ELEC4712: Thesis A Progress Report Formal Verification Principles of a Block Mini-Float Multiply Accumulator Unit

WILLIAM N. MARAIS

Supervisor: Professor Philip Leong

A thesis submitted in fulfilment of
the requirements for the degree of
Bachelor of Engineering (Honours)

School of Electrical Engineering
Faculty of Computer Engineering
The University of Sydney
Australia

24 May 2024

Abstract

This paper aims to detail the progress made to develop the method of a formally verified Block MiniFloat Multiply-Accumulate (MAC) Unit. It initiates a forage into adders, before diving into floating point arithmetic. The paper then attempts to implement a MiniFloat adder that handles overflow and normal/denormal values. It showcases the full range of possible MiniFloat values and by extension the level of precision obtainable. The literature review explores concepts that are core to the understanding of the representation of floating point numbers. Building on this is the connection to verification, wherein the stability and correctness of computer arithmetic is examined and proved to be correct without a doubt. The review concludes with an insight into the design of MAC units in the past to gather a foundation on which to implement the Block MiniFloat MAC unit. This progress report does not follow the exact suite of the final thesis paper, and instead aims to show the mistakes and changes made as the semester goes. This paper also serves as a first attempt at literature review and beckons for a constructive review. Please refer to the Git-hub project space https://github.com/wmar9887/wnm-thesis.

Acknowledgements

- I'd like to formally thank my thesis professor Dr Philip H.W. Leong for his leniency and understanding over a tough personal year and driving me to dig deeper and achieve more than I initially aimed for.
- To all my friends and family who supported me in whichever way that could.
- To BAE Systems for supporting me and allowing me to work freely on unscheduled days so that I could manage my studies.

Contents

CHAPTER 1

Introduction

1.1 Purpose

The purpose of this thesis is to explore the validity and potential of the alternate representation of floating point named Block MiniFloat (BMF). The MiniFloat formats are of interest to a wide range of applications, such as training of Deep Neural Networks (DNN) where the hardware imposes constraints. The low footprint of Block MiniFloat format has potential for high energy efficiency, and exceptional performance. However, it is clear that ensuring accuracy confidence is a high priority of the floating point format if it is to gain recognition and further research. To that goal, this progress report will clarify the goals of the thesis to the following:

- To analyse and verify the mathematical properties of floating-point arithmetic using 8-bit Block MiniFloat.
  - Commutativity
  - Associativity
  - Distributivity
  - Rounding
  - Precision

1.2 Background

Floating point numbers are essential in computing for representing real numbers that require a wide dynamic range. Unlike fixed-point numbers, which have a limited range and precision, floating point numbers can handle very large and very small values, making them indispensable in scientific calculations, engineering simulations, graphics rendering, and various other domains. The IEEE 754 standard defines the format for floating point arithmetic, ensuring consistency and reliability across different computing platforms (Fox 2021).

Computer arithmetic, specifically floating point arithmetic, is crucial in enabling high precision calculations. However, due to the finite representation of numbers, issues such as rounding errors, overflow, and underflow can occur. These inaccuracies can propagate and significantly affect the results of computations.

Formal verification is a rigorous method used to ensure the correctness of systems, including computer arithmetic operations. In the context of floating point arithmetic, formal verification involves mathematically proving that algorithms and implementations conform to their specifications. This process helps identify and eliminate errors that traditional testing might miss, providing a higher level of assurance about the system's reliability.

Formal verification techniques are increasingly vital as computational tasks become more complex and the demand for precision grows. For example, in the verification of floating point units within processors, formal methods can prove properties such as correctness, commutativity, and associativity. This is especially important in safety-critical applications, where computational errors can lead to catastrophic outcomes.

The Block MiniFloat (BMF) representation, an emerging floating point format, exemplifies the ongoing efforts to optimise computer arithmetic. BMF aims to provide a more energy-efficient alternative while maintaining performance and accuracy. Formal verification of BMF involves ensuring its mathematical properties, such as overflow/underflow behaviour, rounding precision, and special case handling, are sound and reliable.

CHAPTER 2

Literature review

2.1 Block MiniFloat Representation and Arithmetic

The rapid advancements in deep neural networks (DNNs) have underscored the need for efficient training methodologies. Traditional floating-point representations, such as FP32 and FP64, often lead to high energy consumption and substantial hardware overhead. Recent trends have explored narrow floating-point representations, known as minifloats, to mitigate these issues. The Block MiniFloat (BMF) representation introduced by (Fox et al. 2021) and (Fox 2021) offers a promising approach to training DNNs with significantly lower precision, thereby enhancing computational efficiency and reducing energy consumption.

