j+1
D – divisor
Q(j+1) – quotient after iteration j+1

Fig. 2.8 The basic SRT stage [18]

Very high radix division applies to dividers which return more than 10 quotient digits after each iteration [18]. Both addition and multiplication are used at each step [16]. The convergence in this case is linear.

Look-up tables are used both in functional iteration and digit recurrence for a better initial approximation. Furthermore, they are used when only a low precision quotient is required [18]. The main advantage of the look-up tables is that they are quite fast, because no arithmetic operations are needed [18]. The look-up tables can be used for direct approximation, linear approximation, interpolation approximation and bipartite approximation [16].

Many of the division techniques can be used also for other more complex operations, like the square root, inverse square root, exponentiation and logarithm [16].

3. Thesis Overview

A PhD. means dissemination of the existing knowledge, and based on this the creation of new knowledge, which can be used and furthermore developed by an entire scientific community. A very rigorous documentation of the state-of-the-art in the scientific domain which the PhD. belongs must be undertaken in order to be able to recognize the disadvantages of the state-of-the-art solutions and then to improve them, thus bringing new ideas and creating new knowledge. In order to do so, the research activity must be undertaken into a scientific environment which can be characterized by tradition, continuity and performance. Such a group is the ACSA (Advanced Computer Science and Architectures) research group, where I belong. Due to the scientific activity of Prof. Mircea Vladutiu, which is my scientific coordinator, and of Mihai Udrescu, Lucian Prodan and Oana Boncalo, this group can offer the premises for a successful PhD. thesis.

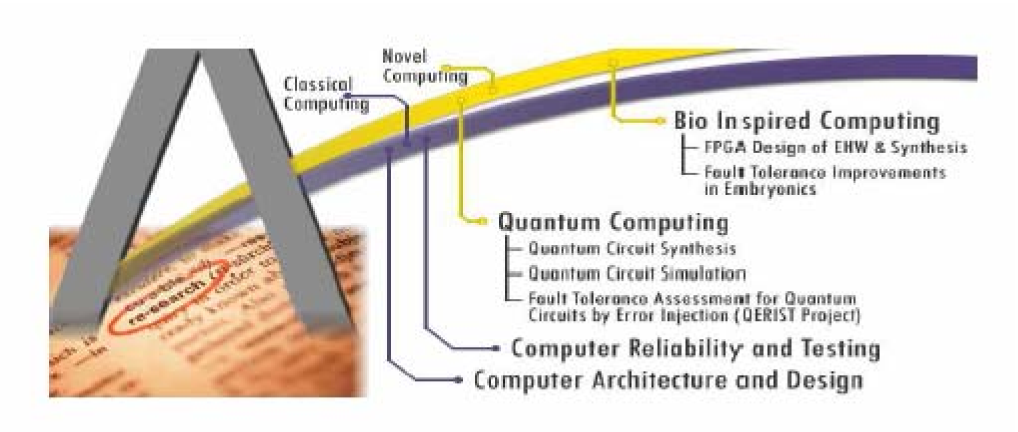

Novel Computing
Classical Computing
Bio Inspired Computing
FPGA Design of EHW & Synthesis
Fault Tolerance Improvements in Embryonics
Quantum Computing
Quantum Circuit Synthesis
Quantum Circuit Simulation
Fault Tolerance Assessment for Quantum Circuits by Error Injection (QERIST Project)
Computer Reliability and Testing
Computer Architecture and Design
research

Fig. 3.1 ACSA laboratory overview

3.1. Proposed Title

The name which I propose for this PhD. is “On the Design of Floating Units for Interval Arithmetic”. This title describes in apprehensive and coherent manner the scientific domain where this PhD. thesis belongs. Furthermore, this title described the methodology for the accomplishing my main objective, which is to design high performance, low cost interval arithmetic units.

