Alternative Approach to Rounding Issues in Precision Computing with Accumulators, with Less Memory Consumption: A Proposal for Implementation

Roy P. Gulla(✉)

555 Highland Dr, Mesquite, NV 89027, USA
rgullape@gmail.com

Abstract. Recent development efforts with the newest version of Unum formatting, posits, have coincided with the continued progress of the new P1788 standard which includes dot products as one of the five fundamental arithmetic operations. These trends have led to increased interest in large scale implementation of accumulators for numerical computation [10], and the creation of encoding and decoding schemes between IEEE float types and posits [5]. Here an alternative in light of these new developments is presented in a way to incorporate the minposis set of Gustafson's Posits formatting, which implements an alternative weighted circuit for fractional bits in his system, capitalizing on inherent memory latency in newer hardware circuitry. A model program template is also touched upon in the new expanded outline of this formatting introduced in [6], proposing the incorporation of fine-grained parallelism, at instruction level.

Keywords: Number representations · Roundoff error · Complex instruction set computer (CISC) · Instruction level parallelism · Fine grained parallelism

1 Introduction to the Approach

1.1 Introduction

Low level hardware designs are rapidly undergoing a major sea level change in response to the new emerging technology surrounding quantum computing. In order to control for some of the inherent volatility in these new products, new formatting for precision computing is taking shape with several performance constraints enacted in order to limit the nearly unlimited bounds for memory consumption in these new devices.

Newer models of hardware and system architectures are paving the way to new models of memory management, which in turn have lead to multilayered computational pathways in hardware circuits. And even given the current trend towards offloading of memory in these newer heterogeneous architectures, in the case of commonly stored precise scientific constants, it is the loading and storing of these large constants which can prove expensive on processor performance.

S. Ao et al. (Eds.): IMECS 2021, *Transactions on Engineering Technologies*, pp. 116–121, 2023.
https://doi.org/10.1007/978-981-19-7138-9_10

Very recently, John Gustafson's POSIT formatting has found some implementations in a square root and divide circuit described in [3], and a new standard creation is underway to directly implement the direct dot product as a fifth arithmetic operation.

In the first of these developments, power consumption is highlighted as a primary concern. And among the differences of the posits formatting than that of the floating point type, the one which might be most attractive to memory management is that of the optional fractional bitfield. So while the IEEE float type can spend the majority of time in determining the right exponent range for a scientific constant, the posits option to exclude the fractional bitfield allows for some more efficient pathways in a high precision computing circuit, so long as the traditional encoding of the floating point standard is abandoned.

In the first portion of the paper the author will describe this proposed arithmetic circuit in light of the new quantum dot cellular automata application implementation with one-bit full adders described in [1], in a way that the instruction set is not in any way bloated with extraneous operands (i.e. store and/or load) (Fig. 1).

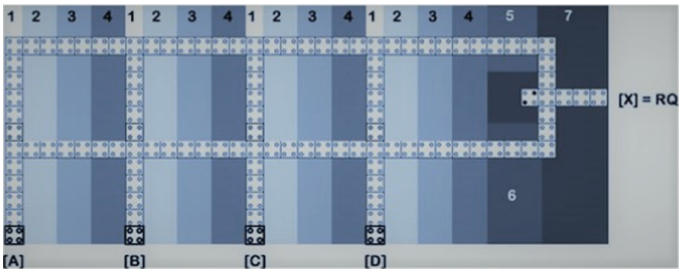


Fig. 1. A parallel to serial QCA w/ majority gate design, which is implemented in [1].

In the final section of this paper a model prototype with minimally accessible instruction sets for global registers will be introduced for implementation as a complex instruction set computer architecture.

1.2 The Problem of Rounding with Variable and Large Word Sizes

The proposed alternative to the typical fractional bitfields of floating point types in the proposed formatting bypasses the issue of loading and storing of intermediate values in memory, which largely contribute to both precision and performance issues in the rounding stages of floating point computation. These are costly operands for comparisons which result in an intermediate difference term for each comparison performed, and it is this comparison difference term which amazingly can require a higher level of precision than the final representation of the number [5].

A simple example of a fundamental issue with binary numbering systems shows how memory constraints can seriously impede pursuits of high precision levels. In the floating point representation of .1, by the tenth subunitary bit, the represented number is still .0008 away from the desired value. And eight bits later, it has overshot this number.

One common example of a non-exactly representable number is presented here in order to introduce the proposed formatting approach of this paper intended to correct for this unpredictable behavior in floating types. In the formatting representation, a repeating minposis, or subunitary, term used in the calculation of the fraction is weighted with integer values. (In the final portion of this paper these weights are encoded as repeated read and shift instructions.)

