VHDL Implementation of Interval Arithmetic Algorithms for Single Precision Floating Point Numbers

Sunita.S.Malaj, S.B.Patil, Bhagappa.R.Umarane

Abstract— This paper proposes a new approach where the design and implementation of single precision (32bit) Interval Arithmetic Adder/subtractor unit is carried using VHDL for computing interval arithmetic operations & functions suited for hardware implementation. The algorithms are coded in VHDL & validated through extensive simulation .This VHDL code is then synthesized by synopsys tool to generate the gate level net list that can be implemented on the FPGA using Xilinx. This paper shows that interval arithmetic can be efficiently implemented in terms of performance and cost.

Index Terms— Algorithm, Adder/Subtractor, Interval Arithmetic, Single Precision IEEE 754 standard, Simulation, Special cases,

1 Introduction

The Interval arithmetic represents number in intervals. Interval arithmetic provides an efficient method for monitoring and controlling errors in numerical calculations and can be used to solve problems that cannot be efficiently solved with floating-point arithmetic. However, existing software packages for interval arithmetic are often too slow for numerically intensive calculations. While conventional floating point arithmetic is provided by fast hardware, interval arithmetic is simulated with software routines based on integer arithmetic. Therefore, the hardware design for interval arithmetic can provide a significant performance improvement over software implementations of interval arithmetic. Physical measurements often result in quantitization (errors and uncertainty, which are difficult to represent using traditional fixed or floating point arithmetic. Interval arithmetic overcomes this difficulty by providing the ability to represent ranges of numbers [22]. For example, if a measurement is known to be greater than or equal to 1.3567 and less than or equal to 1.3568, it can be represented by the interval [1.3567, 1.3568]. Interval arithmetic also provides arithmetic operations and mathematical functions on interval data. When performing interval arithmetic operations and mathematical functions, the resulting intervals are generated, such that they are guaranteed to contain the correct result.. For example,[1.23, 1.24] + [3.45, 3.46] = [4.68, 4.70]This paper presents the design of a interval adder/subtractor. This adder/subtractor requires only slightly more area than a conventional floating point adder/subtractor and provides a significant performance improvement over software implementations of interval addition/subtractor.Numerical errors in floating point (FP) computations can lead to inaccurate results, which often go undetected [l]. Interval arithmetic numbers are actually an ordered pair of real numbers representing the lower & upper bound of the parameter range [14]. Interval arithmetic provides an efficient method for monitoring and controlling these errors by producing two values for each result [2]. These two values correspond to the lower and upper endpoints of an interval which contains the true result. The difference between the upper and lower endpoints defines the width of an interval indicating the accuracy of the result. If either interval endpoint is not representable then the interval is Outward rounded. That is, the upper and lower endpoints are rounded towards plus and minus infinity respectively, so that the final interval is guaranteed to contain the true result [15]. Although naive use of interval arithmetic may result in wide intervals, many interval algorithms have been designed that produce narrow intervals [3].Furthermore, since intend arithmetic provides a lower and upper bound for each result, it can be used to solve problems that cannot be efficiently solved using traditional floating-point arithmetic [3]. This approach offers the performance benefits of dedicated interval hardware with only a marginal increase in area. It also lets interval hardware take advantage of advances in floating-point hardware and VLSI technology, and eliminates the overhead of transferring data between the main processor and an interval coprocessor. This paper presents the study and design of a combined interval adder/subtractor that is constructed by adding a small amount of hardware to a conventional floating-point adder/subtractor [12, 14]. This approach offers the benefit of having a functional unit that performs interval and floating-point addition/subtraction, but requires less additional hardware. This paper is organized as follows. Section 2 presents the important aspects of single precision. Section 3 presents the background information of Interval Arithmetic. Section 4 presents the proposed combined Adder/Subtractor. Section 5 gives results of proposed interval adder/subtractor. Finally, Section 6 presents conclusions.

