



VLSI Research Group

University of Windsor

High Throughput VLSI DSP Using Replicated Finite Rings

Journal of VLSI Signal Processing

Graham A. Jullien, Wenzhe Luo and Neil M. Wigley

Abstract

This paper discusses recently introduced strategies in implementing DSP systems using residue replication; in particular we highlight current work underway in the VLSI Research Group, at the University of Windsor, in the area of constructing high throughput DSP systems on silicon. The paper first briefly reviews the theory and mapping techniques, associated with general residue and residue replication systems, then discusses the detailed VLSI design of an efficient general coefficient inner product step processor using Fermat Primes.

High Throughput VLSI DSP Using Replicated Finite Rings

Graham A. Jullien, Wenzhe Luo and Neil M. Wigley

1 INTRODUCTION

Number Theory is often treated as the last vestige of pure mathematical pursuit. Some results, of course, are usefully applied [1]; signal processing examples are in the areas of cryptography, error correction/fault tolerance and in DSP arithmetic implementation. This paper is concerned with the latter application area. Residue Arithmetic for computer systems has been studied for at least 4 decades, but has never been widely applied to commercial systems. There are good reasons for this, mostly based on the awkwardness of operations that do not exist under closure. Some limited success has been reported in the area of integer arithmetic DSP systems, where the algebraic tricks associated with computing over finite rings and fields, can be used to great effect. Examples are the use of index calculus [2] (finite field ‘exact’ logarithms) and quadratic residue rings for reduced and separable complex arithmetic [3].

We first emphasize the importance of the structure that residue replication imparts to DSP architectures. As with classical residue computations, one-dimensional algorithms, for example, stay one-dimensional at the digit level, thus providing a more benign data path and clock routing environment. The ability to test integrated circuits is dependent on the circuit connectivity, and the isolation of the full computational data path into smaller data paths increases the testability of the chip; for example, some algebraic properties associated with finite ring computations allow almost trivial tests for full fault coverage [9]. The use of replication theory provides two major advantages over classical residue arithmetic computations:

1. Replication of a single modulus ALU; this allows more flexible multiplexing arrangements (no programmable moduli ALUs).
2. The ability to choose a single modulus for its algebraic properties, rather than a set of moduli (with weaker properties).

In this paper we exploit the second property directly, and imply the first. In particular we will consider the single replicated modulus as composite, with computations being performed over sub-rings (i.e. an embedded RNS).

2 COMPUTING OVER FINITE RINGS

In residue systems [4] we deal with rings, or fields, that are used for the actual implementation and rings that are isomorphic to direct products of implementation rings or extensions of them. A given digital signal processing algorithm is mapped from real or complex integer arithmetic to the implementation rings, the computation is carried out there, and the result is then mapped back to obtain the final answer.

We denote by $\mathfrak{R}(m)$ the ring of integers modulo m :

$$\mathfrak{R}(m) = \{S: \oplus_m, \otimes_m\}; \quad S = \{0, 1, \dots, m-1\} \quad (1)$$

where we use the notation $a \oplus_m b$ and $a \otimes_m b$ to imply the residue reduction of a and b modulo m within addition and multiplication. We can extend the notion of addition and multiplication from the elements of S to all of the integers. If \mathfrak{R}_1 and \mathfrak{R}_2 are any two rings, defined with moduli (m_1, m_2) , then we can define the cross-product ring, $\mathfrak{R}_1 \times \mathfrak{R}_2$ as the set of pairs

$(s_1, s_2) \in S_1 \times S_2$, with addition and multiplication defined component wise, i.e. by eqn. (2).

$$\begin{aligned} (a_1, a_2) \oplus_{\mathfrak{R}_1 \times \mathfrak{R}_2} (b_1, b_2) &= (a_1 \oplus_{m_1} b_1, a_2 \oplus_{m_2} b_2) \\ (a_1, a_2) \otimes_{\mathfrak{R}_1 \times \mathfrak{R}_2} (b_1, b_2) &= (a_1 \otimes_{m_1} b_1, a_2 \otimes_{m_2} b_2) \end{aligned} \quad (2)$$

The isomorphism between $\mathfrak{R}(M)$ and the direct product of the $\{\mathfrak{R}(m_k)\}$ means that calculations over $\mathfrak{R}(M)$ can be effectively carried out over each $\mathfrak{R}(m_k)$, independently and in parallel. A final mapping to $\mathfrak{R}(M)$ is performed at the end of a chain of calculations. We have therefore broken down a calculation set in a large dynamic range, M , to several calculations over smaller dynamic ranges. This is the main advantage of using the RNS over a conventional weighted value numbering system (e.g. binary). The advantages of independent (not just parallel) computation should not be underestimated, particularly when very large special purpose high throughput processing arrays are to be built.

