Comparing High Throughput 16-bit MACs Using Residue Number System and Pipelining

Chris Henk
cahenk@smu.edu

***Abstract*—In this paper, we present two specialized designs for high throughput multiply accumulate units (MACs). Current machine vision algorithms rely heavily on fast MACs to perform the dot product operations in Convolutional Neural Networks. Often, 16-bit floating points offer sufficient precision, but existing designs consider wider MACs that perform suboptimally. We propose specialized residues that leverage the lesser dynamic range requirements in this application and provide proof of concept designs. The designs are implemented with fast logic that would be inordinately expensive to use with larger values. Our findings indicate that small residue MACs can improve throughput by 10.5% at just 33.5% additional area despite using FGPA's, which are unfavorable to the design. We conclude that our design has superior performance characteristics to typical MACs in hardware accelerated machine learning applications.**

I. Introduction

Machine vision is an integral component of many artificially intelligent products on the market today. These algorithms are typically implemented using Convolutional Neural Networks (CNNs), which rely heavily on matrix multiplication and convolution operations. These operations can be performed in parallel and use large data sets. Arrays of Multiply Accumulate Units (MACs) are used to take the sum of many products with high performance. Since latency is not a major performance factor, hardware designs should be designed to maximize throughput.

In CNN-based machine vision tasks, half precision 16-bit floating point values can be used with little impact on classification accuracy while using far less memory than full precision values. Consequently, many developers choose to lower memory utilization by truncating values. Existing residue number system (RNS) MACs are designed for wider inputs and must pad the input data. In addition to wasting hardware, this also leads to suboptimal performance. When designing an RNS, delay trade-offs must be made to support the required dynamic range and implement the arithmetic circuitry within cost constraints. If less dynamic range is required, the RNS can be specialized to support fast arithmetic operations without dramatically increasing costs. The critical path can be reduced to just a handful of gate delays which, when combined with pipelining in larger operations, dramatically improves throughput.

This paper presents two distinct sets of residues that have been selected to be compatible with efficient arithmetic hardware units. One set is optimized for use in hybrid carry-look-ahead (CLA) modulo adders that utilize lookup tables (LUTs) to perform efficient modulo operations while the other set is optimized for use in pure LUT-based modulo adders. Each set contains just 4 residues ranging from 4-bits to 2-bits in size. The smallest possible values are used to reduce the number of address lines required for the LUTs. The one exception is the largest residue in the CLA/LUT design, which leverages wiring to perform modulo 2^n operations for free. This has important implications for both the area and delay characteristics of the LUTs. Delay is reduced to less than that of the CLA addition operation which, with proper pipelining, eliminates the typically expensive modulo operation from the critical path of the design. Storing fewer table entries has an obvious positive impact on area. The LUTs must be as small as possible since the design makes extensive use of them in both the modulo adders and, consequently, the modulo multipliers. The overall area complexity required to implement the design is primarily a function of LUT size.

Because latency does not have any significant impact on the performance of our hardware in this application, the design optimizes for throughput. This is accomplished by reducing each operation to only a handful of gate delays and placing registers between them to allow for pipelining. Small residues are highly amenable to this goal because carry propagation delay is minimal in arithmetic logic and they produce simpler LUTs. Modulo operations, one of the more expensive portions of RNS arithmetic designs, can therefor be accomplish at high speed with low area costs.

Our proof of concept design implementation and testing verify the correctness of the arithmetic operations and evaluates the performance of the proposed designs. The pure LUT design is demonstrated to offer no performance benefits over industry standard, high throughput MACs when implemented in two competing FPGA architectures. The hybrid CLA/LUT design is shown to offer significant performance improvements over conventional designs with little additional area required. Because the critical path of RNS encoding is the modulo add operation, encoding will not require the frequency to be decreased. RNS decoding performance is neither helped nor harmed by our optimizations and any delay inconsistencies can be resolved by performing decode operations only once at the end of computation and/or by introducing minor controller logic to multiple decoder units.

While the proof of concept failed to demonstrate that the pure LUT based MAC was superior to industry standard implementations, it is possible that it would perform better under real world circumstances. This circuitry is intended to be realized using emitter coupled logic (ECL) technology, which would enable superior LUT performance at reduced cost with wired ORs. Future work utilizing tooling that supports ECL realization could investigate if the outcome would change. The hybrid CLA/LUT design is faster than industry standard methods with an additional cost low enough to justify its use in high performance applications. By supporting efficient encoding and not harming the decoding process the high performance modulo adders can be integrated into pipelined MACs without increasing delay.