2. SINGLE PRECISION IEEE 754 FORMAT

The single precision IEEE 754 format is as follows:

Sign	Exponent	Mantissa(F)
Bit 31	Bit 30-23	Bit 0-22

Certain values are used for special number representation [12]
If e=255 & f ≠ 0, represents a not a number value (Nan), regardless of s.
If e=255 & f=0, represents a infinity (±∞), depending of s.
If e=0 & f = 0 represents a zero .Note that result depends on S.
If e=0 & f≠0, it is a denormal value.
If 0<e<255, it is a normal value.

3. INTERVAL ARITHMETIC

In Interval arithmetic, each quantity is represented by an interval X = [xl, xu_u], the exact result is said to be contained within this interval such that xl≥x≥xu_u .Each interval can be operated in such a way to contain the value of the quantity it represents. The precise definition of an interval is within an interval model which is stated as follows [6]: [xl_l, xu] ={x€ℜ:x_l≥x≥x_u}, where the lower bound of the interval, x_l is in ƒ U {- ∞}, and the upper bound of interval, x_u is in ƒ U {+∞}. For every operation on real numbers, the interval arithmetic model defines a corresponding interval extension. This operation returns some interval, preferably the smallest one. For elementary operations such as addition and subtraction, implementing these interval extensions is generally straight-forward. However, for certain operations, determining the exact minimum and maximum may be very difficult. In such cases, it is acceptable to return any computable interval that contains the theoretical range of the function not necessarily the smallest one. Computations that involve interval arithmetic pose challenging problems for implementation on digital computers. Interval arithmetic was originall due to inexact inputs. Because of its usefulness in monitoring numerical error, interval arithmetic has been applied to several problems including global optimization, function evaluation, differential equations, finding roots of polynomials, and solving systems of linear equations [18]. Interval arithmetic produces two values for each result. The two values correspond to the lower and upper endpoints of an interval, such that the true result is guaranteed to lie on this interval. For example, if the interval result of a computation is X = [xl xu], where *l* and u, are the lower and upper endpoints of the interval, respectively, then the true result xtrue must lie between xl and xu. The width of an interval *X* is defined as *w (X)* = Xu - Xl. Interval arithmetic specifies how to perform arithmetic operations and mathematical functions on intervals. When performing interval arithmetic on computers, each interval endpoint is typically represented using a single floating point number. This is referred to as standard interval arithmetic. Interval Arithmetic takes into consideration the uncertainty of all the parameters, treating them as interval numbers whose ranges contain the uncertainties in those parameters.. The resulting computations, calculated using Interval Arithmetic would carry the uncertainties associated with the data throughout the analysis. Sensitivity analysis is performed using Interval Arithmetic by assigning bounds to some or all the input parameters and observing the effects on the final interval outcome that will contain all possible solutions due to the variations in input parameters [19]. If the endpoints of a computed interval are not represent able, the interval is outward rounded (i.e., the lower endpoint is rounded toward negative infinity and the upper endpoint is rounded toward positive infinity). This outward rounding causes a widening of the resulting interval. Interval Arithmetic [19, 20, 21] originates from the recognition that frequently there is uncertainty associated with the parameters used in a computation. This form of mathematics uses interval "numbers", which are actually an ordered pair of real numbers representing the lower and upper bound of the parameter range. For example, if we know that r is between 3 and 4 hours, the corresponding interval number would be written as r = [3,4] hours.
Interval arithmetic is built upon a basic set of axioms. If we have two interval numbers X= [a, b] and Y=[c, d], with a b and c d then:

X +Y= [a, b] + [c, d] = [a+c, b+d]
X-Y= [a, b] + (-[c, d]) = [a-d, b-c]
X*Y=[min{ac,ad,bc,bd), max {ac,ad,bc,bd)]
X/Y = [a, b]/ [c, d] = [a, b]*[l/d,l/c],O € [c,d]

Due to domain violations and overflow, an implementation may differ from computer to computer. Recent research into the development of supporting interval arithmetic in the Fortran 77 and Fortran 90 [7] compilers has led to the formation of the Interval Arithmetic Specification (IAS) [8]. This document originally based on work by [9], provides a specification for interval data types; interval constants interval arithmetic operations, interval relational, interval specific functions, interval versions of mathematical functions, and I/O.Interval adder/subtractor are more complicated than floating point adder/subtractor, because an interval defines a range of values. For this paper only interval adder/subtractor is proposed. Table 1 shows the operations necessary for interval adder/subtractor.

