Design of optimized Interval Arithmetic Multiplier

Rajashekar B.Shettar[1] and Dr.R.M.Banakar[2]

[1]Department of Electronics and Communication
BVB College of Engg and Technology,
Hubli, Karnataka, India
raj@bvb.edu

[2]Department of Electronics and Communication
BVB College of Engg and Technology,
Hubli, Karnataka, India
banakar@bvb.edu

ABSTRACT

Many DSP and Control applications that require the user to know how various numerical errors(uncertainty) affect the result. This uncertainty is eliminated by replacing non-interval values with intervals. Since most DSPs operate in real time environments, fast processors are required to implement interval arithmetic. The goal is to develop a platform in which Interval Arithmetic operations are performed at the same computational speed as present day signal processors. So we have proposed the design and implementation of Interval Arithmetic multiplier, which operates with IEEE 754 numbers. The proposed unit consists of a floating point CSD multiplier, Interval operation selector. This architecture implements an algorithm which is faster than conventional algorithm of Interval multiplier . The cost overhead of the proposed unit is 30% with respect to a conventional floating point multiplier. The performance of proposed architecture is better than that of a conventional CSD floating-point multiplier, as it can perform both interval multiplication and floating-point multiplication as well as Interval comparisons

KEYWORDS

DSP, Floating-point, Interval arithmetic, comparator.

1. INTRODUCTION

Digital Signal Processing has become the choice for many applications related to communications, control, multimedia, etc., because of the high performance it achieves for applications that involve limited instruction set for implementing repetitive linear operations such as addition, multiplication, delay, etc. on a stream of sampled data. Often a DSP has been used as an attached coprocessor or combined with one or more FPGA devices to meet the performance and cost requirements for a particular application. Underlying many of these is key to computing digital filters, Fourier transforms an applications is the need for accurate and reliable results, but

errors due to rounding, uncertainty of the data, quantization noise and catastrophic cancelation in floating point computations can lead to inaccuracies[1,2,3]. Sometimes these inaccuracies can go unnoticed. Since many of the applications in signal processing algorithms, operation are recursive and act on a sequence of data. It implies that the numerical errors can grow unbounded over time. An efficient method for monitoring and controlling these inaccuracies is to replace Floating-point arithmetic with Interval arithmetic. Interval Arithmetic began as a methodology to analyze and control numerical errors in computers. Ramon E. Moore published the first book on the subject entitled "Interval Analysis" [4] in 1966. In the 1990's, the interval arithmetic community saw significant growth and today interval algorithms are being used to solve numerical analysis, global optimization, and several engineering and CAD problems [5,6]. One of the drawbacks in using interval algorithms is their slower computational speed. This is due to the fact that interval arithmetic requires the manipulation of two numbers that define the interval. A complete implementation in software has the disadvantage of extra overhead that results from error and range checking, changing rounding modes and memory management. As a result, interval algorithms end up running slower on current computer architectures, compared to real arithmetic their counterparts [7,8,9]. To overcome the overhead due to rounding, a hardware solution is required that will simultaneously compute the rounding for each interval. This can be achieved by using Interval arithmetic units that are specially designed to manipulate interval numbers.

The paper presents Interval Multiplier unit that will provide the basis for building digital signal processing systems that gives reliable results efficiently. Section 2 will provide background information on Interval arithmetic. In section3, we present the basic architecture of the Interval Multiplier unit. Performance analysis of the design in terms of area and delay estimates will be discussed in Section 4, and finally, section 5 provides the conclusion.

