

# An Array Processor for Inner Product Computations Using a Fermat Number ALU

W. Luo, G.A. Jullien, N.M. Wigley, W.C. Miller, Z. Wang<sup>1</sup>  
VLSI Research Group, University of Windsor, Ontario, Canada N9B 3P4

## Abstract

This paper explores an architecture for parallel independent computations of inner products over the direct product ring  $\mathfrak{R}_{257 \times 17}$ . The structure is based on the polynomial mapping of the Modulus Replication RNS for calculations over dynamic ranges much larger than the product of the computational moduli. We show that the computational ring is optimal for our purposes, and introduce basic cells for the efficient calculation of all elements of the polynomial ring computations.

## 1: Introduction

Computing over cross product rings has several advantages for VLSI signal processing [1]. These include the reduction of clock skew problems, natural fault tolerance [2], and ease of testing [3]. More traditional advantages include the promise of high speed arithmetic because of the removal of the carry, but the former advantages, we feel, are the most important.

Although application areas are quite limited, applications to high throughput DSP are prime targets. The usual finite ring mapping strategy is based on the Residue Number System [5], in which data and coefficients are mapped via the modulo reduction operator  $x_i = X \bmod m_i = |X|_{m_i}$ . Calculations are component wise (over the cross product ring) and results recovered (inverses mapping) via the Chinese Remainder Theorem (CRT) [5].

An alternate mapping strategy is to consider the Modulus Replication Residue Number System (MRRNS) [6]. With this method, the moduli can be repetitively used to increase the effective dynamic range. This greatly relaxes the dependence of the dynamic range on the product of the selected moduli. We are also able to embed an RNS within the MRRNS so that the replicated modulus is decomposed into a small number of much smaller moduli. We have successfully shown the equivalent of 46-bit computation using only the moduli  $\{3,5,7\}$ , albeit with large replication counts [7]. The modulus replication technology enables us to optimally select a set of small moduli and to construct a replicative architecture.

This paper discusses a MRRNS based architecture, using an embedded RNS with two Fermat Primes, 17 and 257; we consider a specific inner product computation, a FIR systolic architecture, on  $\mathfrak{R}_{17 \times 257}$ .

---

1. The authors acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada and the Micronet Network of Centres of Excellence. The authors also acknowledge the loan of VLSI design equipment and software from the Canadian Microelectronics Corporation.

## 2: Modulus Replication

Modulus replication greatly reduces the coupling of the dynamic range and the product of the moduli, and presents an opportunity to select one or two ideal moduli to use repetitively to obtain a large dynamic range.

We represent each number as a polynomial with the indeterminates representing weights of  $2^r$ . This is a ‘free’ mapping, since we need only segment the binary data to the  $r$ -bit pieces. We formally map to a finite polynomial ring, but we select moduli for the ring such that the input coefficient data is less than this value, and so no reduction is required. We now map to a cross product ring and perform all calculations over each *implementation* ring. The final results are obtained via the CRT mapping (undoing the embedded RNS) and inverse polynomial mapping. In the datapath, the subpath for each implementation ring will be replicated to several copies, where the replication factor,  $d$ , is determined by the ratio of the bit-length of the input data and  $r$ .

The main steps in the mapping process are given below:

1. *Write the input data as polynomials.*
2. *Polynomial mapping in each implementation ring.* That is to multiply a  $d \times d$  Vandermonde matrix of all possible roots of the mapping ideal on the left of the coefficient vector [6]. After polynomial mapping, we obtain a  $d$ -tuple vector for each implementation ring, which maps to  $d$  subpaths.
3. *Algorithm calculation in each subpath.* There are  $d$  subpaths for each implementation ring, and each of these subpaths involves identical finite ring computational hardware. Therefore, the MRRNS employs  $d \times L$  subpaths in a whole datapath, where  $L$  is the number of implementation rings for the RNS embedded map.
4. *Inverse polynomial mapping in each implementation ring.* For each implementation ring, the outputs of the  $d$  subpaths construct a  $d$ -tuple vector. The inverse polynomial mapping multiplies the inverse vandermonde matrix on the left.
5. *RNS inverse mapping (CRT).* The Mixed Radix Conversion (MRC) algorithm is used.
6. *Final addition.* In general, the resultant polynomial coefficients exceed the weight of the indeterminate and so any overflow has to be accumulated. The overflow is limited to the dynamic range of the embedded RNS, (otherwise unrecoverable errors occur), and so the additions required are also of limited range.