4. COMBINED INTERVAL ADDER/SUBTRACTOR

The combined interval adder/subtractor handles normalized numbers in the IEEE-754 format. The design for the interval adder/subtractor is based on the design of floating point adder/subtractor**.**

Addition: $Z = X + Y = [xl + yl, xu + yu]$ For e.g.: $[4.2, 4.4] + [3.4, 3.5] = [7.6, 7.9]$
Subtraction: $Z = X - Y = [xl - yu, xu - yl]$ For e.g.: $[4.2, 4.4] - [3.4, 3.5] = [0.7, 1.0]$

Table 1. Operations For Interval Adder/Sub Unit

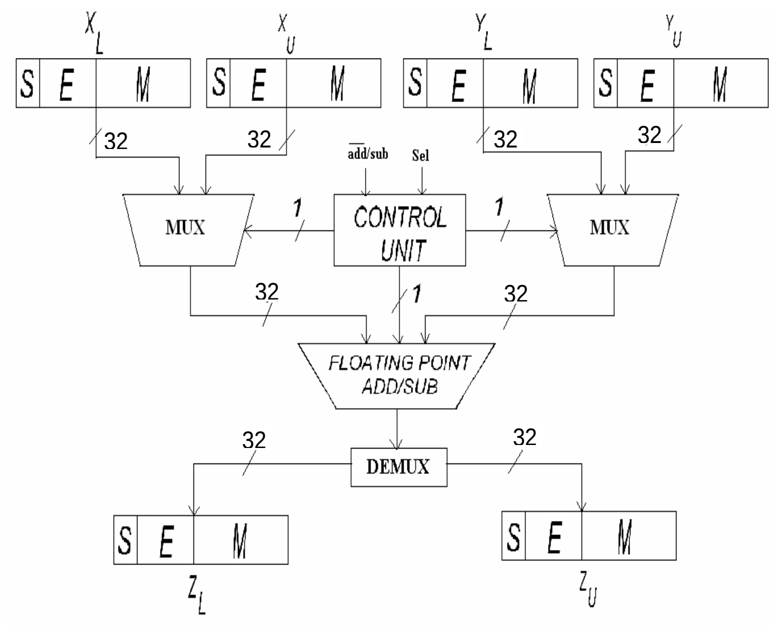


Fig 1.Interval Adder/Subtractor unit

With advances in VLSI technology, it becomes possible for data paths to have more than one functional unit. Therefore, interval hardware which has been previously proposed as a serial implementation can also be implemented in a parallel implementation. The interval adder/subtractor unit is shown in "Figure1. Interval adder/subtractor unit”. The parallel unit does not require any additional logic for selecting the end-points. For interval instructions with two input stored in one register and the upper end point is stored in another register. All input registers, X_L, X_U, Y_L and Y_U, are only used for interval instructions that use two intervals as inputs. In this case, it is assumed that the lower end point of the second interval is in the Y_L register and the upper end point of the second input operand is in the Yu.register, as inputs. In this case, it is assumed that the lower end point of the first interval is in the X_L register and the upper end point of the first input operand is in the X_U register this unit makes use of two multiplexers & one floating point adder/subtractor unit. The four 32 bits single precision floating point numbers are applied as inputs to each of the multiplexers MUX 1 & MUX2 respectively& one select line signal sel as”0” & add/sub signal as “0” .Mux1 selects the 32 bit single precision IEEE 754 floating point number xl and Mux2 selects the 32 bit Single precision IEEE 754 floating point number yl. These two 32 bit single precision IEEE 754 floating point numbers are fed to 32 bit floating point adder/sub unit, which adds these two numbers and stores the lower end point result (xl +yl) in Zl register. Select line (sel) now changes to”1”. The module consisting of MUX1, MUX2 and 32 bit floating point adder are called once again to perform the upper end result. The inputs xu and yu are selected from Mux1 and Mux2 respectively and given to the 32 bit floating point adder unit.The upper end point result (xu + yu) is stored in Zu register. The floating point is designed according to the design of floating point adder
/subtractor [12].The outputs are taken as zl & zu.