The remainder of this paper is organized as follows. In Section II we provide an overview of residue number systems. Section III reviews related work on implementing fast MACs with RNS. In Section IV we introduce our novel MAC and the associated RNS scheme. Section V discusses our proof of concept design. In Section VI we show how the design was verified and performance metrics were acquired. Section VII contains an analysis of the performance testing. This is followed by our final conclusions in Section VIII.

II. Residue Number System

A variety of computational tasks require the device to take the sum of many products with high performance. This operation can be efficiently performed using a multiply accumulate unit, which computes $a \leftarrow a + (b * c)$. Because there are multiple operations, MACs are ideal candidates for optimization using residue number system (RNS) representations internally. Matrix multiplication and convolutions (which are used in neural networks) exhibit high degrees of data parallelism and create long input queues. Consequently, they pair well with deep pipelines that dramatically increase throughput at the cost of some latency. This limits the overhead of RNS conversion even further since decoding is only strictly necessary after a large number of MAC computations have been performed. Carefully chosen residues allow the necessary computations (modulo, for instance) to be performed more efficiently than is possible in general purpose arithmetic circuitry that must be capable of performing the operation given arbitrary inputs.

Residue number systems improve arithmetic performance by reducing the amount of carry propagation and, therefore, the critical path of the circuit. Instead of being defined by a single radix (i.e. a radix polynomial system), an RNS is defined by an n-tuple of integers called residues $(m_1|m_2|...|m_n)$ [8]. In this system, numbers are represented as the n-tuple of results from the modulo of the number and each corresponding residue. That is, a number X in the RNS $(m_1|m_2|...|m_n)$ is defined as: $X = (x_1|x_2|...|x_n)$ where $x_i = X \ mod \ m_i$. This allows us to represent a large number as several small residues which can be processed in parallel, reducing the maximum carry delay to that of the largest residue which increases performance for adders and, therefor, multipliers [7]. While there is some conversion overhead for encoding to and decoding from RNS, for large numbers it is more than accounted for with faster arithmetic [2]. Additionally, in a MAC the overhead only occurs once for every two operations, versus once for every operation in an RNS based adder or multiplier. Decode penalties can be further mitigated in repeated operations by delaying the decode until the complete sequence of operations has been performed.

III. Related Work

MACs using RNS internally have been extensively studied in the Literature. Like conventional MACs, RNS MACs have been demonstrated to benefit from basic pipelining between arithmetic units. Low-level pipelining has also been shown to produce increases in throughput, provided that the depth of the pipeline is acceptable for the given use case [5]. In such cases, fast multiplication is achievable through staging simple modulo adders. As a result, the throughput of the MAC is determined by the critical path of the modulo adder. Modern designs tend to refine either the modulo adder / modulo multiplier subcomponent of the architecture or apply changes that optimize for a specialized use case [3].

For instance, the generalized architecture proposed by Preethy et. all shows how an arbitrary set of residues can be combined with logarithmic representation lookup tables to eliminate multiplication altogether [6]. This approach provides fast computation, but requires several large ROMs to implement logarithmic conversions efficiently. Because a large dynamic range is supported, the tables must necessarily contain many inputs. If smaller residues could be used instead, these tables would become much smaller. In fact, it would become feasible to skip the addition step entirely and replace the log encode, add, log decode process with a single table lookup.

Specialized designs have the obvious limitation of being suitable only for the narrow application for which they were created for. This limitation is compounded when specialization is based on the expected input and/or output data values rather than the broader arithmetic operation. Specializing to arithmetic operations splits the difference because they receive the performance benefits of specialization while being potentially applicable to multiple problems. The authors of [4], for instance, use an improved reduction method where a special kind of RNS pair is selected that reduces outer reduction delay in elliptic curve operations. The benefits of the design are limited to elliptic curve arithmetic, but there are multiple cryptographic problems that can leverage it. Similarly, a RNS MAC specialized for performing dot products on large vectors of small values could apply to any problem that requires such an computation to be made.

Existing work focuses on large input, generalized implementations of RNS MACs. While some specialized circuity exists for certain digital signal processing (DSP) applications that use smaller input sizes, they are tuned according to the distribution of inputs expected for their given application. Many machine learning applications implementing convolutions neural networks can achieve sufficient accuracy with 16-bit (i.e. half word) inputs. Half word values can represented in RNS using extremely small residues (4-bits or less), but the potential benefits have yet to be explored in the Literature. Given that the critical path in most RNS MACs is the modulo adder, the shorter critical path achieved by using smaller residues provides an opportunity for greater throughput to be achieved than when the design must be constrained to support a larger dynamic range. A smaller valued, high performance design will allow more computations to be performed in parallel on a single chip at a higher performance.