## 3: Hardware Optimization

### 3.1: Computing over $\mathfrak{R}_{17 \times 257}$

Using Fermat primes, we can couple binary-like computation with index calculus, to provide a form of logarithmic arithmetic and reduce the complexity of multiplication, which is a large computational overhead for DSP applications; this has been exploited in a recent commercial RNS chip [4]. Index calculus maps the multiplicative group  $G_{\otimes_p} = \{GF(p) - (0); \otimes_p\}$  to the

additive group  $G_{\oplus_p} = \{GF(p) - (p-1); \oplus_{p-1}\}$ . Since the Fermat prime has the form,

$2^{2^f} + 1$ , then the additive group uses a modulus of  $2^{2^f}$  which can be implemented with direct binary coded arithmetic. Although index calculus has been used both commercially [4], and in research projects [8], the sole use of Fermat primes has not been seriously contemplated,

although some early work was reported on computational simplifications over  $2^n + 1$  rings [9]. A further advantage of using  $2^n + 1$  fields is the ability to code data in a Quadratic Residue form and thus take advantage of QRNS independent channel processing of real and imaginary data components [10][11].

The first 4  $2^n + 1$  primes are Fermat numbers, namely 3, 5, 17, and 257. The 5th, and possibly final, Fermat prime,  $2^{16} + 1 = 65537$ , is too large for the required look-up table mapping operations required between the multiplicative and additive sub-groups. 257 is the best choice, though the replication factor will be reasonably large in order to accommodate a large effective dynamic range for our inner product computation. The embedding of a  $257 \times 17$  RNS system, however, provides ample range with only a small overhead in the modular ALU; the total overhead (including conversion) being much lower than the replication overhead of using 257 alone. The other smaller Fermat primes, do not seem to offer any advantages, either in terms of augmenting the embedded RNS or in replacing 17 in the two modulus RNS.

It is important to note that in a sole use of an RNS, the dynamic range of  $257 \times 17$  is insufficient for practical uses. However, in our case, modulus replication technology is employed to expand this dynamic range.

### 3.2: CRT conversion

Over the cross product ring  $\mathfrak{R}_{17 \times 257}$ , each data sample,  $X$ , is represented as a pair of residues  $(x_{17}, x_{257})$  throughout the inner product calculations. The reverse mapping, via a low hardware complexity MRC form of conversion is shown in eqn. (1).

$$X = x_{257} + 257 \times ((9 \otimes_{17} x_{17}) \oplus_{17} (8 \otimes_{17} x_{257})) \quad (1)$$

### 3.3: Trade-offs

We now consider the trade-offs between the number of replications, the FIR blocklength, and the precision of the calculation.

For our purposes, precision has the following components:

1. The bitlength of the input binary data and the inner product (FIR) coefficient data.
2. The probability of correctness of the resultant polynomial coefficients.

Component 2. is a specific property of the MRRNS based on a non-zero probability that a coefficient may overflow the replicated modulus. In RNS calculations, we do not allow any overflow of the modulus since the results will be quite unpredictable, but a small probability of overflow is acceptable in the MRRNS, since the coefficients that are most likely to overflow are in the centre of the polynomial evaluation. After removing the lower part of the final polynomial result (equivalent to precision reduction in standard binary implementations) the overflow is relegated to the least significant part of the result magnitude. Reasonably large probabilities of overflow, however, will lead to coefficient overflow in the more significant parts of the result magnitude. The exact amount of an acceptable probability of overflow is difficult to bound, but from many signal processing experiments, we find that this probability must be considerably less than 0.5%. Even so, the ability of the MRRNS to accept some overflow provides a more robust arithmetic structure than a pure RNS system.

In Table 1 we show some results on the probability of error of the polynomial coefficients with different combinations of bitlength and indeterminate mapping weight. We also show the hardware complexity (in residue multiply/accumulates MACs). The last two columns give the numbers of MACs (modulo 257 and 17) for both the MRRNS datapath and polynomial mapping overhead.

$X$	bitlength	$d$	$N$	prob. of error	MRRNS MAC's	overhead MAC's
2	10	17	200	0%	$N \times 17$	$27 \times 17$
4	11	9	120	0%	$N \times 9$	$15 \times 9$
4	15	13	120	0%	$N \times 13$	$21 \times 13$
8	10	5	160	0.005%	$N \times 5$	$9 \times 5$
8	13	7	160	0.166%	$N \times 7$	$12 \times 7$
8	16	9	40	0%	$N \times 9$	$15 \times 9$
8	19	11	40	0.001%	$N \times 11$	$18 \times 11$
16	13	5	20	0.027%	$N \times 5$	$9 \times 5$

**Table 1.** Trade-offs between indeterminate weight and bitlength

For our purposes we have chosen a target video rate conversion filter with data and coefficients having a bit-length of 10.  $X=8$  provides zero probability of error with a replication factor of 5.

#### 4: The Fermat Number ALU

The basic function of the ALU is to implement two independent MACs over the fields  $GF(17)$  and  $GF(257)$ :  $C' = (A \otimes_{257 \times 17} B) \oplus_{257 \times 17} C$ .