The final mapping is found from the CRT:

$$X = \sum_{k=1}^L \oplus \left\{ \hat{m}_k \otimes_M [x_k \otimes_{m_k} (\hat{m}_k)^{-1}] \right\} \quad (3)$$

with $\hat{m}_k = M/m_k$ and $()^{-1}$ the multiplicative inverse operator. We have also used the notation

$\sum_{k=1}^L \oplus$ to indicate summation over the ring \mathfrak{R}_M .

2.1 Polynomial rings and quotient rings

We let $\mathfrak{R}[X]$ denote the ring of polynomials in the indeterminate, X , as shown in eqn. (4).

$$X: \mathfrak{R}[X] = \left\{ \sum_{k=0}^n a_k X^k : (a_k \in \mathfrak{R}, n \leq 0) \right\} \quad (4)$$

We define the ring $\mathfrak{R}[X_1, X_2, \dots, X_S]$ to be the ring of multivariate polynomials in the indeterminates. We use polynomial rings, where the base ring \mathfrak{R} , is a modular ring, $\mathfrak{R}[M]$, and we write $\mathfrak{R}_M[X_1, X_2, \dots, X_S]$ in place of $\mathfrak{R}(M)[X_1, X_2, \dots, X_S]$. For a given polynomial

$g(X) \in \mathfrak{R}(X)$ we consider the set $(g(X))$ of all (polynomial) multiples of $g(X)$. This set is the ‘ideal’ generated by the polynomial $g(X)$ in the ring $\mathfrak{R}[X]$. The quotient ring

$\mathfrak{R}[X]/g(X)$ is then defined to consist of all elements of the form $f(X) + (g(X))$, with

$f(X) \in \mathfrak{R}[X]$. We now let indeterminates represent various powers of 2 in the binary representation of the data samples [5][6]. This allows the data to be expressed as polynomials with small coefficients. These coefficients are then mapped to a direct product ring consisting of many copies of Z_M (the ring of integers modulo m) as factors. This has been referred to as a Modulus Replication RNS (MRRNS) [5].

In order to be able to perform useful computations, the modulus, M , has to be able to contain the coefficients of result polynomials. Multiplication will be the major problem in coefficient growth, and we assume that the algorithm is arranged so that only single cascades of multipliers are used prior to the application of mapping circuitry. We can further decompose M , to allow the use of very small rings, by the application of a RNS. The mathematical derivations are somewhat tedious, and the reader is referred to a more complete description in [6].

The same features associated with computation over direct product rings are present with this technique; we may even embed a standard RNS within the mapping structure [6].

2.2 Modulus Replication Mapping Algorithm

The main steps in the preferred 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 [5]. 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 APPLICATIONS TO DSP SYSTEMS

Purely feedforward algorithms provide the niche area for efficient residue implementations. In terms of applications, this covers a large subset of algorithms commonly used in DSP systems. Obvious examples are DFT, DCT, FIR filters and general matrix operations. There are also non-obvious uses, such as digital waveform synthesizers [7]. Systems built either entirely, or partially, with feedforward building blocks, are potential candidates for residue implementation approaches. Recent work on minimizing operations that do not exist under closure has revealed strategies that also preserve the arithmetic efficiency of fast algorithms [8]. An example of a 15 sample Discrete Cosine Transform that supports such a single multiplication cascade is shown in Figure 1 a). The strip in the centre contains the multiplications, all other operations being additions or subtractions.