2. INTERVAL ARITHMETIC

In the following discussions, intervals are denoted by capital letters and real numbers are denoted by small letters. The lower and upper interval end points of an interval X are denoted as x_l and x_u, respectively. A closed interval $X= [x_l , x_u]$ consists of set of real numbers between and including the two endpoints x_l and x_u (i.e., $X=\{x: x_l < x < x_u\}$). When performing arithmetic operation on computer, one or both of the interval endpoints may be not represent able. In this case, the interval endpoints are computed by outward rounding. Outward rounding requires that the lower endpoint is rounded towards negative infinity (∇), and the upper endpoint is rounded towards positive infinity (Δ). Outward rounding ensures that the resulting interval encloses the true result. The four basic arithmetic operations (i.e. addition, subtraction, multiplication, and division) are defined for intervals. Interval addition and subtraction are defined as: $Z=X+Y=[x_l+y_l,x_u +y_u]$, $Z=X-Y=[x_l-y_u, x_u - y_l]$

3. INTERVAL ARITHMETIC MULTIPLIER

Interval multiplication is defined as follows:

$Z = X*Y = \min(x_l y_l, x_l y_u, x_u y_l, x_u y_{lu})$, $\max(x_l y_l, x_l y_u, x_u y_l, x_u y_{lu})$. Several algorithms have been developed in order to increase the performance of the interval multiplication, by either reducing the number of the required operations, or by increasing the throughput of the multiplier. The first approach consists of examining the signs of the operands. Nine cases for interval multiplication are obtained, as presented in Table 1. For the first eight cases, only two floating point multiplications are needed. However, in the ninth case (when both intervals contain zero) four floating point multiplications followed by four comparisons are required[10]. But, there is no significant difference between this algorithm and performing the multiplication with above expression. The main disadvantage of this type of algorithm is that the number of steps for the multiplication are variable from case to case, which makes it hard to implement with pipelined structures. We propose new algorithm to evaluate the result for Interval Arithmetic multiplication shown in Table 2. Interval multiplication evaluation is done in two stages. During the stage1 evaluation, we obtain the product of two 64 bit double precision IEEE 754 floating point numbers is done. Here we obtain the following result for stage1, i.e. $p = x_l . y_l$, $q = x_l . y_u$, $r = x_u . y_1$, $t = x_u . y_u$ .. The stage2 evaluation is mainly related to finding minimum and maximum values of compared results of p, q, r & t, and storing the final result of multiplication in Z_l and Z_u registers. During stage2, minimum1 and maximum 1 values are obtained by comparing p and q. Also minimum 2 and maximum 2 values is obtained by comparing minimum1 and r. Finally Z_l and Z_u are obtained by comparing minimum2 and t, and maximum2 and t. The final result is stored in Z_l, Z_u registers. With the help of proposed algorithm, we need only three floating-point comparisons and four floating-point multiplications to get result of interval multiplication. But with the conventional interval multiplication, we need to verify all the nine case, which consumes more time and hardware.

Table1. Cases for Interval multipication

Case	Condition	Z	Example
1	$x_l>0, y_l>0$	$[x_l y_{;l}, x_u y_u]$	[l,2].[3,4]=[3.8]
2	$x_1>0, y_u<0$	$[x_u y_1, x_l y_u]$	[1,2].[-4,-3]=[-8,-3]
3	$x_u<0, y_l>0$	$[x_l y_u, x_u y_u]$	[-2,-1].[3,4]=[-8,-3]
4	$x_u<0, y_u<0$	$[x_u y_{u,}, x_l yl]$	[-2,-1].[-4,-3] =[3,8]
5	$x_l<0<x_u, y_l>0$	$[x_l y_l, x_u y_u]$	[-1,2].[3,4]=[-4,8]
6	$x_l<0<x_u$, $y_u<0$	$[x_u y_l, x_l y_l]$	[-1,2][-4,-3]=[-8,4]
7	$x_l>0, y_l<0<y_u$	$[x_l y_l, x_u y_u]$	[11,2].[-4,3]=[-8,6]
8	$x_l<0$, $y_l<0<y_u$	$[x_l y_u, x_l y_l]$	[-2,-1].[-4,3]=[-6,8]
9	$x_l<0<x_u$, $y_l<0<y_u$	[m n ,m x]	[-2,l][-4,3]=[-6,8]

Table 2. Efficient Algorithm used for Interval multiplication

$p = x_l . y_l$	min1 = min(p,q)
$q = x_l . y_u$	max1 = max(p,q)
$r = x_u . y_l$	Min2 = min (min1,r)
$t = x_u . y_u$	Max2 = max(max1,r)
Z_l=min(min2,t)	Z_u= max(max2,t)

It has much less calculations than the conventional interval multiplication shown in Table 1.The figure 1 shows the calculation of p and q, same can be used to find the values of r and t.