In our modulus replication system, using  $d=5$ , the arithmetic path is divided into 10 independent paths, 5 for each field; each subpath is composed of  $N$  MACs. Here we investigate two schemes for the finite field MAC, and present a comparison of the two designs below. Both the designs utilized index calculus. For brevity, only the Mod 257 MAC is discussed.

##### 4.4: Scheme 1: full-index domain implementation

We will represent the Fermat Field data as elements of the additive group  $G_{\oplus_{257}}$  together with a 9th bit that represents the missing field element, 256. Our arithmetic unit therefore computes multiplication by Mod 256 addition of these group elements; the 9th bit can be treated as a *NAN* indicator (define '1' as *NAN*) [12]. We now invoke the same 'trick' that is used in [12] to compute both multiplication and addition using  $G_{\oplus_{257}}$  elements (this 'trick' was well-known to users of slide-rules!). We write the IPSP in the following form:

$$c_{n+1} = g^{\gamma_{n+1}} = g^{\gamma_n} \otimes_{257} (g^{(\alpha \oplus \beta \oplus [-\gamma])} + 1) \quad (2)$$

which can now be expressed over  $G_{\oplus}$  as shown in eqn. (2) (note that we have dropped the subscript 256 for the additive operator).

Let us denote the index map of  $X$  as  $I(X)$ , and the inverse map as  $I^{-1}(X)$ . So  $\alpha = I(A); \beta = I(B); \gamma = I(C); \gamma' = I(C')$ . An example of the notation used for NAN is  $\alpha_{NAN}$  which is the special bit for input  $\alpha$ ;  $\alpha_{NAN} = 1$  implies that  $A = 0$ . The following four conditions of eqn. (3) apply to the manipulation of the NAN bit:

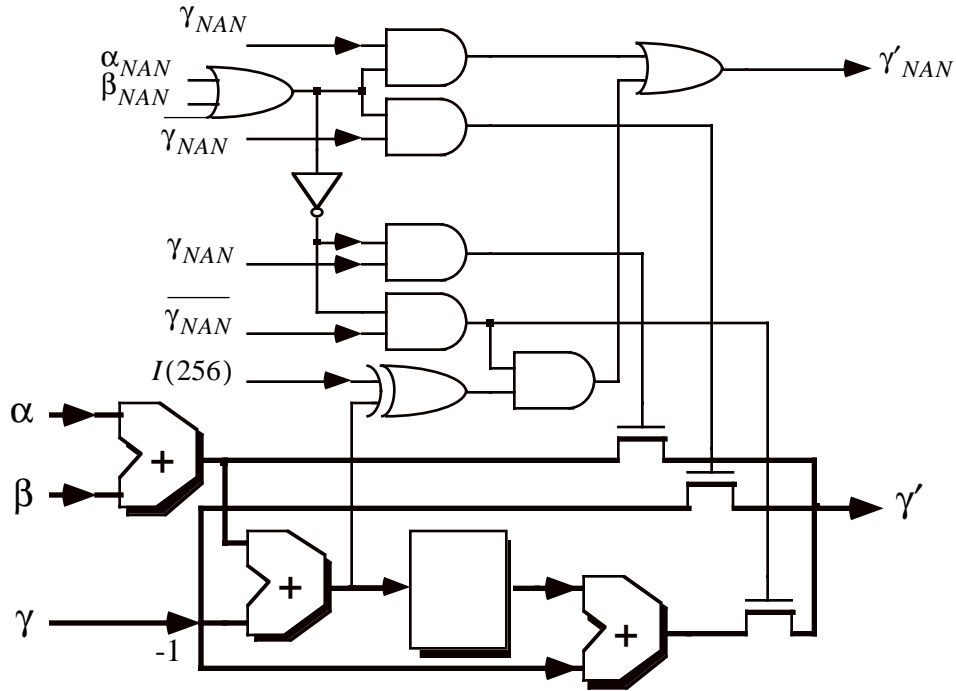
$$\begin{aligned} \gamma_{NAN} = 1 & \quad \begin{cases} \alpha_{NAN} \text{ or } \beta_{NAN} = 1; & \gamma_{NAN} = 1 \\ \alpha_{NAN} \text{ or } \beta_{NAN} = 0; & F = A \otimes_{257} B; \gamma' = \alpha \oplus_{256} \beta \end{cases} \\ \gamma_{NAN} = 0 & \quad \begin{cases} \alpha_{NAN} \text{ or } \beta_{NAN} = 1; & F = C; \quad \gamma' = \gamma \\ \alpha_{NAN} \text{ or } \beta_{NAN} = 0; & C' = C \otimes \{A \otimes B \otimes (C)^{-1} \oplus 1\} \\ & \gamma' = \gamma \oplus I(I^{-1}(\alpha \oplus \beta \oplus \gamma) \oplus 1) \end{cases} \end{aligned} \quad (3)$$

It is also possible for  $F$  modulo 257 to be zero, when eqn. (4) holds true.