Decimal Value	**.076923076923..(1/13)**
Alternate Expanded form	**1/13 = (1/16)(1+3/13)** **=(1/16)(1+3(1/16) (1+3/13))** **=(1/16) (1+3(1/16)(1+3(1/16) (1+3/13)))** **etc.**
Minposis Fractional Bits Binary Representation	**00010**

Fig. 2. It is worth noting here that the repeating digits in the lower table's rows hint at a costly string representation, which is avoided here. But the use of weighted bits is important in the sections to follow.

1.3 The Number Rings of These Non-exactly Representable Numbers

Here we intend to introduce a formatting which more efficiently deals with these fractional bit fields that vary in width for each implementation. Particularly, those numbers which can be represented in the following geometric series form fit the proposed formatting:

$$\frac{1/2^{i}}{1-1/2^{i}}, i \in N$$

The approach taken with the these numbers here hopes to present one of the very pronounced differences between the posit's minposis bitfield and the subunitary digits of IEEE floating point types, namely the optional fractional bitfield, and to expand on a possible implementation of these bits.

With a repeating bit pattern as that shown in the above Fig. 2, the floating point fractional bits are both a source of memory and precision issues. The issue is that the repeating remainder in the binary numbering scheme cannot be transformed into binary form, so an iterative expansion approach is taken as shown in Fig. 2 above and Table 1 below, utilizing a feedback adder circuit such as the one in Fig. 3.

1.4 A Common Floating Point Constant with a Weighted Bit Representation

An alternative representation of the natural log base is demonstrated here as a final example for the proposed alternative formatting, incorporating a weighted representation of bits to calculate its value in the accumulator, as opposed to the traditional and costly approach of loading its stored value. A simple description of the iterative scheme used in the computation of its value in real time is given here:

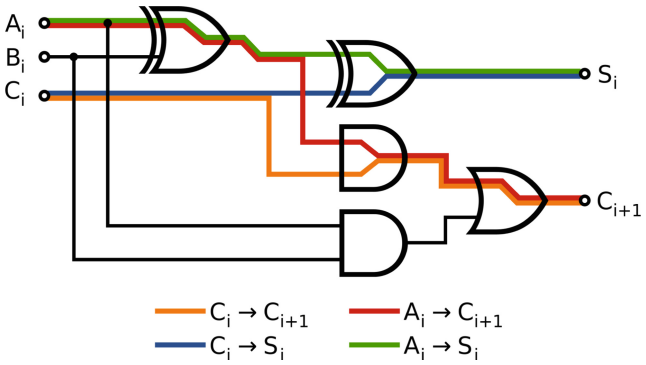


Fig. 3. The design of the serial full adder circuit architecture. In this paper A_i and B_i represent the same bit operand

- Iterate through the subunitary bits, and,
- Repeat the adding of the current bit until the value overshoots, at which point discard the previous summand, and continue the iteration.

This weighted bit representation of the natural log base is shown in Table 1 to machine level precision, and includes the extra 49^{th}, 52^{nd}, and 54^{th} subunitary bits, which can more easily be implemented when no fractional bits are included in the format.

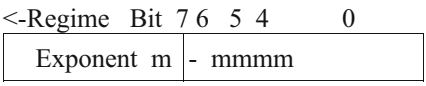
Table 1. The natural log base, with the weighted bit formatting

SET SIGNIFICAND BITS	2,4,5,7,8,9,10,11,12,17,20,21,22,23, 24,25,26,27,29,30,31,32,33,34,35,3 6,37,38,39,40,42,43,44,45,46,47,49, 52,54
CORRESPONDING BIT WEIGHTS	2,2,2,2,2,2,2,2,2,2,2,2,4,2,2,2,4,4,2,2,2,2,4,2,2,2,8,4,4,8,8,8,4,4,4,2,2,2,2,1
SUPERUNITARY BIT	1

2 Proposal for Implementation of the Approach

In [5], the difficult case of a non-exactly representable number in a formatting system with large storage constraints is clearly demonstrated. Summarizing the point here, these constraints place limitations on allocating enough memory for both wide exponent bit fields and wide fractional bit fields. The solution proposed here is to sidestep the wide (and in Gustafson's system, optional) fractional bit field by utilizing a single set bit in a QCA feedback loop to compute our minposis values.