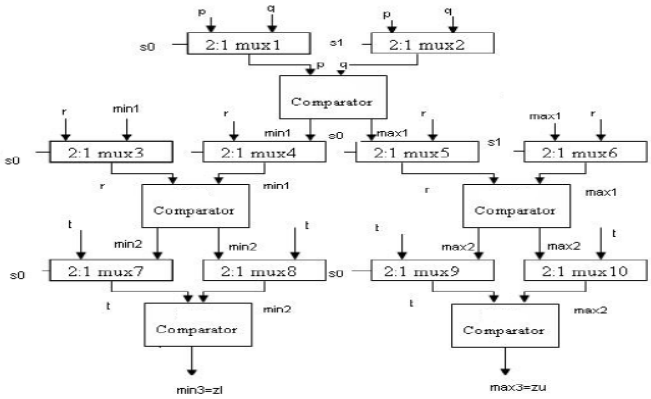

Figure 1.Stage 1 evaluation of Interval multiplier

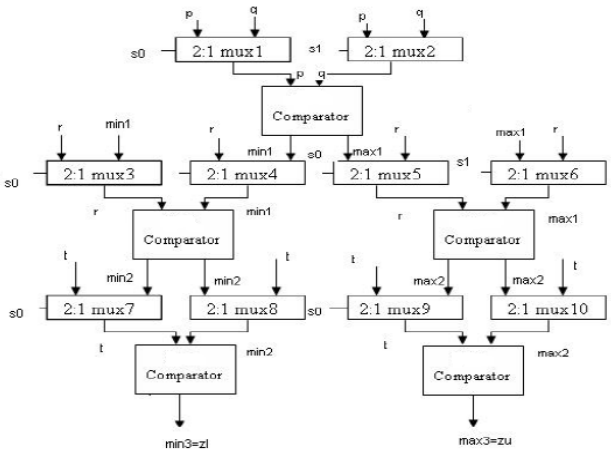

Figure2.Stage2 evaluation

The stage 2 evaluation of the interval multiplier is shown in figure 2. The calculated values of p,q are used as input to mux 1 and r, t to mux2 in stage 2 evaluation. When select line is (s0,s1) ”0”, p and q are selected and passed to 64 bit comparator. When select line is ”1”, r and t are selected and passed to 64 bit comparator, the comparator produces the min1 and max1 values,min1 along with r is given to the mux3 and mux4 respectively,max1 along with r is given to the mux 5 and mux6 respectively. The output of mux3 and mux4(r,min1) are fed into the 64 bit comparator and the output of mux5 and mux6(r,max1) are fed into the 64 bit comparator. Select line along with p, q are passed to muxM5 and muxM6 to evaluate minimum1 (min1) and Maximum1 (Max1) respectively. Next time select line (s02,s12) becomes 01, minimum1 (min1) and r are selected from mux3 and mux4, passed to 64 bit comparator, resultant select line(sel3)along with minimum1 and r are given to mux M7 to get minimum 2 (min2) value. Now select line (s02,s12) become 10 and minimum2 (mim2)and t are passed to 64 bit comparator, depending on select line(sel323) and minimum2(min2),t the result minimum3 (min3) is obtained through mux m9 which is stored in Z_l. Now select line is (s02,s12) made 01, maximum1 (Max1) and r are selected through multiplexers 3 and 4 ,and passed to 64 bit comparator. 3 bit select line (sel321) along with Max1 and r given to multiplexer M8 to obtain the result maximum2 (Max2). Next select line(so2,s12) becomes 10, maximum 2 (Max2) and t are selected through mux3 and mux4, passed to 64 bit comparator, compared, the resultant select line(sel322M) along with max2 and t are given to mux M10 to get maximum 3 (Max3), the result Maximum3 is stored at Zu. Thus interval multiplication result Z is represented as (Zl, Zu). By implementing this algorithm, we have reduced the number of comparisons to almost half of the required.