$$A \otimes B \otimes C = 257 \Rightarrow A \otimes B \otimes (C)^{-1} \oplus 1 = 257 \Rightarrow A \otimes B \otimes (C)^{-1} = 256 \quad (4)$$

Then  $\alpha \oplus \beta \oplus (-\gamma) = I(256) = 128$  and the result returns to  $\gamma'_{NAN} = 1$ . The full index calculus arithmetic unit for  $MAC_{\text{mod}257}$  is shown in Figure 1. The function:

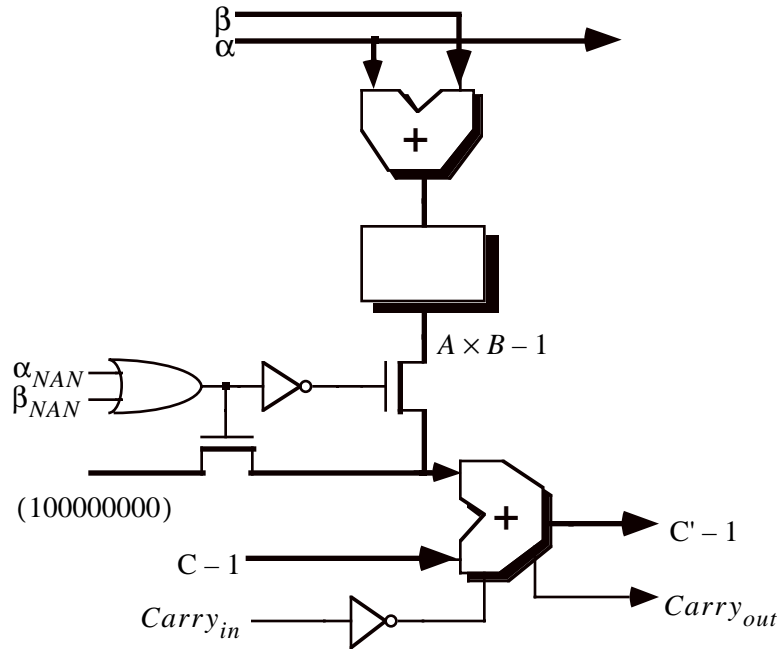
$I(I^{-1}(\alpha \oplus \beta \oplus (-\gamma)) \oplus 1)$  is implemented in the  $256 \times 8$  ROM.



**Figure 1. Modulo (257)-MAC Implementation in full-index domain**

#### 4.5: MAC Implementation in the half-index domain

To reduce the complexities associated with implementing the *NAN* bits, an alternative architecture is suggested in Figure 2.



**Figure 2. Modulo(257)-MAC Implementation in the half-index domain**

This design features an index-domain multiplication ( $A \otimes B$ ) and a normal domain addition ( $\oplus C$ ). The indices of  $A$ ,  $B$  are added in an 8-bit binary adder, as before, and the sum is passed to a ROM of the same size as before  $256 \times 8$ . The ROM stores the operation  $I^{-1}(\alpha \oplus \beta) \oplus (-1) = A \otimes B \oplus (-1)$ . The addition is modulo 257, and we use a familiar diminished-1 adder [13]; the output of the ROM is  $A \otimes B \oplus (-1)$ . The other input,  $C$ , and the output,  $C'$ , are also in the diminished-1 form. Based on the diminished-1's code, the carry from the MSB is inverted and added to the LSB. In this design, the iterative MACs are pipelined, and the addition is passed to the next MAC as the carry-in. Therefore,  $C$  and  $C'$  have a diminished-1 internal format. If  $A$  or  $B$  is zero, then  $A \otimes B = 0$  and the diminished-1 code is 256. A comparison between the two schemes shows that this diminished-1 scheme is more efficient than the pure index calculus technique. Note that this is not the case with more general moduli.