IV. Multiply Accumulate Unit

We implement two versions of the small residue optimized design and compare their performance and cost characteristics. Arithmetic circuits that use RNS require three separate components: a binary to RNS encoder, the arithmetic circuitry, and a RNS to binary decoder. Thus, in order for RNS to be useful in real world scenarios the encode/decode circuity must be efficient. We use standard practice parallel encoder/decoder using LUTs and, in the case of the decoder, the CRT method to meet this requirement. The critical path of standard RNS encoding is the modulo adder, as is the critical path of the proposed MAC. Since the design is pipelined and the critical path of the two components is identical, the throughput gains achieved through our design will not be starved of inputs or require additional encoding units to maximize utilization. The decode operation involves an expensive modulo operation that could potentially increase delay. This can be dealt with by either introducing simple controller logic where a value is only decoded when a flag marking the end of an input vector is set or by providing multiple decoder units to keep the rest of the components at maximum utilization. The MAC unit itself is comprised of an array of modulo multipliers, modulo adders, and accumulation/pipeline registers. Figure 2 details the design of an individual residue unit in the array.

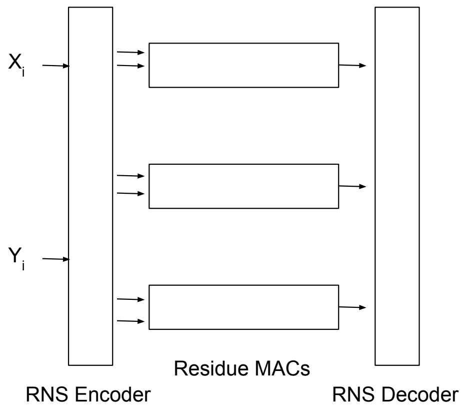

Fig. 1. Typical 3-Stage Residue Number System Multiply Accumulate Unit

Since these operations do not need to impact the performance of the arithmetic portions of our MAC, they are not within the scope of this research and excluded from our prototype. Similarly, floating point operations are implemented using fixed point circuitry with some additional controller logic. The delay of the calculation is dependent on the delay of the underlying computations. Since floating point overhead is independent of the underlying arithmetic, it too is considered to be outside the scope of this research. In IEEE 754, a 16-bit half word floating point uses an 11 bit significand. This is the component with the largest delay, so our prototype implements an 11-bit RNS MAC to determine the theoretical performance figures of the complete design [1]. We propose two sets of residues, each of which is optimized for efficient implementation with a different modulo addition algorithm.

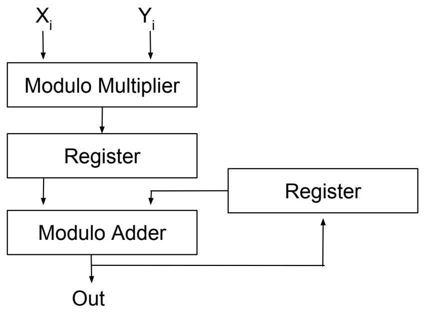

Fig. 2. Multiply Accumulate Unit with Pipelining

A. Pure LUT Modulo Adder

The pure LUT modulo adder is the simplest subcomponent of the design. The modulo table for an addition operation of order $m_i + m_i$ requires $\lfloor log(m_i) \rfloor + 1$ address bits. Whereas the modulo addition table for an addition operation of order $m_i + m_i$ requires $2\lceil log(m_i) \rceil$ address bits. Since the largest residue $m_0 = 9$ the worst case table requires just 3 additional address lines. Thus, especially if the implementation technology can efficiently implement tables, it is feasible to skip the addition all together. With larger tables the delay would grow, but more importantly the area requirements would grow far too expensive. With these considerations taken into account, we propose the RNS $(9|8|7|5)$.

B. Hybrid CLA/LUT Modulo Adder

Traditional fast adders are not costly when the number of bits remains low. Also, the cost of performing modulo is low because a small lookup table for an addition operation of order $m_i + m_i$ requires $\lfloor log(m_i) \rfloor + 1$ address bits. We can also leverage the fact that taking some modulo $m = 2^n$ can be done for free through wiring. Thus, we can pick a larger residue of that order and then use a few additional registers to construct a hybrid CLA/LUT modulo adder offers both high performance and low area characteristics. For the purposes of this prototype, we propose the RNS $16|9|7|3)$. Given that CLA arithmetic delay does not scale linearly, it may be better to chose a 2^n residue that is far larger than the other residues. Determining the ideal residues for such an arrangement is a potential topic for future research. In particular, the goal would be to find a residue such that the residue 9 term could be made smaller and that the additional CLA logic had less delay than the modulo table for residue 9. This is because the residue 9 modulo table ends up being the critical path of the design.

