Accurate Floating Point Arithmetic through Hardware Error-Free Transformations

Manouk V. Manoukian and George A. Constantinides

Imperial College, London
{mm3309,gac1}@ic.ac.uk

Abstract. This paper presents a hardware approach to performing accurate floating point addition and multiplication using the idea of error-free transformations. Specialized iterative algorithms are implemented for computing arbitrarily accurate sums and dot products. The results of a Xilinx Virtex 6 implementation are given, area and performance are compared against standard floating point units and it is shown that the time$\times$area product can be improved by a factor of 4 and 6 over software error-free addition and multiplication. Furthermore, it is illustrated that a number of area and performance trade-offs can be made when calculating vector sums and dot products.

1 Introduction

FPGA based designs offer great flexibility to applications modeled using floating point data processing. Designers can choose highly optimized IP cores or customisable and exotic operations [1,2]. The accuracy of floating point operations has always been a major concern and is dealt with in a variety of ways, both in software and hardware [3]. Traditionally, increased precision in hardware floating point operations is achieved by committing to large floating point units. However, if maximal accuracy is required in a calculation, very long fixed point registers have to be used. These registers can be upward of 600 bits in length for IEEE 32 bit standard floating point numbers [4,5].

This work migrates a method known in software to an FPGA context. It is known to be possible to represent the result of an atomic floating point addition or multiplication without any error by using only *two* registers of standard size [6]. The input pair is transformed into an output pair, where one output corresponds to the traditional result and the other to the error present in the calculation. Such error-free operations are very costly in software [6]. However, in hardware the cost is easily reduced, as we show in this paper. Error-free addition and multiplication have been recently used in a number of projects to develop accurate arithmetic algorithms, mainly for sums and dot products [7,8]. Distillation, one of the most popular among these algorithms, is well suited for FPGA implementation. By using the hardware accelerated atomic operations, as well as exploiting parallelism in the algorithm's structure, we can create high throughput circuits for vector sums and dot products.

2 Atomic Operations

2.1 Accurate Addition/Subtraction

Error-free floating point addition algorithms have been known since the mid 1960s. Two of the most well known algorithms are 2Sum which is due to Møller [9] and Knuth [10] and Fast2Sum which was created by Kahan [11] and later formalised by Dekker [6] in 1971. Both algorithms produce the same results but Fast2Sum is arguably simpler to understand. Algorithm 1 transforms the precise unrounded sum of two floating point numbers $a + b$ into the sum of two floating point numbers $x + y$. Number x is equal to the rounded-to-nearest floating point sum $RN(a+b)$ and $y = a+b-RN(a+b)$ which is the exact error in the sum due to roundoff error and insufficient precision. Other rounding modes can be used but only with round-to-nearest it is possible to guarantee no loss of information.

Algorithm 1. The Fast2Sum Algorithm

```
if $|a| < |b|$ swap $a$ and $b$
$x \leftarrow RN(a + b)$
$z \leftarrow RN(x - a)$
$y \leftarrow RN(b - z)$
```

The algorithm requires 6 FLOPs (floating point operations) in total, 3 additions/subtractions, 2 absolute values and 1 comparison [6]. The absolute values and the comparison are used to check whether $|a| \geq |b|$ and swap the numbers if this is not the case. This branch can be especially problematic in heavily pipelined CPUs. Furthermore, the 3 additions exhibit data dependencies and cannot be executed in parallel. In a FPGA, however, we do not have to abide with the limitations of standard floating point adders. It is possible to modify a regular adder to provide both outputs x and y. Such a circuit has been created before [12]; in that work the operation provides the error information on demand, by stalling the pipeline and thus greatly reducing the throughput. In this work the process has been fully incorporated into the design of the adder, maintaining the standard adder pipeline structure.

The logic behind the circuit is straightforward, see Figure 1. When two floating point numbers a, b with $|a| \geq |b|$ are added, the significand of b, sig_b, has to be aligned with sig_a. After the operands have been aligned, some or all of the least significant digits of b can be outside the range of the result mantissa. These "residual" bits that are normally discarded, constitute the error in the addition. The error calculation, besides the isolation of residual digits, needs to account for rounding in the result, for the possibility of subtraction as well as for overflow and cancellation in the fraction addition.