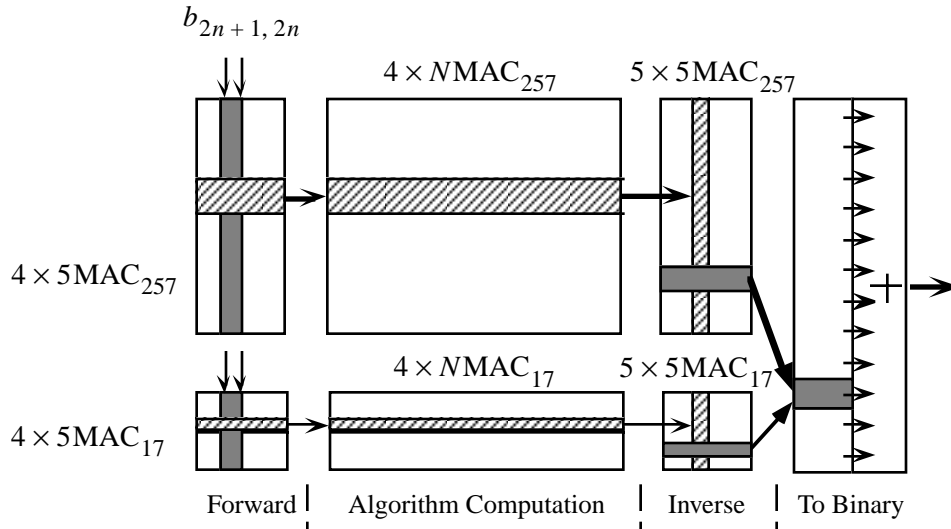
### 5: FIR architecture design

#### 5.6: Complete architecture

The complete architecture of the target FIR filter implemented over  $\mathfrak{R}_{17 \times 257}$  is shown in Figure 3. The structure is basically divided into three parts according to the MRRNS algorithm:

1. MRRNS encoding (Forward polynomial mapping);

2. MRRNS computational paths;
3. MRRNS decoding, which is divided into two steps:
  - 3.1. Inverse polynomial mapping;
  - 3.2. RNS to binary conversion and the final addition.



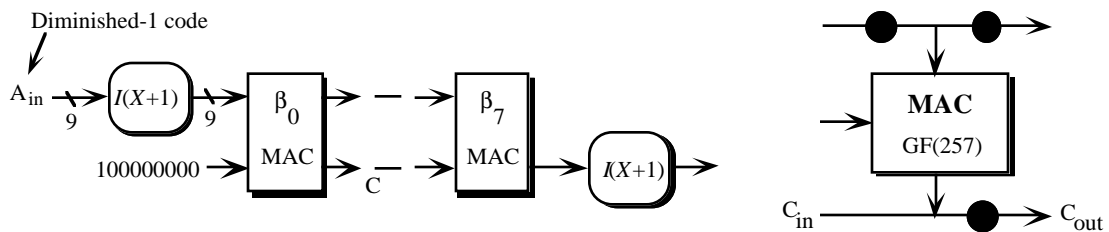
**Figure 3. The complete architecture for the FIR filter**

Both forward and inverse polynomial mappings are matrix-vector multiplications. For each input data sample, it requires fewer than  $2d^2$  integer multiplications for the forward and inverse operations in each implementation ring. We use the MAC units in this pipelined mapping hardware:  $4 \times 5 = 20$  for the forward and  $5 \times 5 = 25$  for the inverse.

After MRRNS encoding, the algorithm is computed over the  $d=5$  subpaths using the MACs for each field. After inverse polynomial mapping, 10 internal results are produced. For each weight of  $8^k$ , the pair of integers,  $(x^{(k)}_{257}, x^{(k)}_{17})$ , are mapped to  $\mathfrak{R}_{17 \times 257}$  before the final  $8^k$ -weight addition is carried out. The hardware realization of this process uses the MRC converter described previously.

### 5.7: Arithmetic path

The FIR systolic array selected for our design is shown in Figure 4 along with the structure for the pipelining of each MAC.



**Figure 4. Systolic structure in one subpath**

We use three pipeline stages by inserting pipeline registers between the index addition, the ROM, and the diminished-1 adder. A pipeline latch is shown by a filled circle.

### 5.8: Forward and inverse polynomial mapping

The forward and inverse mappings are implemented over GF(17) and GF(257), respectively. The forward mapping array is shown in Figure 5 (for GF(257)) and the inverse in. It should be noted that the results from the MAC array are in the internal diminished-1 code (with a carry which should be added at the LSB). Therefore, before going to the next stage, we map to the normal form by adding a “1” and the carry to the 8-bit number using an 8-bit fast adder. Each MAC has a built-in pipeline register for storing the product.