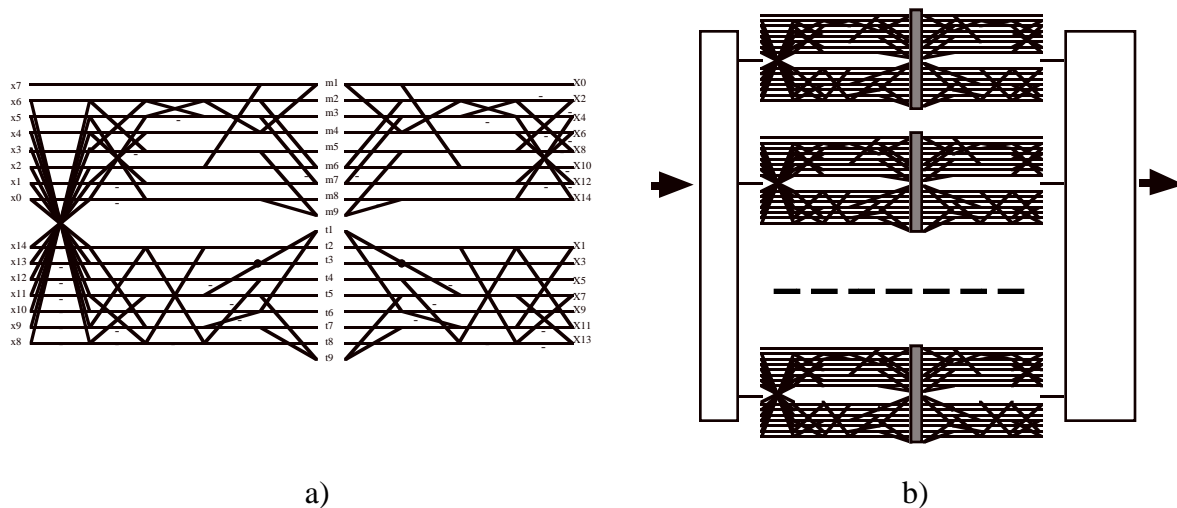


Figure 1. Mapping a 15-point Fast DCT to a Residue System

Figure 1 b) shows how this algorithm will be mapped to a set of independent computations over cross product rings. The algorithm is simply replicated with each computation being performed over different rings (as in the case of an RNS map), or replications of the same ring (as in the case of a MRRNS map). In the latter case we may embed an RNS system in order to compute over large rings via the use of direct products of smaller rings. Using this approach we have been able to show the computation of the equivalent of over 40-bits of dynamic range with the moduli: $\{3,5,7\}$ [6].

A major stumbling block at the circuit level is the hardware required to implement residue computations. Traditional techniques have concentrated on either the use of stored tables, or the use of ring moduli that are very close to a power of 2. The use of index calculus has also proved to be a powerful tool in efficient implementation of general coefficient inner product type computations. We will use all of these special *tricks* in selecting ring moduli for computing over replicated finite rings.

4 SPECIAL ARCHITECTURES FOR REPLICATION APPLICATIONS

If we consider only feedforward algorithms, then we need not be concerned with pipelining issues with regard to dynamic range scaling around recursive feedback loops. Our algorithms can usually be implemented with some form of a finite ring ALU. This can take on a variety of forms depending on the algorithm. A very important building block is the inner product step processor

(IPSP), and this has been discussed many times over the past decade or so. It does, however, provide a nucleation centre for discussing the computational issues. We start the discussion by a quick review of index calculus over Galois Fields.

4.1 Prime Moduli

If we restrict the general modulus to be prime (still ‘fairly’ general!) then the use of index calculus is particularly interesting (and quite well explored!), and is the trick used in a commercial FIR filter chip from INMOS [2]. There is also evidence from the academic research community that this represents a viable decomposition strategy [3]. Some early work from our group details the construction of prime moduli multipliers using only n-bit ROMs [10]. In the case of [3], all of the computations are performed using index calculus; multiplication is clearly easy, and we can use the old ‘slide-rule’ trick to perform addition with a log calculator [11].

4.1.1 Index Calculus

We wish to compute over the Galois Field $GF(p) = \{S: \oplus_p, \otimes_p\}$ where p is prime. We invoke

the mapping, $a \otimes_p b = g^{(\alpha \oplus_{p-1} \beta)}$ between the additive group,

$$G_{\oplus} = \{(\{S\} - (p-1)): \oplus_{p-1}\}, \text{ and the multiplicative group,}$$

$$(G_{\otimes} = \{(\{S\} - (0)): \otimes_p\}) \text{ , where } g \text{ is a generator of } G_{\otimes} \text{ . It is apparent that in order to per-}$$

form multiplication over the field (except with the element 0), all one has to do is to map to the additive group, perform addition Mod $p-1$ and then map back. To perform addition, we simply factor out the addend and store all combinations of the remaining factor in a ROM.