The Flow is as follows:
Step [1]: Read the two 32 bit single precision IEEE 754 numbers.
Step [2]: Mantissa, exponent and sign separator: The two, 32 bit single precision IEEE 754 numbers are given as inputs to separate the mantissa designated as ’f ’, the first 23 bits (bit 0 to bit 22), the exponents designated as ’e’, the next 8 bits (bit 23 to bit 30) and the last bit (bit 31) as sign bit .
Step [3]: Mantissa Swapping (including the hidden bit).
The separated exponents are compared using a comparator which compares the exponents for less then, greater then and equal to. If the first exponent is less then the second then the respective mantissas are swapped. If the first exponent is greater then the second or equal the mantissas are not swapped.
Step [4]: Exponent difference
Take exponent a and exponent b. Use twos complement method to obtain the difference of the two exponents.The mantissa whose exponent is smaller is shifted right by the value equal to the difference of the two exponents.
Step [5]: Sign comparator
The signs of both the 32 bit single precision IEEE 754 number are compared using comparator to decide whether the two numbers are to be subtracted or added If both sign bit are different shifted mantissa is complemented (two complement) and added to the first mantissa. If both the sign bits are same then the shifted mantissa is added with the first mantissa. The added result is designated a ’mantissa sum’.
Step [6]: Negative representation of the mantissa sum.
• Both the 32 bit single precision IEEE 754 number has different signs

•The most significant bit of mantissa sum is '1'

• If there is no carry in step 5.

If the mantissa sum is positive retain mantissa sum.

Step [7]: Normalization

Mantissa sum is normalized.

Exponent is obtained

Step [8]: Resultant sign the sign of the complete result is equal to the sign of the larger 32bit single

Precision IEEE 754 number.

Step [9] Concatenation:

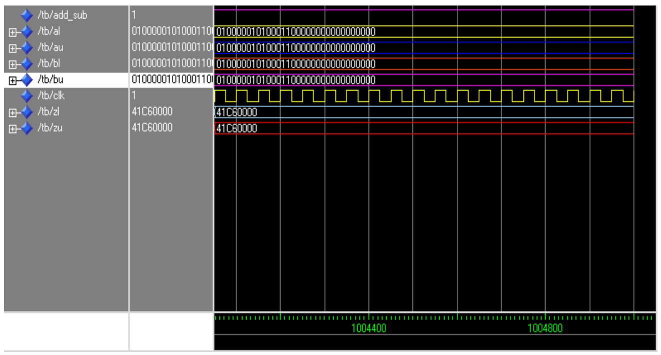
The complete result is obtained by concatenating sign, exponent and mantissa sum.

Addition is a complex operation because depending on the signs of the operands, it may actually be a subtraction. Since floating point numbers are coded as "sign/magnitude" reversing the sign –bit inverses the sign. Consequently the same operator performs as well addition or subtraction for floating point according to two operand's signs [13]. For subtraction add/sub signal is made high in case of interval arithmetic where the subtraction operation of interval numbers is performed.

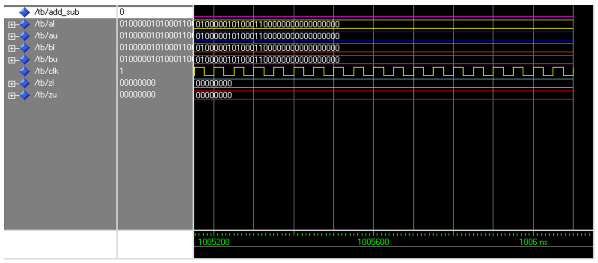
5. RESULTS

5.1. For Interval Arithmetic adder/subtractor

Simulation Results for Addition :



Simulation Results for Subtraction



REFERENCES

[1] D. Goldberg, "What Every Computer Scientist Should Know About Floating-point Arithmetic, "*ACM Computing Surveys,* vol. 23, pp. 5-48, and 1991

[2] R. E. Moore, *Reliability in Computing: The Role of Interval Methods in Scientific Computations,* Academic Press, 1988.