3.2. Thesis Objectives

The objectives for this PhD. will follow two main scientific directions. The first direction will focus on the design of units for interval arithmetic, while the second will be focused on the impact of the designed units on specific interval arithmetic applications.

Regarding the first scientific direction, four interval operations will be targeted: addition, multiplication, multiply-add fused and divide-add fused. All four units will be designed in VHDL, at gate level for IEEE simple precision format and at more behavioral level (major modules of the design) IEEE double precision format. Both cost and latency for the proposed units will be estimated using these VHDL models. Furthermore, a gate level model for a simplified IEEE format of only 16 bits will be developed in order to allow exhaustive functional testing.

Regarding the addition unit, a fast design based on the double path floating point adder will be looked for. As in conventional floating point arithmetic, additions are expected to be the most common operations in interval arithmetic. Thus, a fast adder is justified. Furthermore, a significant cost increase must be avoided for the proposed unit.

Regarding interval multiplication, both algorithm and hardware implementation will be looked for. The algorithm will be designed in order to reduce the drawbacks of having large number of operations for interval multiplication. Furthermore, the algorithm must be easily implemented in circuitry.

As in conventional floating point arithmetic, there are some applications (like matrix multiplication) where a multiply-add fused (MAF) may be more advantageous than using a multiplier and an adder. The interval multiply-add fused will be the first of its kind, to the best of our knowledge, there is no such interval unit. Thus, both algorithm and hardware implementation must be developed for this unit.

Regarding the divide-add fused, this hardware unit will be used in order to increase the performance of the Newton's interval method. This method has its main operations a division followed by a subtraction. Thus, a divide-add fused (DAF) unit is justified.

Regarding the second scientific direction, the impact of the proposed units on the specific interval applications will be looked for. The applications will be evaluated and statistic data (like the percentages of each operation) will be extracted. Because the most common programming languages do not provide support for interval arithmetic, an interval arithmetic emulator will be developed.

3.3. Thesis Outline

The proposed outline is given belong. This outline is the today's vision on the future PhD. However, future changes can appear.

1. **Introduction** – in this chapter a overall view of the scientific domain will be made and the opportunity for this thesis will be presented.
2. **An Interval Arithmetic System Perspective** – this chapter will be dedicated to the analysis of the specific interval arithmetic applications; a presentation of these applications will be realized; a presentation of the methodology of analysis will be made; and finally the results will be given, which will present the hardware requirements for these applications.
3. **Interval Addition** – this chapter will be dedicated to the interval addition unit; the chapter will contain the state-of-the-art regarding both floating point addition and interval addition; the proposed unit will be presented and detailed; cost and performance estimates are given; the impact for both interval arithmetic applications and for floating point is estimated; also comparison with the state-of-the-art both in conventional floating point and interval arithmetic is realized.
4. **Interval Multiplication –** this chapter will present the interval multiplication unit; the chapter will contain state-of-the-art regarding floating point multiplication; interval multiplication algorithms are examined and detailed; the proposed interval multiplication algorithm will be ; its hardware implementation will be detailed; cost and performance estimates are given; also comparison with the state-of-the-art both in conventional floating point and interval arithmetic is realized.
5. **Interval Multiply-Add Fused** – this chapter is dedicated to the interval multiply-add fused unit; the state-of-the-art of the floating point multiply-add fused will be realized; interval multiply-

add fused algorithms will be proposed; hardware implementation of the proposed algorithms are given; cost and performance estimates are given; comparisons with floating point multiply-add fused units will be made; the impact on specific interval arithmetic applications will be performed.

6. **Interval Divide-Add Fused** – this chapter will present the interval divide-add fused unit; a detailed presentation of the floating point SRT division will be made; a floating point divide-add fused will be proposed; an interval divide-add fused algorithm and its hardware implementations will be given; cost and performance estimates are realized; the impact on specific interval arithmetic application will be analyzed.
7. **Conclusion** – this chapter will present the concluding remarks of the PhD. thesis; the major contributions of this thesis are summarized; furthermore the impact of this thesis is given.