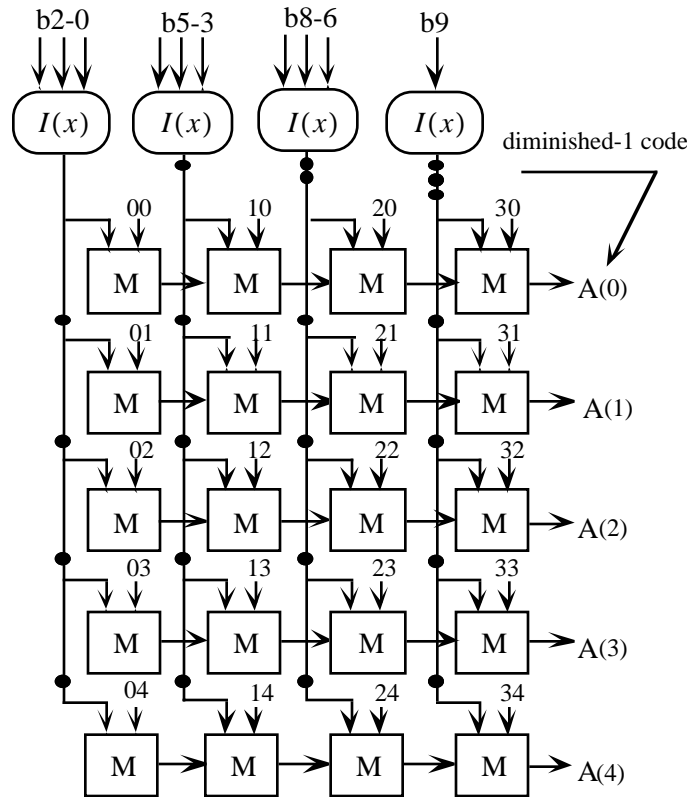


Figure 5. Forward polynomial mapping

### 5.9: RNS to binary system and final addition

We translate the residue representations to mixed radix representations for each polynomial coefficient, i.e. from  $(x_{257}, x_{17})$  to  $(a_{257}, a_{17})$ .

Using the MRC mapping we obtain eqn. (5).

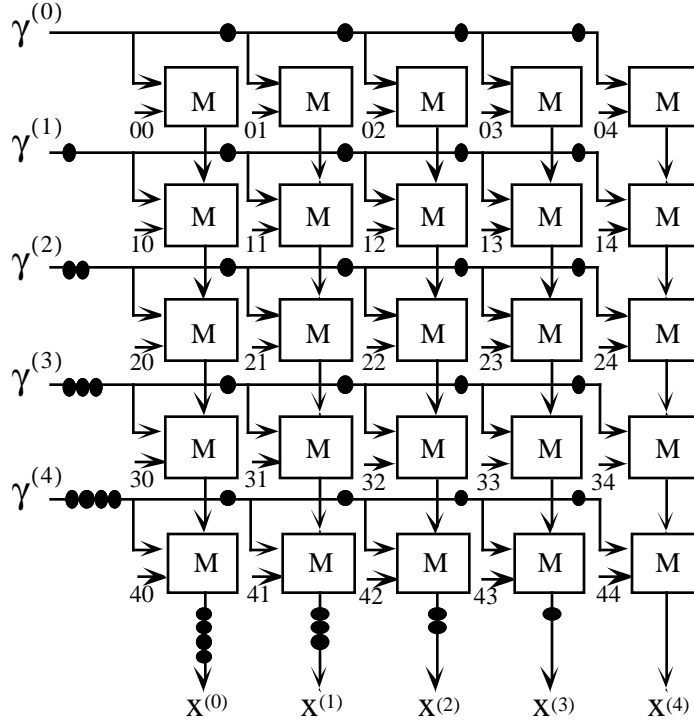
$$X = a_{257} + 257 \times a_{17} \quad (5)$$

where,

$$a_{257} = x_{257} \quad (6)$$

$$a_{17} = 8 \otimes_{17}(x_{257} \oplus_{17}(17 - x_{17})) = 8 \oplus_{17}(x_{257} \oplus_{17}P) \quad (7)$$

We calculate  $P = 17 - x_{17}$  by a reduced ROM [14], and  $x_{257} \oplus_{17}P$  using a fast adder; the result (Q) is a 9-bit number ( $q_8q_7q_6\dots q_1q_0$ ).



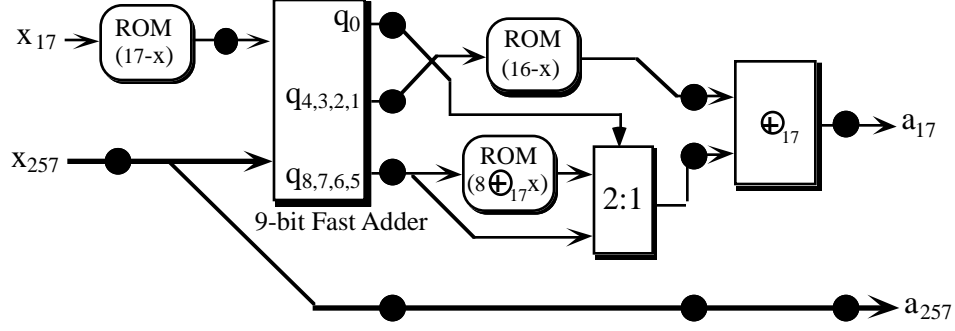
**Figure 6. Inverse polynomial mapping**

We compute  $8 \otimes_{17}Q$  by a 3-position binary left shift followed by a mod 17 reduction. The reduction of the 12-bit shifted result uses the substitutions  $|2^4|_{17} = -1$ ;  $2^4 \otimes_{17}2^4 = 1$ .

Dividing this 12-bit result into three 4-bit sections: A( $q_8q_7q_6q_5$ ), B( $q_4q_3q_2q_1$ ) and C( $q_0000$ ), then:

$$a_{17} = ((A \oplus_{17}C) \oplus_{17}(16 - B) \oplus_{17}1) \quad (8)$$

The complete pipelined mixed radix conversion is shown in Figure 7.