2.2 Accurate Multiplication

Similar to addition, an error-free transformation can be achieved with floating point multiplication as well. The aim is to transform the product of two floating

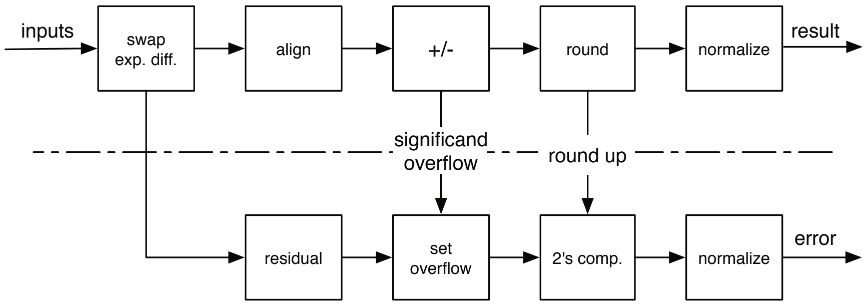

Fig. 1. The error calculation flow diagram on the bottom incorporated with the standard FP adder diagram on the top

point numbers $a \cdot b$ to a sum of two floating point numbers $x + y$, where $x = RN(a \cdot b)$ and $y = a \cdot b - RN(a \cdot b)$. The best known way to achieve this is Dekker's product [6]. This algorithm in software requires 17 FLOPs in total, 7 multiplications, 10 additions/subtractions and has a dependency depth of 8, allowing for parallel execution of multiple operations.

A hardware implementation of error-free multiplication can be considerably more cost effective. When two floating point numbers of precision[1] p are multiplied, the infinitely precise result can only be $2p$ or $2p - 1$ digits wide. The most significant half of that is output once rounded as the significand of the result. The second, least significant half, is normally truncated but it can be used instead to calculate the error in the product. The error calculation circuit has to account for potential rounding in the result as well as set the sign of the error. Unfortunately, the error output needs to be normalised whereas the product output does not need any normalisation. This does create the need for a large normalisation unit and introduces a noticeable delay in the calculations as will be discussed in the conclusions.

3 Accurate Vector Sums and Dot Products

When calculating the sum of multiple floating point numbers with p digits of precision each, often more than p digits are needed to represent the result with maximal accuracy. To achieve this, two major types of algorithms exist in the literature, multiple-digit [4] and multiple-term algorithms [13].

Multiple-digit algorithms use very long accumulators and require some bit manipulation to position the results in the right slot of the accumulator.

Multiple-term algorithms use error-free transformations to produce results that are expressed as expansions $x = x_n + x_{n-1} + ... + x_2 + x_1$. Each component of the expansion has p-digits of precision and is sorted by magnitude $\forall i. |x_{i+1}| \geq |x_i| \wedge (|x_{i+1}| = |x_i| \rightarrow x_i = 0)$. The most important property of the multiple-term

[1] Precision is the length of the floating point number's significand including the often suppressed MSB in radix 2.

result is that the terms are non-overlapping[2]; this means that the result will have the smallest possible number of non-zero terms. The sign of the result is the sign of the largest component since $|x_i| > \sum_{j=0}^{i-1} |x_j|$. Thus, crude approximation of the result can be made from the component with the largest magnitude [7].

This work is focused on error-free transformations and more specifically on a commonly used multiple-term algorithm often referred to as the distillation algorithm [7,8]. Distillation is used to transform an input vector of n floating point numbers into an output vector that holds the exact sum $S = \sum_{i=1}^{n} d_i$ expressed as a multiple-term expansion.