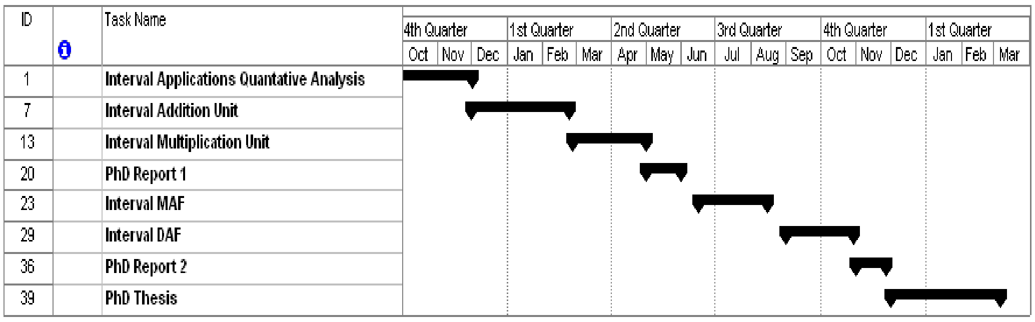
Two PhD. reports will be presented. The contents of the PhD. reports will be the following:

- PhD. Report 1 – this report will be dedicated to interval addition and multiplication
- PhD. Report 2 – this report will be dedicated to interval multiply-add fused and divide-add fused

3.4. Realization

3.4.1. Activity Planning

In order to achieve the major thesis objectives, on the two main scientific directions, and to follow the proposed thesis outline, a very detailed and carefully planning of the research and development activities has to be followed. The major tasks of the research process follow the major chapters of the proposed PhD. thesis layout. Furthermore, the two PhD. reports and the completion of the thesis are also included in the major tasks of the scientific process. Fig 3.2 presents the timeline and an estimated duration for the accomplishment of the major tasks in the research process.