If we consider the IPSP of eqn. (5):

$$\begin{aligned} c_{out} &= (a_{in} \otimes_p b_{in}) \oplus_p c_{in} \\ a_{out} &= a_{in} \end{aligned} \tag{5}$$

then we can use index calculus, as in eqn. (6) (we refer to this as the *slide-rule trick*.)

$$\gamma_{out} = \gamma_{in} \oplus_{p-1} \Psi(g^{(\alpha_{in} \oplus_{p-1} \beta_{in} \oplus_{p-1} [-\gamma_{in}])} \oplus_p 1) \quad (6)$$

A ROM stores the forward mapping function, Ψ , and a mod $p-1$ adder/subtractor is used to form the n -bit address input. Addition over general moduli has been explored by several authors (a compendium can be found in [4],) more recent work from our group can be found in [12].

4.2 Computing Over Special Ring Moduli

Much work has been reported on circuitry that can be used for special moduli [4]. The selection of moduli close to a power of 2 is of particular interest, and moduli sets, such as

$\{2^n - 1, 2^n, 2^n + 1\}$, have attracted much interest over the past 2 decades. With the introduction

of the MRRNS we now have the ideal strategy of using a single, specially selected, ring modulus, and replicating computations over this ring to obtain the required dynamic range. Based on the

Galois Field IPSP, we can restrict our prime modulus to the form $2^{2^t} + 1$ (a Fermat number which is prime for at least the first 5). The advantage of using a prime of this form is that the index calculus can be performed using 2^{2^t} -bit binary adders. Probably the most perfect numbers for this approach are 257 and 17. We can restrict the calculations to Mod 257, but the ease with which we can embed a $\{17,257\}$ RNS within a MRRNS mapping is too good to pass up.

In the following sections we present some results on the use of this special ring in a replication architecture to implement FIR filters (a specific example of inner product computations). We will briefly review the theoretical design results, since more details are available in a recent conference publication [14]. Following this brief review we will detail the design of a *Fermat ALU*.

4.2.1 Computing over $\mathfrak{R}_{17 \times 257}$

The first four $2^n + 1$ odd primes are also Fermat numbers, namely 3, 5, 17, and 257. The 5th Fermat number is also prime, $2^{16} + 1 = 65537$, but 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 257×17 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 257×17 is insufficient for practical uses. However, in our case, modulus replication technology is employed to expand this dynamic range.

4.2.2 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. (7).

$$X = x_{257} + 257 \times ((9 \otimes_{17} x_{17}) \oplus_{17} (8 \otimes_{17} x_{257})) \quad (7)$$

4.3 Choice of Parameters

Modulus replication is an integer arithmetic system, and it is important to look at the trade-offs between the number of replications, the FIR blocklength, and the precision of the calculation.

A specific finite precision effect of the MRRNS is the 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 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%	$N \times 17$	27×17
4	11	9	120	0%	$N \times 9$	15×9
4	15	13	120	0%	$N \times 13$	21×13
8	10	5	160	0.005%	$N \times 5$	9×5
8	13	7	160	0.166%	$N \times 7$	12×7
8	16	9	40	0%	$N \times 9$	15×9
8	19	11	40	0.001%	$N \times 11$	18×11
16	13	5	20	0.027%	$N \times 5$	9×5

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 0.005% probability of error with a replication factor of 5.

4.4 The Fermat Number ALU

The basic function of the ALU is to implement two independent MACs over the fields GF(17) and GF(257).

$$C' = (A \otimes_{257 \times 17} B) \oplus_{257 \times 17} C. \quad (8)$$

We have explored two different architectures for the Fermat Number MAC. The first technique utilizes the *slide-rule trick* to maintain all data in the index domain; i.e., we apply eqn. (6) for the multiplier accumulator. The second technique borrows from some early work on Fermat Number Theoretic Transforms, using a diminished-ones translation technique to perform addition over each Galois Field with index calculus restricted to implementing multiplication only. Our study reveals that the diminished-ones representation provides a lower hardware solution (more architectural details can be found in [14],) and we limit our discussion to this second architecture. For brevity, only the Mod 257 MAC is discussed.

4.4.1 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, and the sum is passed to a ROM of size 256×8 . The ROM stores the operation $I^{-1}(\alpha \oplus \beta) \oplus (-1) = A \otimes B \oplus (-1)$.