**Figure 7. Mixed radix representation conversion**

The binary results are determined for each polynomial coefficient before the final addition with different weights of  $8^k$  is performed. In our design, this addition is merged with the final  $8^k$ -weighted addition in one CSA array. Since,  $X = a_{257} + 257 \times a_{17} = a_{257} + (2^8 + 1) \times a_{17}$  the final  $8^k$ -weighted addition is as shown in eqn. (9).

$$X_{out} = X^{(4)} \times 8^4 + X^{(3)} \times 8^3 + X^{(2)} \times 8^2 + X^{(1)} \times 8 + X^{(0)} \quad (9)$$

Merging with the mixed radix addition, produces eqn. (10):

$$X_{out} = a_{17}^{(4)} \times 2^{12} + a_{17}^{(3)} \times 2^9 + a_{17}^{(2)} \times 2^6 + a_{17}^{(1)} \times 2^3 + a_{17}^{(0)} \quad (10)$$

It should be noted that,  $a_{257}^{(k)}$  has 9bits and  $a_{17}^{(k)}$  has 5 bits. The bit-map of this addition array is shown in Figure 8. This array is quite similar to the implementation of a parallel multiplier, and so we may compress the partial products, using a CSA array or other parallel counter, and sum the final two partial products with a fast adder. Pipeline registers can be inserted into the process for a higher throughput.

## 6: Simulation Results

Early simulations have been conducted into the design of the GF(257) MAC. The design has been developed to provide approximately equal delay paths through each of the three pipeline stages. This allows us to reduce area and power of the inherently faster paths. We have used a non-precharged ROM structure (that has low power dissipation) and Manchester carry-chain adders using simple static logic. The worst case conditions occur when the input carry travels through the entire Manchester chain of both input and output adders, and the ROM bit-lines completely charge and discharge. The delay measurements are shown in Table 2.

Results taken from SPECTRE simulations are shown in Figure 9. The equalization of the individual stages is clearly in evidence. The ability to pipeline the structure for high rates (in excess of 200MHz) is quite possible. An alternative single pipelined, MAC, will allow throughput rates of at least 50MHz, which will allow lower *Power.Area* implementations for a variety of video-rate conversion filters.

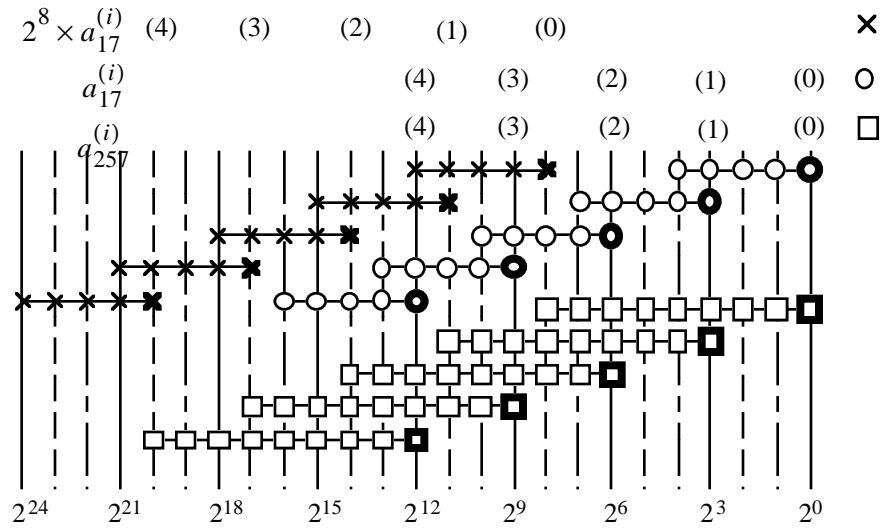


Figure 8. Bit-map of the final addition array

Carry Direction	Input Adder	ROM	Output Adder	Total
Up	3.1ns	4.8ns	3.9ns	11.7ns
Down	3.7ns	3.9ns	3.7ns	11.3ns

Table 2. Worst case delays in the Fermat ALU (Mod 257)