Algorithm 2. The Distillation Algorithm

```
repeat
   for $i \leftarrow 1$ to $n-1$ do
      $(d_{i+1}, d_i) \leftarrow$ Fast2Sum$(d_i, d_{i+1})$
   end for
until $d_i$ non-overlapping with $d_{i+1}$
```

Distillation (Algorithm 2) is achieved by sweeping repeatedly through the input data and applying the error-free addition algorithm (Fast2Sum) until the vector is transformed into the expansion of the accurate sum. The algorithm terminates when all the non-zero elements of the vector are non-overlapping. The number of iterations needed for termination is not predefined and depends not only on the input data, but also on the order the data appears. In the worst case scenario the algorithm would take n iterations to complete. In practice considerably fewer iterations are needed. For example, for large random test vectors (millions of inputs) using the IEEE 754 32-bit floating point format, the number of iterations never exceeds nineteen.

Distillation is not only useful for calculating completely accurate sums. When used with a fixed number of iterations K, then the most significant component of the result (the first element of the vector) is actually the same as if the sum was computed in K-fold precision and then rounded back to working precision [8]. Therefore, distillation can be used for both accurate multi-term results as well as accurately rounded single-term results.

The error-free adder that has been developed can be utilised to replace the Fast2Sum algorithm to accelerate the accumulation (Figure 2). Further acceleration of the accumulation can be accomplished by unrolling the loop and skewing the iterations. In Figure 3 three iterations have been unrolled, and so three different adders can be used in a pipelined fashion. If the number of elements to be added is much larger than the number of adders, then the iterations are executed in parallel. Another benefit of using multiple accurate adders is minimization of the memory traffic, since it is no longer necessary to temporarily store the results of the intermediate calculations. It is likely that the number of iterations needed is not a multiple of the available adders. This means that more iterations

[2] Two floating point values x and y are non-overlapping if the least significant nonzero bit of x is more significant than the most significant nonzero bit of y, or vice-versa.

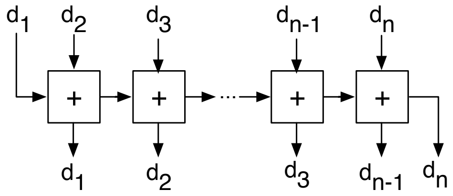

Fig. 2. Single distillation iteration, using a single error-free adder

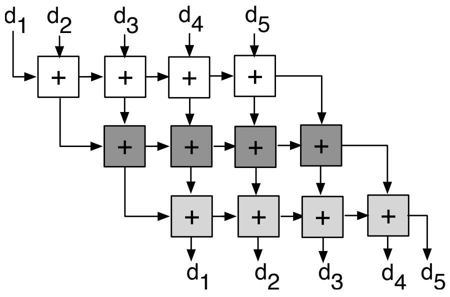

Fig. 3. Three unrolled distillation iteration, using separate adders

might be executed than is necessary. Fortunately extra iterations do not corrupt the result and since iterations are executed in parallel there is no performance disadvantage.

Computing dot products does not pose a difficult task. Consider the dot product $\sum_{i=0}^{n} a_i \cdot b_i$. Dekker's product (error-free multiplication) can be used to transform the products $a_i \cdot b_i$ to the sums $x_{2i} + x_{2i+1}$. Then only the accurate sum $\sum_{i=0}^{2n} x_i$ needs to be calculated. This can be achieved by using an error-free multiplier to convert the input products into sums and then simply apply distillation.

4 Results

For this work the FloPoCo framework [2] was used to develop all of the accurate operations mentioned. Although the VHDL generated is not parametric, we can generate operations of arbitrary size. The most important operation implemented is the error-free adder. If the error output is ignored then the accurate adders generated are indistinguishable from a standardised adder of the same size, both in terms of results, pipeline depth and latency. This means that the developed unit can replace the 6 FLOPs of the Fast2Sum algorithm with effectively one standard addition at only somewhat increased area cost. In Table 1 a 32 bit adder implemented on a Xilinx Virtex 6 FPGA is compared with a error-free adder. The error-free adder has the same latency - it was possible to maintain the same critical path - and requires a 47% larger area for the error calculation circuit. Therefore, the sixfold decrease in FLOPs and the 1.47 increase in area means an improvement in the time×area product by a factor of $\frac{6}{1.47} \approx 4$, compared to software libraries. An adder with double the precision, 47 bits (56 bit in total) instead of 23 bits, is also compared (1). The longer adder does offer increased accuracy but by no means is it error free. The error-free adder can give more accurate results faster and uses significantly less area.