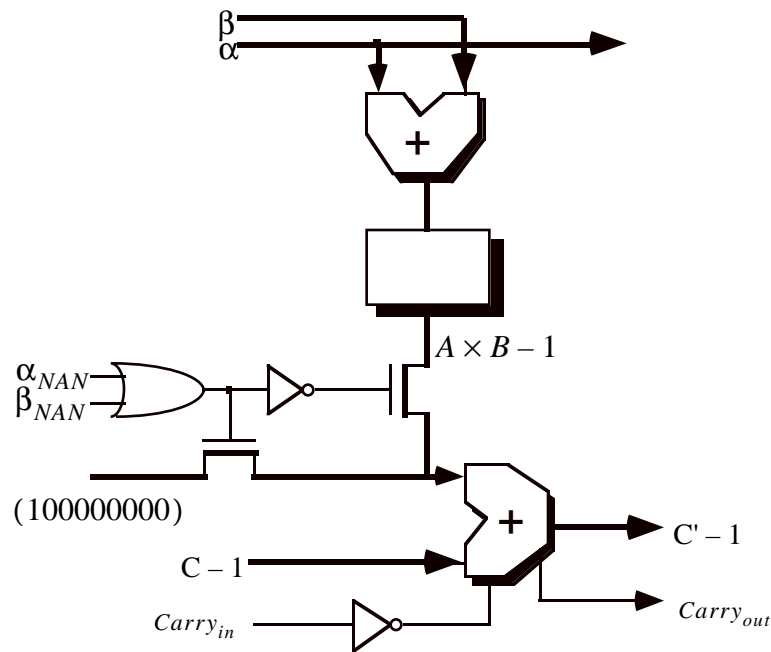


Figure 2. Modulo(257)-MAC Implementation in the half-index domain

The structure of the MAC is shown in Figure 2. The addition is modulo 257 (over the Galois Field), and we use a diminished-ones adder [15]; the output of the ROM is $A \otimes B \oplus (-1)$. The other input, C , and the output, C' , are also in the diminished-one form. Based on the diminished-ones 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. A comparison between the two schemes shows that this diminished-one scheme is more efficient than the pure index calculus technique. Note that this is not the case with more general moduli, and so this represents an interesting result.

4.5 FIR architecture design

4.5.1 Complete architecture

The complete architecture of the target FIR filter is shown in Figure 3. The structure is basically divided into three parts:

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.

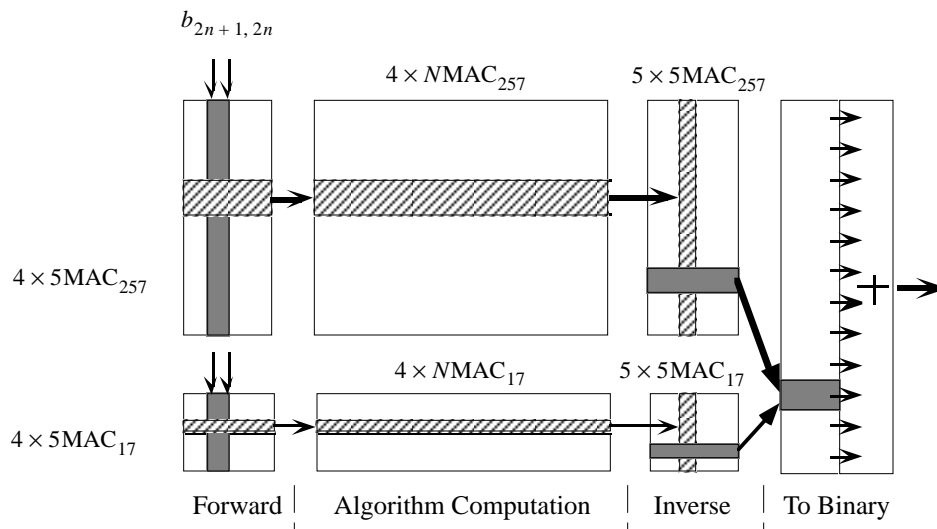


Figure 3. The complete architecture for the FIR filter

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. This is interesting, since we are in a position to measure the hardware overhead in terms of equivalent filter coefficient hardware. In this case the overhead is equivalent to 9 extra coefficients; for a 150 tap filter this amounts to only a 6% overhead, considerably smaller than most RNS designs.

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 $\mathbb{R}_{17 \times 257}$, before the final 8^k -weight addition is carried out. The hardware realization of this process uses the MRC converter described previously.

4.5.2 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. It is to be noted that any of the numerous systolic arrays for convolution can be used for each subpath.