[3] R. B. Kearfott and V. Kreinovich, Eds., *Applications of Interval Computations,* Kluwer Academic Publishers, 1996.

[4] J. E. Stine, *Design issues for accurate and reliable arithmetic,* Ph.D. thesis, Lehigh University, 2000

[5] J. E. Stine and M. J. Schulte, "A two's complement/floating-point comparator in Complementary Pass Logic," in *Conference on Custom Integrated Circuits,* 2004

[6] D. Good and R. London, "Computer Interval Arithmetic: Definition and Proof of Correct Implementation," *Journal of the ACM,* vol. 17, no.4, pp. 603-612, October 1970.

[7] M. J. Schulte, A. Akkas, V. A. Zelov, and J. C. Burley, "Adding interval support to the GNU Fortran compiler," in Abstracts of the International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numeric (SCAN-SB), September 1998.

[8] D. Chiarev and G. W. Walster, "Interval Arithmetic Specification," Tech. Rep.,Sun Microsystems,Inc.,1998,Available at http://www.mscs.mu.edu/- globsol/walsterpapers.html/.

[9] R. B. Kearfott et. al., "A Specific Proposal for Interval Arithmetic in FORTRAN," Tech. Rep., University of Southwestern Louisiana, 1996.

[l0] J. L. Hennessy and D. A. Patterson, *Computer Architecture a Quantitative Approach,* Morgan Kaufmann, 1990.

[11] J. Schulte, K. C. Bickerstaff, and E. E. Swartdander, Jr., "Hardware Interval Multipliers,"

Journal of Theoretical and Applied Informatics, vol. 3, no. 2; pp. 73-90, 1996.

[12] Guillermo Marcus, Patricia Hinojosa, Alfonso Avila and Juan Nolazco-Flores"A Fully synthesizable single precision ,floating point adder/subtractor & multiplier in VHDL for general & educational use" International Caracas Conference on devices circuits and systems" Dominican Republic, Nov.3-5, 2004

[13]Dhiraj sangwan & Mahesh k yadav,″Design & implementation of adder/subtractor & multiplication units for floating point arithmetic″ International journal of Electronics Engineering,2(1),2010,pp.197-203.

[14] Claudio M. Rocco S., Wilhem Klindt″Distribution systems reliability uncertainty evaluation using an interval arithmetic approach″

[15]Rajshekhar shettar, DR R.M.Banakar & Dr.P.S.V.Natraj″Implementation of Interval Arithmetic algorithms on FPGAs″ICCIMA 2007

[16]Distribution systems reliability uncertainty evaluation using an interval arithmetic approach.

[17]R. E. Moore. Interval Analysis. Prentice Hall, 1966.

[18]R. E. Moore. Reliability in Computing: The *Role* of Interval

Methods in Scientific Computations. Academic Press, 1988.
[19]R.Moore: "Methods and Applications of Interval Analysis", SIAM Studies in Applied Mathematics, Philadelphia, 1979
[20] G.Alefeld., J.Herzberger: "Introduction to Interval Computations", Academic Press, New York, 1983
[21]A.Neumaier: "Interval Methods for Systems of Equations", Cambridge University Press, 1990
[22] R. E. Moore. *Interval Analysis.* Prentice Hall, Englewood Cliffs, NJ,

Authors Information
1)Sunita.S.Malaj
P.G Final Sem Student
D.Y.Patil College of Engg & Tech,
Kolhapur
ssmalaj@gmail.com

2)S.B.Patil
Dept of EC,
D.Y.Patil College of Engg & Tech
Kolhapur
S_b_Patil2000@indiatimes.com

3) Bhagappa.R.Umarane
Dept of EC,
K.L.E College of Engg & Tech,
Chikodi
bru1972@rediffmail.com