The multiplication comparison metrics in Table 2 show significantly different results. When comparing a standard 32-bit multiplier with the error-free implementation, a large increase in area and latency is apparent. Both of these are due to the fact that we need to use a normalisation unit for the residual.

Table 1. Adder Comparisson on the Virtex 6 FPGA

Adder Type	Number of Slices	Increase	Latency	Increase
32-bit	177 slices	–	20ns	–
32-bit error-free	260 slices	47%	20ns	0%
56-bit	392 slices	121%	33ns	65%

The normalisation unit requires a large number of slices and has a long pipeline that increases the overall latency. On the other hand there is no need for a normalisation unit in the standard design, because the multiplication of numbers will need at most one bit of alignment. The doubling of the area shown in Table 2 refers only to the number of slices used; note that all the multipliers in the table also use the same number (4) of DSP blocks. Software libraries require 17 FLOPs for a single error-free multiplication, our implementation therefore outperforms software alternatives in time×area product by a factor of $\frac{1}{2.02} \times \frac{17}{1.4} \approx 6$.

In a flexible FPGA environment it is possible to have multipliers with standard inputs that produce wider and more accurate results. For example if we opt for a 47-bit precision for the output the result is infinitely precise without any cost in area and with smaller latency. The absolute accuracy is possible because in every floating point multiplier the infinitely precise result is at most $2p$ bits wide and is calculated anyway. The seemingly strange reduction in latency is actually expected since an accurate result does not require any rounding circuitry.

Table 2. Multiplier Comparisson on the Virtex 6 FPGA

Multiplier Type	Number of Slices	Increase	Latency	Increase
32-bits in 32-bits out	60 slices	–	20ns	–
32-bit error-free	121 slices	102%	28ns	40%
32-bits in 56-bits out	61 slices	2%	11ns	-45%

The accurate summation and dot product algorithms are not only accelerated by the aforementioned atomic operations but can also be accelerated by using multiple adders to exploit the parallelism between consecutive iterations. The area consumed increases linearly with the number of adders utilised. The performance increase however, is not linear and is slightly dampened because the design's maximum frequency drops as the FPGA becomes more saturated. Besides the number of adders, another parameter that can be adjusted is the adder width. If the input precision is kept constant, increasing the adder precision will reduce the number of iterations needed to converge to the precise result, but will also make the adder slower. We used random test cases with 32 bit floating point numbers uniformly distributed in the entire floating point range and studied how the accumulator behaves. Figure 4 illustrates that increasing the adders' precision over the original 24 bits, reduces the iterations needed for the algorithm to complete.

If both the number of adders as well as the adders' precision are varied, then the design space for the accurate accumulator is illustrated in Figure 5.

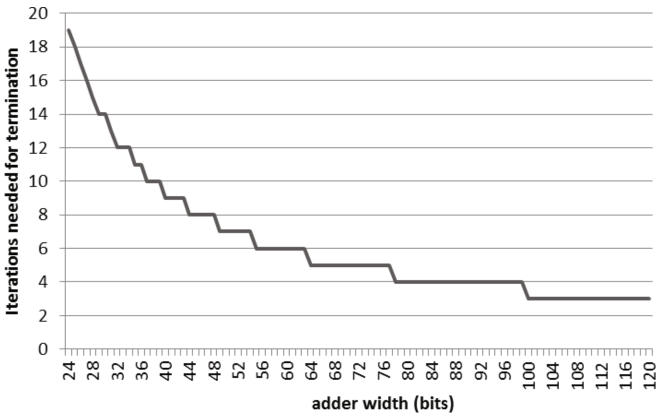

Fig. 4. Effect of the adder precision on the number of iteration required to terminate with a fixed specified accuracy level, independent of adder width