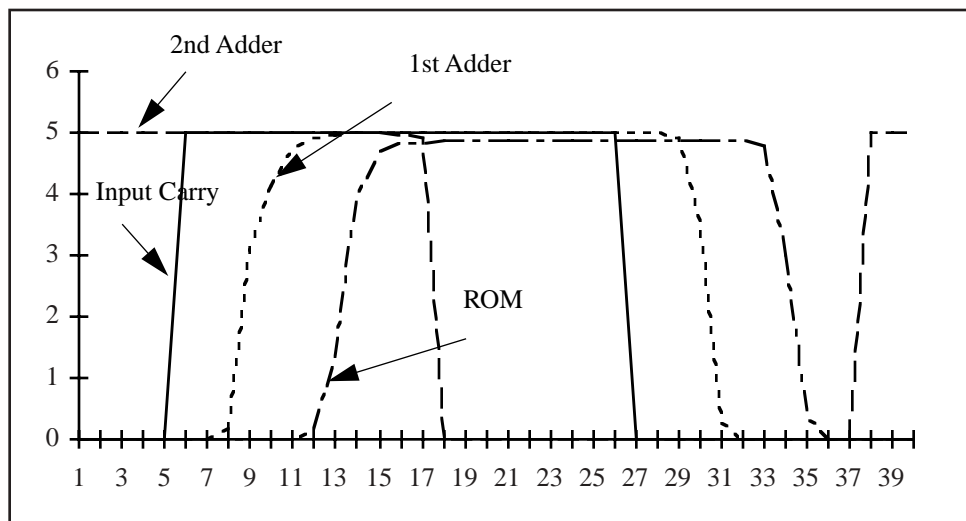


Figure 9. SPECTRE results for worst case carry-in conditions

## 7: Conclusions

In this paper we have demonstrated techniques and architectures to compute inner products using direct product rings defined from Fermat Number fields. We have isolated two specific fields (GF(17) and GF(257)) as having ideal properties for efficient implementation of residue computations. The fields are replicated using the MRRNS polynomial mapping technique, and details are given for both the replicated Fermat ALU and the array architecture. We also justify the use of a half-index calculus over a full-index calculus Fermat ALU, and present a new conversion strategy based on a parallel adder structure.

## 8: References

- [1] G.A. Jullien. "Architectures and Building Blocks for Data Stream DSP Processors." Invited Chapter in VLSI Design Methodologies for Digital Signal Processing Architectures, Ch. 9, pp. 319-364, Kluwer Academic Publishers, 1994.
- [2] Jullien, G.A., Bizzan, S. and Wigley, N.M. "Using Redundant Finite Rings for Fault Tolerant Signal Processors." Proceedings SPIE 1994., San Diego, CA.
- [3] G.A.Jullien, M.Taheri, S.Bandyopadhyay, W.C. Miller, "A Low-Overhead Scheme for Testing a Bit Level Finite Ring Systolic Array." J. VLSI Sig. Proc., Vol 2.3, pp. 131-138, 1990.
- [4] Barraclough, S.R., Sotheran, M., Burgin, K., Wise, A.P., Vadher, A., Robbins, W.P. and Forsythe, R.M. "The Design and Implementation of the IMS A110 Image and Signal Processor." IEEE Custom Integrated Circuits Conf., pp. 24.5.1-24.5.4, 1989.
- [5] Soderstrand, M.A., Jenkins, W.K., Jullien, G.A. and Taylor, F.J. "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing." IEEE Press. NY., 1986.
- [6] Wigley, N.M. and Jullien, G.A. "On Modulus Replication for Residue Arithmetic Computations of Complex Inner Products." IEEE Trans. Comp. 39, pp. 1065-1076., 1990.
- [7] Wigley, N.M. and Jullien, G.A. "Large Dynamic Range Computations Over Small Finite Rings." IEEE Trans. Comp. Vol. 43, No. 1, pp. 76-86, 1994.
- [8] J.D.Mellott, J.C.Smith and F.J.Taylor, "The Gauss Machine: A Galois-Enhanced Quadratic Residue Number System Systolic Array", Proceedings of the 11th IEEE International Symposium on Computer Arithmetic, pp.156-162, 1993.
- [9] D.P Agrawal and T.R.N. Rao, "Modulo  $2^n + 1$  Arithmetic Logic." IEE J. Electronic Circuits and Systems, Vol. 2, pp.186-188, 1978.
- [10] Jenkins, W.K. and Krogmier, J.J. "Error Detection and Correction in Quadratic Residue Number Systems." 26th Midwest Symposium on CAS, Puebla, Mexico. pp. 408-411, 1983.
- [11] G.A.Jullien, R.Krishnan and W.C.Miller, "Complex Digital Signal Processing Over Finite Rings", IEEE Tran. on Circuits and Systems, CAS-34, No.4, pp.365-377, 1987.
- [12] G. Zelniker and F.J. Taylor, "A Reduced-Complexity Finite Field ALU", IEEE Trans. on CAS, Vol.38, No.12, pp.1571-1573, 1991.
- [13] L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform." IEEE Trans. Acoustics, Speech and Signal Processing, Vol. ASSP-24, No. 5., Oct, 1976.
- [14] G.A.Jullien, W.C.Miller, R.Gronin, L.Del Pup, S.Bizzan, D.Zhang, "Dynamic Computational Blocks for Bit-Level Systolic Arrays", IEEE Journal of Solid-State Circuits, Vol.29, No.1, pp.14-22, 1994.