Implementing these feedback loops, much as feedback loops are in [11], the intention is to wrap pipelined arithmetic circuits into posit decoders/encoders, particularly for those numbers where the fractional portion of the posits number is a significant enough part of the representation. Since this portion of the posit number could be computed



m-mmmm (fractional bits alternate encoding):
0- 0010:
0- 0010:
0- 0010:
0- 0010:

Fig. 4. An illustration of copies of the proposed alternative to the fractional bitfield (with a word size of one half byte) necessary for our offloaded compute instructions is shown. Note the size of the exponent or the regime fields is not dealt with here, as the format is proposed to be implemented in a vector processor. Here the weight value for our arithmetic circuit is 4. The multilayered QCA architecture could implement each copy.

simultaneously with the regime and/or the exponent in the QCA architecture described in [11], no additional performance costs would be incurred with this alternative to the posits' fractional field. And a suggested implementation of these arithmetic circuits utilized in place of the fractional bits would be with a complex instruction set as that of the arm neon processor.

As implementations of these numerical types go through layers of architecture to higher levels, what once was a double type can become mixed with integer and decimal types, and precision suffers under these race conditions. And when these global (general purpose) registers are implemented in high performance compute nodes being shared on a large cluster of systems, either data corruption, or performance fall-off becomes an issue.

Therefore the design choice for the implementation here is to avoid altogether the use of global general purpose registers, and in particular to avoid the implementation of the proposed instruction sets in any higher level language wrappers, and instead pursue the course of instruction level parallelism. This both enables the avoidance of global general purpose registers, and implements more local and effective usage of memory, making global memory registers unnecessary. And as architectures trend more and more towards multithreaded ones, the larger blocks of instructions which are mapped into ISBs should be grouped together at the earliest point in the control flow graph anyhow (i.e. in the closest sequential proximity to each other in the code block). In the case of dataflow architectures where token matching is a crucial part of the compilation process, this can prove vital to processor performance. Therefore we attempt not to branch anywhere outside our current block of compute instructions.

In order to do so, we suggest a code design already in place in some legacy and most newer arm neon processors, with recursive C macro calls to generate the product of sums circuitry.

```
#define D2 (X) D1 (D1 (X) D(D(D() ) ) )
```

3 Suggestions for Further Development of the Model

However fine grained we design our proposed parallelism for the arithmetic circuits used in these computations, and restrict our memory space consumption, this limiting of memory consumption will in no way damper our QCA architecture's performance.

For the purpose of ease in the testing of the model, we suggest a word in our vectorized system with one set bit (as shown in Fig. 2 and Fig. 4) in the initialization stages. Following this line of thought, in-wire processing described in [11] is a suggested way of incorporating one bit blocks of data in the circuits, with multiple fetch cycles (by means of a feedback circuit as described in [11]) being used to "weight" the bits.

Complex instruction set computers such as those produced by Intel have shown how instruction level parallelism can be successfully implemented to maximize the performance of their supercomputing architectures. As branching is already an available mechanism engineered into modern arithmetic circuits, the proposal is to implement such instruction level parallelism at the most localized level, by means of the branching mechanism alluded to in Sect. 2 above.

References

1. Angizi, S.F.: A structured ultra dense QCA one-bit full adder cell. Quantum Matter **4**, 1–6 (2015)
2. Bian, T.D.: Context aware quantum simulation of a matrix stored in quantum memory. Quantum Inf. Process. **18**(12), 1–12 (2019)
3. Cheng, S.L.: Posit arithmetic hardware implementations with the minimum cost divider and square root. Electronics **2020**(9), 1622 (2020)
4. Enbody, R.J.: Performance monitoring in advanced computing architecture. In: WCAE 1998: Proceedings of the 1998 Workshop on Computer Architecture Education, Barcelona, pp. 17-es. ACM, 1998 June
5. Florent de Dinechin, L.F.-M.: Posits: the good, the bad and the ugly. hal-01959581v3 (2019)
6. Gulla, R.P.: Two alternative approaches to rounding issues in precision computing with accumulators, with less memory consumption. In: Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2021, 20–22 October 2021, Hong Kong, pp. 240–243 (2021)
7. Gustafson, J.: The End of Error: Unum Computing. CRC Press, Boca Raton (2015)
8. Hittinger, J.L.: Universal coding of the reals: alternatives to IEEE float. In: CoNGA 2018: Proceedings of the Conference for Next Generation Arithmetic: March 2018, Singapore, pp. 1–14. ACM (2018)
9. Johnson, J.: Rethinking Floating Point for Deep Learning. arXiv:1811.01721 (2018)
10. Koenig, J.: A hardware accelerator for computing an exact dot product. In: 2017 ARITH Symposium on Computer Arithmetic, London, pp. 114–121. IEEE (2017)
11. Kogge, P.N.: Exploring and exploiting wire-level pipelining in emerging technologies. In: ACM SIGARCH Computer Architecture News. ACM SIGARCH Computer Architecture News, pp. 166–177 (2001)
12. Kumar, R.P.: A new compiler for space-time scheduling of ILP processors. Int. J. Comput. Electr. Eng. **4**(3), 536–543 (2011)