The proposed interval multiplier is presented in figure 3. The block diagram consists of one 64 bit double precision CSD (Canonic Signed Digit) floating point multiplier[10] and Interval operation selector, multiplexors and registers[10]. The mux1 and mux2 multiplexors select upper or lower end point numbers of intervals based on select inputs s1 and s2 to calculate floating point multiplication. The output of floating-point multiplier is then fed to interval operation selector block [11,12] so as to compare interval arithmetic numbers or floating point number comparisons depending on s3 select line input. In the following discussions we discuss the CSD multiplier used in our proposed implementation of Interval multiplier.

3.1 CSD multiplier

The IEEE 754 complaint floating-point multiplier uses CSD algorithm for faster multiplication of two floating-point numbers[10]. The product of two n- bit binary number is obtained using CSD multiplication algorithm in [(n/2)-1] steps. This in turn significantly reduces the hardware (i.e. the number of adders required) as compared to other multiplication algorithms. The block diagram of complete CSD multiplier unit is shown in figure4. It consists of Two 106-bit registers i.e. mantREG(X) and mantREG(Y) respectively. One 106-bit 2’s Complement unit A CSD multiplication unit. mantREG(X): It is a 106-bit register containing the 48-bit multiplier. mantREG(Y): It is a 106-bit register containing the 48-bit multiplicand. 2’s. Complement unit is used to perform the 2’s complement operation on the 106-bit multiplicand. The CSD Multiplication unit is shown in figure5. This unit provides the 106-bit product by multiplying the contents of mantREG(X) and mantREG(Y) using the Canonic Signed Digit Algorithm.