Floating-point representations are widely used in DNN training due to their broad dynamic range and precision. However, formats like FP32 and FP64 demand high memory bandwidth and incur significant hardware costs. To address these limitations, reduced precision formats such as FP16 and BFloat16 have been adopted by major hardware manufacturers like NVIDIA and Google. These formats strike a balance between precision and resource efficiency, yet further improvements are necessary to meet the growing computational demands of modern DNNs.

Minifloats, which use fewer bits (typically ranging from 4 to 8), lead to increased computational density and lower power consumption. (Fox et al. 2021) have demonstrated that 8-bit minifloats can achieve FP32-level accuracy across different tasks and datasets. Minifloats are particularly advantageous because they allow for a flexible trade-off between dynamic range and precision, optimising hardware resources.

Block floating-point (BFP) representation, a precursor to BMF, shares exponents across blocks of numbers to provide a coarse-grained dynamic range. While BFP offers improvements in hardware efficiency, it often incurs accuracy losses on complex datasets. Minifloats, such as HFP8, use a similar approach but dedicate more bits to exponents, maintaining high training accuracy at the cost of increased hardware complexity.

(Fox et al. 2021) introduces Block MiniFloat (BMF), extending the concept of BFP by integrating minifloat values with shared exponent biases across blocks. This representation optimises both the dynamic range and precision of the underlying arithmetic operations, making it suitable for end-to-end DNN training with reduced precision. BM formats generalise a wide spectrum of reduced precision representations, offering a more efficient alternative to both INT8 and FP8.

The BMF representation addresses two primary challenges: minimising data loss with fewer bits and increasing the performance density of floating-point operations. To mitigate data loss, BMF supports gradual underflow, where denormal numbers provide precision close to zero, preventing abrupt truncation of small values. Additionally, BMF utilises a Kulisch accumulator, a fixed-point accumulator that ensures error-free summation of floating-point products, enhancing computational efficiency and reducing the area and power requirements compared to traditional floating-point units. Hardware synthesis results demonstrate that BM formats with up to 4 exponent bits achieve a substantial reduction in hardware area while maintaining training accuracy.

Empirical evaluations of BM on standard DNN benchmarks, such as ResNet trained on ImageNet, show that 6-bit BM achieves near FP32 accuracy with FMA units that are 4.1× smaller and consume 2.3× less energy than FP8 units (Fox et al. 2021).

2.2 Formal Verification of Floating-Point Hardware Design

Floating-point arithmetic is a cornerstone of modern computing, enabling a wide range of scientific, engineering, and financial applications. Ensuring the correctness of floating-point hardware designs is critical due to the potential for subtle and significant errors in computation. David Russinoff's book, Formal Verification of Floating-Point Hardware Design: A Mathematical Approach, addresses this challenge by presenting formal methods for verifying floating-point hardware, ensuring compliance with IEEE 754 standards, and improving reliability in hardware implementations.

Floating-point arithmetic facilitates the representation of a vast range of values using a finite number of bits. The IEEE 754 standard specifies formats and operations for floating-point arithmetic, emphasising precision, rounding, and special values like NaN and infinity. Despite its widespread adoption, implementing IEEE 754-compliant floating-point hardware is fraught with complexity, leading to potential inaccuracies and inconsistencies (Russinoff 1994).

Formal verification employs mathematical techniques to prove the correctness of hardware and software designs. This is crucial for applications requiring high precision and reliability, such as scientific computing and cryptographic algorithms. Traditional testing methods are often inadequate due to the sheer number of possible floating-point inputs and operations. Russinoff's book highlights the necessity of formal verification in achieving reliable and accurate hardware designs.
Several techniques have been developed for the formal verification of floating-point hardware, and Russinoff's work thoroughly examines these methods:

- Theorem Proving
  - Theorem proving involves the use of formal logic to prove the correctness of hardware designs. Tools like HOL Light and Coq provide frameworks for verifying floating-point operations by constructing formal proofs. While theorem provers require significant manual effort and expertise, they offer high assurance of correctness.