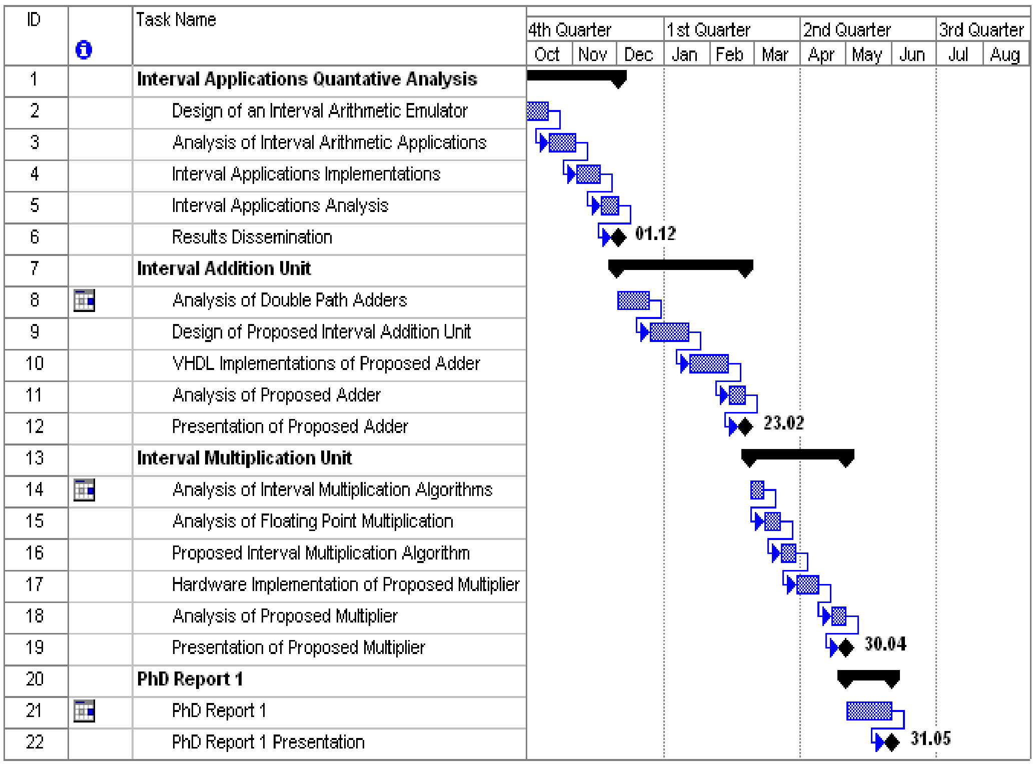


ID		Task Name
1		Interval Applications Quantative Analysis
7		Interval Addition Unit
13		Interval Multiplication Unit
20		PhD Report 1
23		Interval MAF
29		Interval DAF
36		PhD Report 2
39		PhD Thesis

Fig. 3.2 Major activities planning

In order to achieve each major research and development activity, smaller activities have to be accomplished. These smaller activities are specific to every major task.

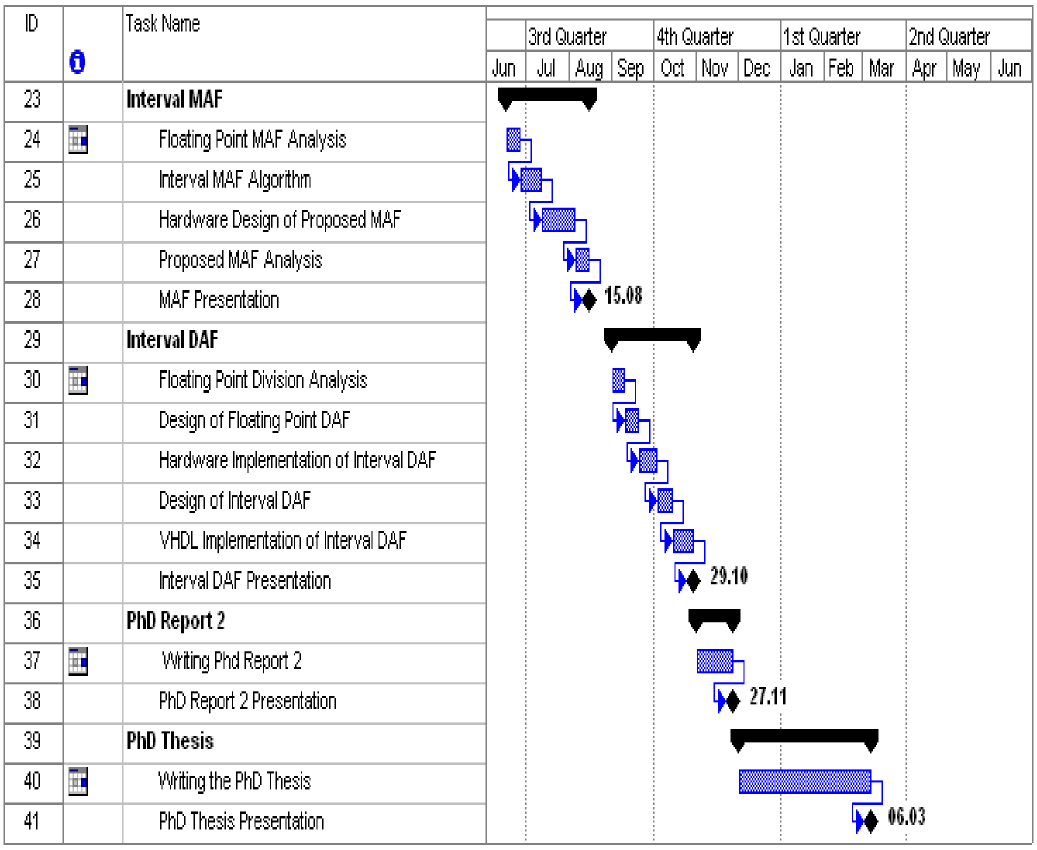
In Fig. 3.3 are depicted the activities which will result in the completion of the PhD. report No. 1. These activities are planned for the academic year 2007-2008. The estimated timeline and milestones for each of major activities are presented in the below graphic.