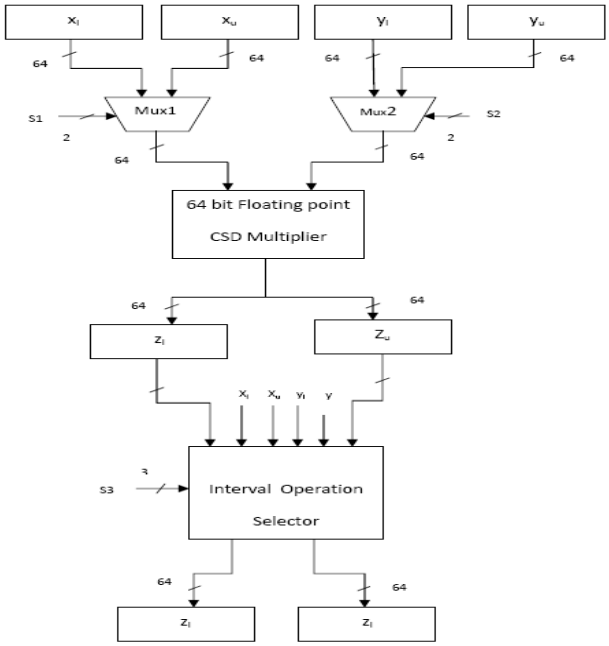

Figure3. The proposed block diagram of Interval multiplier and comparator

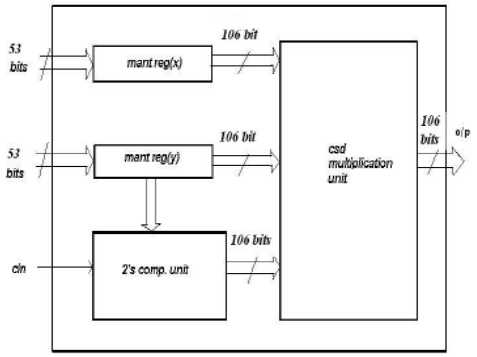

Figure4. Block diagram of complete CSD multiplier unit

This unit receives three inputs of which one is of 53-bits (contents of mantREG(X)) and two are of 106-bits each (mantREG(Y) and the output of the 2's complement unit).The output of this unit is a 106-bit product which is CSD multiplier output. The unit consists of a CSD Vector Generation Unit, CSD Logic and Shift Control Unit and 106-bit Adder Block.

3.2 Interval operation selector

Interval selector block can perform the Interval hull, intersection, Interval comparisons and Interval maxima and minimum and floating-point comparisons[12]. For the floating-point comparison, the floating-point numbers to be compared are stored in the registers X_l and X_u. The minimum of the two numbers is stored in Z_l register and the maximum is stored in the register Z_u.

This operation is completed in one cycle. For comparison purposes, number of instructions required to execute interval operations is taken as bench mark to examine the efficiency of compiler support and interval operation selector. The compiler support for interval arithmetic represents the current state of the art in terms of software support for the interval arithmetic[14]. For this study, the Sun's Forte Fortran 95 compiler is used to produce sequences of instructions to implement interval and floating-point selection operations The number of instructions required for these operations is shown in Table 3. With the interval operation selector, interval intersection, comparisons and interval selection operations require only one instruction as shown in Table 3. With the interval operation selector, there is 80% reduction in number of instructions required to execute interval operations as compared to software solutions.

4. COST AND PERFORMANCE ANALYSIS

The architecture we have described in the last section has been synthesized following a *VHDL*–based design flow [15]. First of all, the logic blocks have been described in VHDL, and simulated to guarantee a correct behavior. Then a single VHDL file has been written, hierarchically connecting all the blocks. After exhaustive simulation, this architecture has been synthesized employing a FPGA Advantage tool. The resulting file of this pre-layout synthesis has been mapped, placed and routed into a Xilinx XC2VP30 FPGA device. A cycle time of 17.347 ns has been achieved as a result of this implementation, which leads to an operation frequency over 60 MHz. The relative area of the used basic gates is presented in Table 4. Two models were build, one for CSD floating-point multiplier and another one for Interval multiplier. The analysis was performed for IEEE 754 double precision number formats[14]. As it is shown in Table 4, the proposed implementation has about 30% cost increase with respect to a conventional multiplier.

5. CONCLUSIONS

This paper presents an effective way to design and implement interval arithmetic units, by proposing a new type of interval multiplier. This multiplier is suitable both for interval multiplication and as well as for floating-point multiplication. The performance of the interval multiplication is better as compared to conventional floating point multiplication as it also does interval comparisons. The proposed IA multiplier uses four multiplications and three comparisons to compute given any two Interval numbers, where as for convention interval multiplication algorithm, the number of steps for the multiplication is variable from case to case, which makes it hard to implement with pipelined structures. The cost overhead of the proposed unit is almost 30% higher, mainly due to the two floating comparators. However, the comparator can also be used for other important interval operations, such as interval hull and intersection. This way, the functionality and performance of the proposed Interval multiplier is increased and, at the same time, a more efficient usage of the active area is achieved.

Table 3. Number of instructions required to execute the operation

Operation	Number of instructions					
	SUN's Forte Fortran 95 Compiler				Interval Operation Selector	
	Interval		Floating-point		Interval	Floating-point
	Min	Max	Min	Max		
Intersection	5	15	N/A	N/A	1	N/A
Hull	5	11	N/A	N/A	1	N/A
Minimum	5	11	2	2	1	1
Maximum	5	11	2	2	1	1

Table 4. Estimated Area for Interval Multiplier unit and for floating- point multiplier Unit

Basic Gates	Not	And,Or,Nand, Nor	Xor	Total
Floating point Multiplier	340	9800	3200	13340
Proposed Interval multiplier	610	11400	6240	18250

References:

[1] Bohlender G, ″What Do We Need Beyond IEEE Arithmetic?", Computer Arithmetic and Self-Validating Numerical Methods, ed., pp. 1-32, Boston Academic Press, 1990.