//

- Model Checking
  - Model checking automates the verification process by exhaustively exploring the state space of a hardware design. Tools such as SMV and NuSMV have been adapted for floating-point verification. Model checkers can handle complex designs with minimal manual intervention, though they may struggle with scalability due to state space explosion.
- Symbolic Simulation
  - Symbolic simulation combines aspects of theorem proving and model checking, using symbolic values to represent multiple concrete values. This approach can efficiently verify properties of floating-point circuits. Tools like KLEE and STP support symbolic execution, enabling the verification of floating-point software that runs on hardware.

In summary, David Russinoff's approach offers a thorough exploration of the principles and techniques for verifying floating-point arithmetic. The insights from this work are particularly relevant to this thesis, providing a solid foundation for analysing and verifying the Block MiniFloat format. By leveraging the formal verification methods discussed in the book, this thesis aims to confirm the accuracy and reliability of BMF.

2.3 Literature Review: Design and Performance Analysis of Multiply-Accumulate (MAC) Unit

Multiply-Accumulate (MAC) units are fundamental building blocks in digital signal processing (DSP) and neural network applications. Their performance and efficiency significantly impact the overall system's computational capabilities. The design and performance analysis of MAC units explores various architectural optimisations to enhance speed, area, and power consumption (SaiKumar et al. 2014).

MAC units perform the operation $Y = \sum(A_i \times B_i)$, which is crucial in algorithms involving convolution, filtering, and matrix multiplication. Traditional MAC architectures focus on maximising throughput and minimising latency. However, the increasing complexity of DSP and machine learning algorithms necessitates advanced MAC designs that balance performance with power efficiency and area constraints.

Several architectural designs of MAC units have been discussed (SaiKumar et al. 2014). The conventional MAC unit consists of a multiplier followed by an adder and an accumulator register. This straightforward design prioritises simplicity, but often falls short in high-performance applications due to its limited speed and efficiency.

To enhance performance, pipeline architecture segments the MAC operation into multiple stages. Each stage processes a part of the operation, allowing for parallelism and higher clock frequencies. While pipelining reduces latency and increases throughput, it introduces complexity in terms of control logic and register management.

Various optimisation techniques to improve MAC unit performance have been explored (SaiKumar et al. 2014). Reducing power consumption is critical for battery-operated devices and energy-efficient systems. Techniques such as clock gating, operand isolation, and dynamic voltage scaling are employed to minimise power usage without compromising performance. Clock gating reduces switching activity by disabling the clock signal to idle components, while operand isolation prevents unnecessary data transitions.

Performance enhancements focus on increasing the speed and throughput of MAC operations. Techniques include the use of advanced arithmetic circuits like carry-save adders and Wallace tree multipliers, which expedite the addition process by reducing the number of sequential adder stages.

Experimental results comparing various MAC architectures using standard benchmarks indicate that pipeline and parallel MAC units achieve significant performance improvements over conventional designs. Specifically, pipeline MAC units show a 2x increase in throughput with a slight area and power overhead, while parallel MAC units demonstrate a 4x increase in performance at the cost of higher power consumption (SaiKumar et al. 2014).

The design and performance analysis of MAC units are crucial for advancing DSP and neural network applications. Architectural optimisations enhance MAC unit efficiency in terms of speed, area, and power consumption. Future research should continue to explore innovative techniques to further optimise MAC units for emerging computational demands.

CHAPTER 3

Method and Results

In this section, we will attempt to summarise the efforts of the semester. From a high level view, the majority of time was initially spent reading various websites, articles, and books about what formal verification entails. With cursory knowledge at hand, we began to research Block Minifloat. It became immediately clear that key information was lacking on the concept of floating point (FP) representation as a whole. We then proceeded tentatively to create a floating point adder.

3.1 Phase 1: Weeks 1 – 4

All work going forward was compiled and run through the Jupyter Notebook platform. The original plan was to use existing literature and build a simple 32-bit adder that could catch things like overflow and underflow. Since this is a relatively simple program, we moved onto implementing the same program but with floating point representation. A focus was placed upon making sure that all arithmetic no matter the representation (int, float etc.) did not rely on the in built functions that the python compiler would do naturally for programmers. This led to the decision to implement bit-shifting and bit-masking in order to extract the relevant information, and assign it to variables which we could work with.

3.2 Phase 2: Weeks 5 – 8

We started with a 32-bit Floating Point adder, however it became clear that something was not well understood and was causing the program to output consistently wrong values. This

caused a long lead time to figure out the potential problem. In an attempt to regain control, it was thought that dealing with a larger bit-width was only adding to the confusion, and so we switched to the original task; the MiniFloat format. This proved to be the right direction, as the smaller exponent and mantissa sizes allowed the numbers to be more digestible and easily readable.