ID		Task Name
1		**Interval Applications Quantative Analysis**
2		Design of an Interval Arithmetic Emulator
3		Analysis of Interval Arithmetic Applications
4		Interval Applications Implementations
5		Interval Applications Analysis
6		Results Dissemination
7		**Interval Addition Unit**
8	▦	Analysis of Double Path Adders
9		Design of Proposed Interval Addition Unit
10		VHDL Implementations of Proposed Adder
11		Analysis of Proposed Adder
12		Presentation of Proposed Adder
13		**Interval Multiplication Unit**
14	▦	Analysis of Interval Multiplication Algorithms
15		Analysis of Floating Point Multiplication
16		Proposed Interval Multiplication Algorithm
17		Hardware Implementation of Proposed Multiplier
18		Analysis of Proposed Multiplier
19		Presentation of Proposed Multiplier
20		**PhD Report 1**
21	▦	PhD Report 1
22		PhD Report 1 Presentation

Fig 3.3 Planned activities for the 2007-2008 academic year

For the second academic year (2008-2009), the finalization of the PhD. report No. 1 and of the PhD. thesis are the main goals. Also, the design of the interval MAF and interval DAF are estimated to take place in this period. In Fig. 3.4 are depicted the activities which must be undertaken in order to achieve the major tasks of this academic year.



ID	ℹ	Task Name
23		**Interval MAF**
24	▦	Floating Point MAF Analysis
25		Interval MAF Algorithm
26		Hardware Design of Proposed MAF
27		Proposed MAF Analysis
28		MAF Presentation
29		**Interval DAF**
30	▦	Floating Point Division Analysis
31		Design of Floating Point DAF
32		Hardware Implementation of Interval DAF
33		Design of Interval DAF
34		VHDL Implementation of Interval DAF
35		Interval DAF Presentation
36		**PhD Report 2**
37	▦	Writing Phd Report 2
38		PhD Report 2 Presentation
39		**PhD Thesis**
40	▦	Writing the PhD Thesis
41		PhD Thesis Presentation

Fig 3.4 Planned activities for the 2007-2008 academic year

Thus, by a accomplishing each of the major tasks as planned and by sustaining all the activities planned, the PhD. thesis can be defended by the second quarter of 2009, following the directions dictated by the University’s Doctoral Program.

3.4.2. System Design

The main scientific direction of this PhD. will be the design of the interval arithmetic unit for addition, multiplication, multiply-add fused and divide-add fused. These units must be evaluated in terms of both performance and cost and must be compared with both existing interval arithmetic units and conventional floating point units.

In order to achieve the above presented tasks, the proposed units will be designed in VHDL. VHDL will be used because it provides a technology independent hardware description language. Thus, my designs will be technology independent, which is a major advantage for me, due to the inaccessibility of any semiconductor chip processing technology.

For each unit, three VHDL models will be designed. A first model will be made at gate level for a simplified 16 bits IEEE format. The role of this model will be for functional testing, because at 16 bits exhaustive testing is possible.

A second VHDL model will be made at gate level for IEEE simple precision. The role of this model will be to estimate the cost of the design. Furthermore, VHDL models for other existing units will be made in order to achieve an efficient and relevant cost comparison.

A third model will be made for IEEE double precision numbers. This model will be made at a more behavioral level, where the major subcomponents will be described at behavioral level. All the details, and especially the delays, for these subcomponents will be taken from the existing literature. Thus, a relevant performance comparison can be achieved.