C. Modulo Multiplier

Our design uses parallel shift and add multipliers. The shift operation is performed using small, n-entry LUTs, one for each bit, that have precomputed modulo shift values. An additional mux connected to ground is used to handle the 0 case. The partial products are then combined in $log(n)$

adder stages. The modulo adders, which are the critical path in the pipelined design, perform their computations in just a few gate delays. This design is much simpler than typical multipliers, but doesn't make speed trade offs to achieve simplicity because the adder tree is only 1-2 stages of fast adders. The only difference between the two multiplier designs are the values in the precomputed LUTs and the adder trees, which vary according the residue $m_i$and the number of residue bits $\lceil log(m_i) \rceil$ respectively.

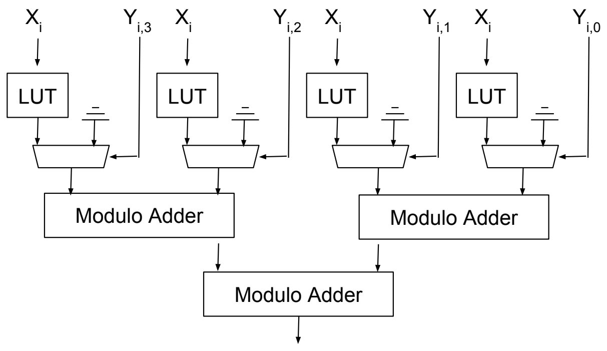

Fig. 3. 4-bit Parallel Shift and Add Modulo Multiplier

V. Implementation

The design would ideally be realized using emitter-coupled logic (ECL). ECL is a common choice for high performance designs, but the specifics of our design are particularly well suited for this technology. High speed transistor state changes create low gate delays, which improve frequency. Since our design uses large amounts of pipelining it's critical path is relatively short, meaning that gate delay has a significant effect on the max frequency. ECL also supports high fanout and the wired OR gate. These qualities improve the efficiency and speed of the large multiplexers that are used to implement the many LUTs in our design. ECL's main downside is increased power utilization, but this is not a concern for high performance applications with access to mainline power.

However, while constructing the prototype we were limited to using FPGAs supported by our tooling. Considering this is a proof of concept phase, and that our performance figures are equal or lesser to what would be seen with an ECL realization. Therefore any conclusions reached in favor of our design will be applicable to real world, ECL implementations. Modern FPGAs are fast enough for use in market products that would benefit from hardware acceleration but can not justify the immense expense of fabricating custom chips. It is common to see them in portable devices and some cloud hosting providers now even offer FPGA products. In addition to being an acceptable model for determining the efficacy of our design, it also shows how the design could potentially be used in FPGAs should a product benefit from it without being able to justify purchasing custom hardware to incorporate it.

VI. Design Verification and Performance Testing

Test vectors were developed for the different levels of the design hierarchy: full design, modulo adders, residue multipliers, and shift LUTs. Because the chosen residues are small, many of the sub-components could be checked for complete correctness by brute force testing every possible input signal. This was done for the pure LUT modulo adder, the modulo LUTs in the hybrid modulo adder, and for the shift LUTs. Tables can be incredibly tedious to produce Verilog for by hand, so they were grammatically generated using a basic python script. The correctness of the data values are ensued by the script and tables are synthesized from case statements, so proper of implementation of the tables themselves is guaranteed. In a real world implementation, similar testing could be done for the individual residue multipliers and adders, but we chose instead to pick representative samples. Once the complete MAC, which is stateful, is considered it becomes impractical to perform brute force testing and representative samples must be chose instead. Samples were chosen from input classes small, large, zero, etc. with consideration given to ensuring that modulo rollover was verified.

Performance testing of our two designs and a control design were performed on two simulated FPGAs: the MAX II EPM570F100C4 and the Stratix II EP2S15F672C3. These devices have different performance levels and different architectures. By including both the results are more representative of how the designs will perform against the control on FPGAs more generally. The control design is a basic pipelined MAC of the kind in Figure 2 that was implemented using "LPM Mega-Functions" under the fast setting. These library components are fast modules meant to simplify designing fast circuits and are reflective of current industry standard performance figures.