In keeping with the simplification process into a journey with smaller sub-tasks, the next thing created was a small function which could take all possible permutations of an 8-bit binary number (i.e. $0b'00000000 - 0b'11111111$) and outputs its corresponding decimal representation. This was a great tool as it allowed us to see the precision that MiniFloat could output at different values of exponents at a certain bias. When the exponent is equal to zero we enter the denormal value region which gives us superior precision close to zero.

3.3 Phase 3: Weeks 9 – 12

Unfortunately, as timetabling clashes occurred, work on the program slowed. The program was able to output the full range of possible MiniFloat values, and a supplementary program was quickly implemented to help print out auxiliary information so that the output from the program was not a simple value. However, the adder was still not functioning correctly. The fix was in the form of three stages:

- Handling of denormal numbers and normalisation
- Exponent difference and overflow
- Removing leading ones

To add MiniFloats, we start by taking the difference of the unsigned exponents, which as an integer is straightforward. Depending on whether the first float's exponent is larger than the other or vice versa, you apply this difference as a bit-shift on the smaller mantissa to the right. This creates a paradox wherein the best results occur when adding two values with similar exponent sizes. This is due to the limiting factor of MiniFloats only having three bits for the mantissa, and thus an exponent difference of more than three causes all precision to be lost in

the smaller number. Once the mantissas have been adjusted accordingly, they can be added. This takes into account the sign bit, and allows for positive/negative values. We create several variables to store the final sign, exponent, and mantissa for the final transformation to decimal representation.

In order to have this calculation be correct, it's imperative that denormal/normal values are handled in the same manner. This requires the shifting of bits to either add a leading one for normal values or to raise the denormal up. Furthermore, when mantissas are added, there is naturally the chance of overflow. This has to be caught and passed onto the exponent so that the magnitude of the value can scale. The value is normalised, and the leading one is removed afterwards and calculated.

We then select the method of calculating the decimal value based on the exponent:[1]

- Bias

$$\beta = 2^{e-1} - 1$$

- Denormal: *When E = 0*

$$X\langle e, m\rangle = (-1)^S \times 2^{1-\beta} \times (0 + F \times 2^{-m})$$

- Normal:

$$X\langle e, m\rangle = (-1)^S \times 2^{E-\beta} \times (1 + F \times 2^{-m})$$

This is where the project has left off for now, and will continue to be expanded on. This final phase in semester one had a big breakthrough not only in the program but in conceptual understanding that allows further substantial testing and implementation of mathematical verification.

[1]Where S, E, and F, β represents the Sign, Exponent, Mantissa values and exponent bias for the binary offset encoding scheme.

CHAPTER 4

Conclusion

In conclusion, this first semester has showcased a greater than expected learning difficulty with regard to implementing floating point representations. This necessitated research to be undertaken to shore up knowledge deficiencies, and to embark and on an arduous trial and error pathway. We were able to create a program however which can successfully take two MiniFloat binary values and compute them to give the correct decimal value. Further, it is supplemented with two functions; one which calculates an individual MiniFloat binary value in decimal, and another that calculates the entire range of MiniFloat values. This allows us to display the final operation with supporting details, as well as compare the final value to the range of possible outputs.

Progression into Thesis B

From this point, it is believed that there will be a direct route towards a general verification claim. As stated initially in the introduction of this paper, the mathematical foundations such as commutativity, associativity, and distributivity will be the focus going forward, and there is high confidence in its achievability. Once the adder is verified, the multiplier will be assimilated into the model. Finally, the MAC unit will be put together in a Kulisch Accumulator arrangement.

Bibliography

Fox, Sean (2021). ‘Specialised Architectures and Arithmetic for Machine Learning’. PhD thesis. University of Sydney, pp. 24–30.

Fox, Sean et al. (2021). ‘A Block MiniFloat Representation for Training Deep Neural Networks’. In: *Proceedings of ICLR 2021*. Sydney Australia, pp. 1–5.

Russinoff, David (1994). *Formal Verification of Floating-Point Hardware Design: A Mathematical Approach*. 2nd ed. Austin USA: Springer.

SaiKumar, Maroju, D.Ashok Kumar and P.Samundiswary (2014). ‘Design and Performance Analysis of Multiply-Accumulate (MAC) Unit’. In: *Proceedings of ICCPCT 2014*, pp. 1084–1089.