[2] Goldberg D, "What Every Computer Scientist Should Know About Floating-Point Arithmetic", Computing Surveys, vol.23,no. 1, pp 5-48, 1991.

[3] Schulte M. J, Akkas A, Zelov V, and Burley J.C, "The Interval Enhanced GNU Fortran Compiler", Reliable Computing, vol.5, no. 3, pp. 311-322, Aug. 1999.

[4] Moore, R.E, Interval Analysis, Prentice Hall,Englewood Cliffs, 1966.

[5] L. Jaulin, M. Kieffer, 0. Didrit, and E. Walter, Applied Interval Analysis.Springer, 2001.

[6] W. Edmonson, R. Gupte, S. Ocloo, J. Gianchandani, and W. Alexander, "Highly pipelined Interval Arithmetic Logic Unitfor Signal Processing and Control Applications," in Proc. NSF Workshop Reliable Engineering Computing,Feb. 2006, pp.189-196.

[7] M. Schulte and E. Swartzlander, Jr., "Software and hardware techniquesfor accurate self-validating arithmetic," in Applications of Interval Com putations, R. Kearfott and V. Kreinovich, Eds. Kluwer, 1996.

[8] U. Kulisch, Advanced Arithmetic for the DigitalComputer. New York, Springer-Verlag, 2002.

[9] A. Amaricai, M. Vladutiu, L. Prodan, M. Udrescu, O. Boncalo, "Design of Addition and Multiplication Units for HighPerformance Interval Arithmetic Processor", Proceedings of 10th IEEE Design and Diagnostics of Electronic Circuits and Systems, 2007,pp. 223-226

[10] Y Wang, L.S.De Brunner, D.Zhou. Victor DeBrunner, "A multiplier structure based recoding. IEEE computers, 2007.

[11] A. Akkas, "A Combined Interval and Floating-Point Comparator/Selector", Proceedings of 13th IEEE Conference of Application-Specific Systems, Architectures and Processors (ASAP), 2002, pp.

[12] Rajashekar B. Shettar, Dr. R.M.Banakar, Dr. P.S.V.Nataraj, "Design and implementation of Interval arithmetic Algorithms"ICIIS , Srilanka,2006.

[13] M.J. Schulte,V. Zelov, A. Akkas, and J.C.Burley,"Adding Interval support to the GNU Fortran Compiler,"Manuscript, Lehigh University, 1997

[14] IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE-754/1985

[15] S. Sjoholm and L. Lindh. *VHDL for designers*. Prentice Hall,cop., 1997.

Authors

Rajashekar Shettar received B.E degree in Electronics and Communication Engineering, From Karnatak University, India in 1990 and M.Tech in System and Controls from IIT Bombay, Mumbai. He is pursuing his PhD at Visveswaraya Technological University, Presently he is working as Assistant Professor in Electronics and Communication Engg Dept, BVB Engineering College Hubli Karnataka.

Dr.R.M.Banakar received B.E degree in Electronics and Communication Engineering From Karnatak University India in 1984 and M.Tech in Digital Communication from Regional Engineering College, Surathkal, and Karnataka. She has a couple of years experience in Indian Space Research Organization (ISRO). She completed her PhD in the area of low power application specific design methodology from IIT Delhi in 2004.Presently she is working as Professor and Head of Electronics and Communication Engg Dept, BVB Engineering College Hubli Karnataka. She is the Member of ISTE, IETE, MIE and IEEE scientific and professional societies. Her current areas of research include SOC, VLSI Architecture and WCDMA.