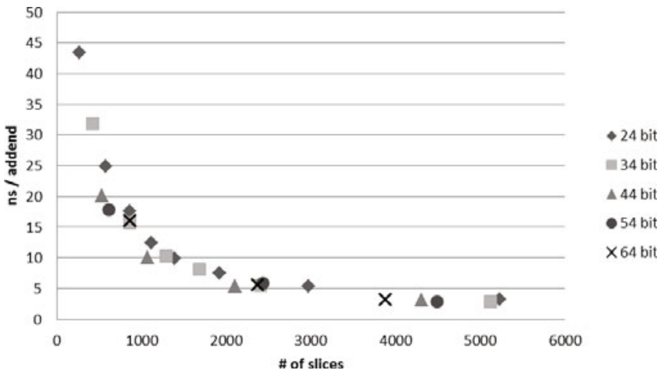

Fig. 5. The design space of a 32 bit accurate accumulator, considering alternative conditions of adder width and iteration count to achieve the same accuracy

The vertical axis illustrates the time increase in the accumulation for every extra addend. The horizontal axis illustrates the area resource and reflects the increasing number of adders from left to right. It is clear that for high throughput applications we will have to sacrifice area to accommodate more adders in the design. However, if we use wider adders it is possible to achieve the same level of performance with a smaller footprint. The benefit of using wider adders is also noticeable but not as prominent in small area designs. By using a wider adder we can more than double the throughput with a reasonably small area investment. Near the performance×area sweet spot the adder width does not play a vital role; the level of performance is dictated mainly by the total area committed to the design.

5 Conclusion

In this work circuits were developed that can replace software based error-free addition and multiplication. The need for such circuits is apparent to researchers working on error-free transformation and they have expressed their wish for hardware support for their applications [8].

Our error-free adder implementation can be used to accelerate algorithms that already use the software versions and can outperform them in time×area product by a factor of 4×. Our implementation is also a viable method for executing accurate addition in general.Results are produced faster, less area is consumed and greater accuracy is achieved compared to adders that are wider than the inputs. On the other hand, although the error-free multiplier implementation can surpass its software counterpart by a factor of 6× as we have shown, it is not useful as a general purpose accurate multiplier. If accurate multiplication is the main design goal then customised standard multipliers offer a better design choice.

Finally, we have shown how the accurate atomic operations can be combined to calculate precise multi-term or accurately rounded single-term sums and dot products. These algorithms are well suited for FPGA implementation because of their inherent parallelism, and we have illustrated the performance-area tradeoffs that can be achieved by varying both the number of floating point units utilised, as well as units' word length.

References

1. VFLOAT, `http://www.ece.neu.edu/groups/rpl/projects/floatingpoint/`
2. Flopoco project, `http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/`
3. Muller, J., Brisebarre, N., de Dinechin, F., Jeannerod, C.P., Vincent, L., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of Floating-Point Arithmetic. Springer, Heidelberg (2010)
4. Kulisch, U.W.: Circuitry for generating scalar products and sums of floating-point numbers with maximum accuracy. United States Patent 4622650 (1986)
5. Kulisch, U.W.: Advanced Arithmetic for the Digital Computer: Design of Arithmetic Units. Springer, New York (2002)
6. Dekker, T.J.: A floating-point technique for extending the available precision. Numerische Mathematik 18, 224–242 (1971); 10.1007/BF01397083
7. Shewchuk, J.R.: Adaptive precision fp arithmetic and fast robust geometric predicates. Discrete and Computational Geometry 18 (1996)
8. Rump, S.M.: Error-free transformations and ill-conditioned problems. In: International Workshop on Verified Computations and Related Topics (2009)
9. Møller, O.: Quasi double-precision in floating point addition. BIT Numerical Mathematics 5, 37–50 (1965); 10.1007/BF01975722
10. Knuth, D.: The Art of Computer Programming: Seminumerical Algorithms, vol. 2. Addison Wesley, Reading (1969)
11. Kahan, W.: Pracniques: further remarks on reducing truncation errors. Commun. ACM 8(1), 40 (1965)
12. Dieter, W.R., Kaveti, A., Dietz, H.G.: Low-cost microarchitectural support for improved floating-point accuracy. IEEE Comput. Archit. Lett. 6 (2007)
13. Ogita, T., Rump, S.M., Oishi, S.: Accurate sum and dot product. SIAM J. Sci. Comput. 26(6), 1955–1988 (2005)