3.4.3. Dissemination

An important issue for this PhD. is constituted by the results dissemination. The dissemination will follow two major directions:

- The first direction is focused on the dissemination needed for accomplishing the PhD. thesis and is based around the two PhD. report and the PhD. thesis. In this way, all the scientific results of my research will be evaluated and criticized by the ACSA research group and by the Computer Science and Engineering Department, where I belong.
- The second direction is focused on publication in major conference proceedings and journals. In this way, my work during this PhD. will be evaluated by the entire scientific community. The main topic of the conferences where I attend to publish will be computer arithmetic, in particular, and digital design, in general. Some conferences which I

attend to publish are: IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, Euromicro Digital System Design, IEEE Circuits and Systems Conference, IEEE Symposium on Computer Arithmetic.

3.5. Potential Contributions

This thesis main goal is to offer a new perspective in the design process of interval arithmetic units, offering viable and efficient solutions in order to provide hardware support for interval arithmetic. Thus, significant contributions will be made for each designed unit. The potential contributions can be classified in to classes:

- Better solutions for existing interval arithmetic units – this can apply to the interval addition and multiplication units;
- New interval arithmetic units – this principal can apply to the multiply-add fused and divide-add fused; to the best of our knowledge no such units have been proposed.

Another major potential contribution may be given by the second scientific direction, being the first attempt to analyze interval arithmetic application in order to design specific circuitry dedicated to them.

4. Conclusions

As it was stated in the beginning of this report, this PhD. comes to fill the gap of dedicated hardware support for interval arithmetic. A new approach for this major objective is proposed: to design the hardware units from almost zero, by taking into account all the specifications of interval arithmetic. This way efficient hardware unit both in performance and cost can be obtained. Thus, a better exploitation of the more reliable interval applications is possible.

The PhD. is like a journey. It has a start point, which is represented by the state-of-the-art of the scientific domains where the PhD. belongs. In this case, the scientific domain is represented by the floating point and interval arithmetic. In this PhD. report Chapter 2 is dedicated to the state-of-the-art in both interval and conventional floating point arithmetic. Algorithms and hardware designs for interval arithmetic units are presented. An overview of the IEEE 754 standard for binary floating point arithmetic, which will be used for all the proposed designs, is given. Principles in hardware designs for floating point addition, multiplication, multiply-add fused and division are detailed.

The journey has a destination. This destination is represented by the final PhD. thesis. An outline for this PhD. is given. This outline points out the major objectives and the potential contributions. Furthermore, the main objectives and the contributions are presented in this PhD. report.

Furthermore, the means for reaching the destination are presented in detail in this PhD. report. The main milestones and the activities associated with them are given. A detailed planning for these milestones and activities are depicted. Also, the methodology for design and analysis of the proposed units is presented. And last, but not least, the dissemination plans are given.

Thus, this report constitutes the first step for the proposed PhD. thesis. The future work will follow the guidelines presented in this report.

References:

[1] A. Akkas -A Combined Interval and Floating Point Comparator/Selector – Proceedings of the 13th IEEE Conference of Application Specific Architectures and Processors (ASAP), 2002
[2] J. Bruguera, T. Lang – Leading One Prediction with Concurrent Position Corection – IEEE Trans. on Computers, Vol. 48, No. 10, 1999
[3]J. Bruguera, T. Lang – Floating Point Fused Multiply-Add with Reduced Latency- Proceedings of the IEEE International Conference on Computer Design (ICCD), 2002
[4] J. Bruguera, T. Lang – Floating Point Fused Multiply-Add: Reduced Latency for Floating Point Addition – Proceedings of the 17th IEEE Conference on Computer Arithmetic (ARITH-17), 2005
[5] COPRIN project, -http://www-sop.inria.fr/coprin/developpements/ main.html -, INRIA, France
[6] M. Ercegovac, T. Lang – Digital Arithmetic – Morgan-Kaufmann Publishers, 2006
[7] G. Even, P.M. Seidel – A Comparisson of Three Rounding Algorithms for IEEE Floating Point Multiplication – IEEE Trans on Computers, Vol. 49, No. 7, 2000
[8] P. M. Farmwald – On the Design of High Performance Digital Arithmetic Circuits – PhD. Thesis, Stanford University, 1981
[9] D. Goldberg – What Every Computer Scientist Should Know About Floating Point Arithmetic – ACM Computing Surveys, Vol 32, No 1, 1991
[10] V. Gorshtein, A. Grushin, S. Shevtsov - Floating point addition methods and apparatus – US Patent No. 5808926, Sun Microsystems, 1998
[11] B. Hayes – A Lucid Interval – American Scientist, Vol. 91, No. 9, 2003
[12] R.M. Jessani, M. Putrino – Comparisson of Single and Dual Pass Multiply-Add Fused Floating Point Units – IEEE Trans. on Computers, Vol. 47, No. 9, 1998
[13] R. Kirchner, U. Kulisch – Hardware Support for Interval Arithmetic – Reliable Computing, Vol. 12, No. 3, 2006
[14] P. Kornerup – Digit Selection for SRT Division and Square Root – IEEE Trans. on Computers, Vol. 54, No. 3, 2005
[15] U. Kulisch, -Advanced Arithmetic for the Digital Computers- Springer-Verlag, Vienna, 2002
[16] A.A. Liddicoat – High Performance Arithmetic for Division and the Elementary Functions - PhD. Thesis, Stanford University, 2002
[17] S.F. Oberman – Floating Point Arithmetic Unit Including an Efficient Close Data Path – US Patent No. 6094668, AMD, 2000
[18] S.F. Oberman – Design Issues in High Performance Floating Point Arithmetic Units – PhD. Thesis, Stanford University, 1996
[19] R. Rogenmoser, L. O'Donnel – Method and Apparatus to Correct Leading One Prediction – US Patent No. 6988115, Broadcom Corporation, 2006
[20] J.F. Sanjuan-Estrada, L.G. Casado, I. Garcia – Reliable Algorithms for Ray Intersection in Computer Graphics Based on Interval Arithmetic – Proceedings on 16th IEEE Brazilian Symposium on Computer Graphics and Image Processing, 2003

[21] M.J. Schulte – A Variable Precision, Interval Arithmetic Processor – PhD. Thesis, University of Texas at Austin, 1996
[22] M.J. Schulte, E. Swartzlander – Hardware Design and Arithmetic Algorithms for Variable-Precision, Interval Arithmetic Coprocessor – Proceedings of 12th IEEE Conference on Computer Arithmetic (ARITH-12), 1995
[23] P.M. Seidel – On the Design of IEEE Compliant Floating Point Units and Their Quantitative Analysis – PhD. Thesis, University of Saalanden, 1999
[24] P.M. Seidel, G. Even – On the Design of Fast IEEE Floating Point Adder – Proceedings of the 15th IEEE Conference on Computer Arithmetic (ARITH-15), 2001
[25] G. Steele - Floating point system with improved support of interval arithmetic – US Patent No. 7069288, Sun Microsystems, 2006
[26] J.E. Stine, M.J. Schulte – A Combined Interval and Floating Point Multiplier – Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI), 1998
[27] G. Walster, E. Hansen – Method and apparatus for solving systems of nonlinear equations using interval arithmetic – US Patent No. 6915321, Sun Microsystems, 2005
[28] G. Walster, E. Hansen – Solving systems of nonlinear equations using interval arithmetic and term consistency– US Patent No. 6859817, Sun Microsystems, 2005
[29] G. Walster, E. Hansen – Termination criteria for the interval version of Newton's method for solving systems of non-linear equations – US Patent No. 6920472, Sun Microsystems, 2005
[30] ANSI/IEEE 754-1985 Standard for Binary Floating Point Arithmetic