VII. Performance Analysis

Table I shows the performance and cost metrics for the competing designs on the MAX II FPGA, while Table II shows the performance and cost metrics for the competing designs on the Stratix II FPGA. In both tables, delay is measured in nanoseconds, frequency is measured in megahertz, and area is measured in logic elements. Our results show that the pure LUT design had a higher delay than initially anticipated, and actually performed worse than the industry standard implementation. As a result the full circuit was not considered, so the area measurements are only for the modulo multiplier and not for the entire MAC. This is not useful for comparison purposes, so the values were excluded from the table. While the pure LUT design was unsuccessful, the hybrid design was not. As the table shows, the hybrid design is significantly faster the the LPM based design is. On the Max II the 10.5% improvement does come at the cost of 33.5% additional area, but the amount is quite modest in comparison to the speedup that was achieved. The Stratix II receives a larger 24.76% performance improvement, but at a far greater area cost of 472% in ALUTs and 318% in registers, which brings diminishing returns into question.

Design	Delay (ns)	Frequency (MHz)	Area (aggregate)
Pure	7.294	127.06	—
Hybrid	4.170	210.7	251
LPM	4.670	190.62	188

TABLE I. Performance Comparison - MAX II EPM570F100C4

Design	Delay (ns)	Frequency (MHz)	Area (aggregate)
Pure	1.726	579.37	—
Hybrid	1.369	730.46	126
LPM	1.709	585.14	22

TABLE II. PERFORMANCE COMPARISON STRATIX II EP2S15F672C3

VIII. CONCLUSIONS

The proof of concept failed to demonstrate that our pure LUT design was superior to standard methods. There are multiple potential explanations for this. The first is that fast arithmetic is more efficient in terms of delay than a lookup table because the number of address bits is too high. Another potential explanation is that the FPGA technology was unable to efficiently implement the tables, but that if it were implemented in a more suitable technology such as ECL we would observe different performance figures. Finally, there may be a better way to implement the table in Verilog and/or a more sophisticated synthesis tool may be able to find a more efficient way to implement it. Determining which of these potential explanations is correct is a potential topic for future work.

Our hybrid design produced better performance metrics at a marginally increased cost over conventional MAC implementations. Our testing methodology provides a proof of concept of the performance benefits at the worst case delay the design would encounter and did so in a technology that is usable in existing products. Additionally, there is evidence to suggest that the results could be improved to an even greater degree in applications that warrant custom chip fabrication. Large compute clusters employing our design could accelerate machine vision training tasks in data centers. Future work refining the design, the chosen residues in particular, could be done to extract additional performance. While the model suggests a full scale design should be viable, after any improvements are made the concept should be developed into a full scale 16-bit floating point unit to bring the design to market.

REFERENCES

[1] *IEEE standard for binary floating-point arithmetic*. Institute of Electrical and Electronics Engineers, New York, 1985. Note: Standard 754–1985.

[2] HOHNE, R. A., AND SIFERD, R. A programmable high performance processor using the residue number system and cmos vlsi technology. In *Proceedings of the IEEE National Aerospace and Electronics Conference* (May 1989), pp. 41–43 vol.1.

[3] LEE, S.-M., CHUNG, J.-H., YOON, H.-S., AND LEE, M. M.-O. High speed and ultra-low power 16/spl times/16 mac design using tg techniques for web-based multimedia system. In *Proceedings 2000. Design Automation Conference. (IEEE Cat. No.00CH37106)* (June 2000), pp. 17–18.

[4] MO, Y., AND LI, S. Fast rns implementation of elliptic curve point multiplication in gf(p) with selected base pairs. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)* (Sept 2017), pp. 1–6.

[5] PIESTRAK, S. J., AND BEREZOWSKI, K. S. Architecture of efficient rns-based digital signal processor with very low-level pipelining. In *IET Irish Signals and Systems Conference (ISSC 2008)* (June 2008), pp. 127–132.

[6] PREETHY, A. P., RADHAKRISHNAN, D., AND OMONDI, A. A high performance rns multiply-accumulate unit. In *Proceedings of the 11th Great Lakes Symposium on VLSI* (New York, NY, USA, 2001), GLSVLSI '01, ACM, pp. 145–148.

[7] R, D., V, B., SAHOO, S. K., SAMHITHA, N. R., CHERIAN, N. A., AND JACOB, P. M. Implementation of floating point mac using residue number system. In *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)* (Feb 2014), pp. 461–465.

[8] SODERSTRAND, M. A., JENKINS, W. K., JULLIEN, G. A., AND TAYLOR, F. J., Eds. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, Piscataway, NJ, USA, 1986.