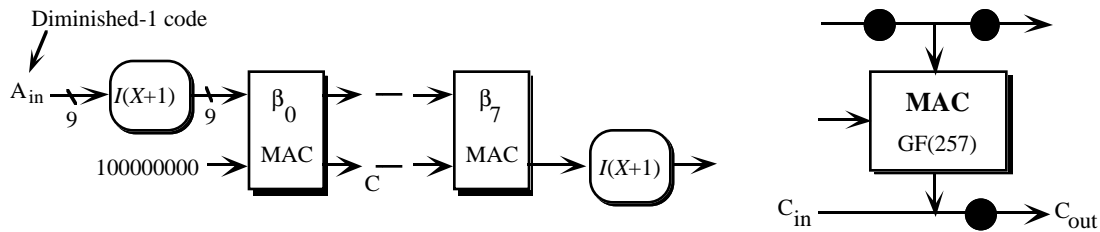


Figure 4. Systolic structure in one subpath

4.5.3 Forward and inverse polynomial mapping

The forward and inverse mappings are implemented over $GF(17)$ and $GF(257)$, respectively [14]. As an example of the regularity of MRRNS mapping, the inverse array is shown in Figure 5. The computational elements are, in fact, the same multiplier accumulator that is used for the subpath computations. This is to be contrasted to an RNS system, where very irregular architectures are used to perform inverse mapping. 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.

4.5.4 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}) .

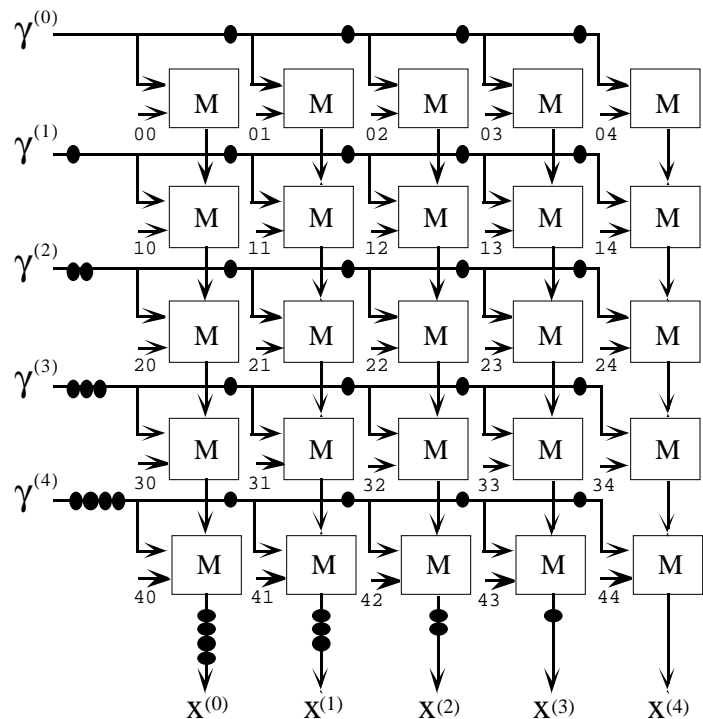


Figure 5. Inverse polynomial mapping

Here we invoke an MRC mapping for the RNS conversion, as shown in Figure 6 [14]. We have included this figure as representative of the irregular form of RNS conversion, as discussed earlier. In this case the conversion to $\mathfrak{R}_{257 \times 17}$ is not a major concern in the overall architecture.

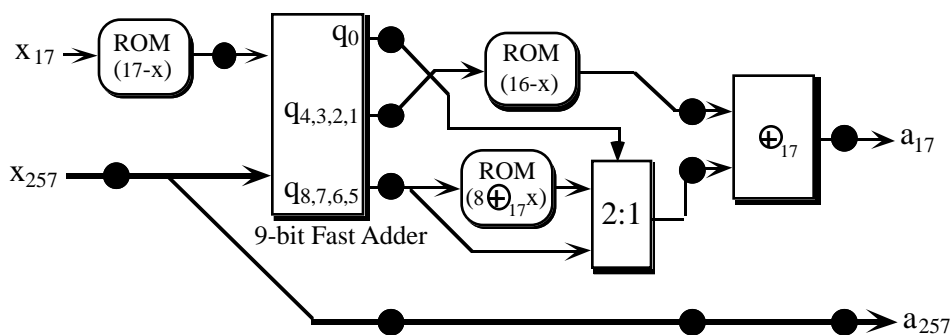


Figure 6. Mixed radix representation conversion

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, as shown in Figure 7 [14]. 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.

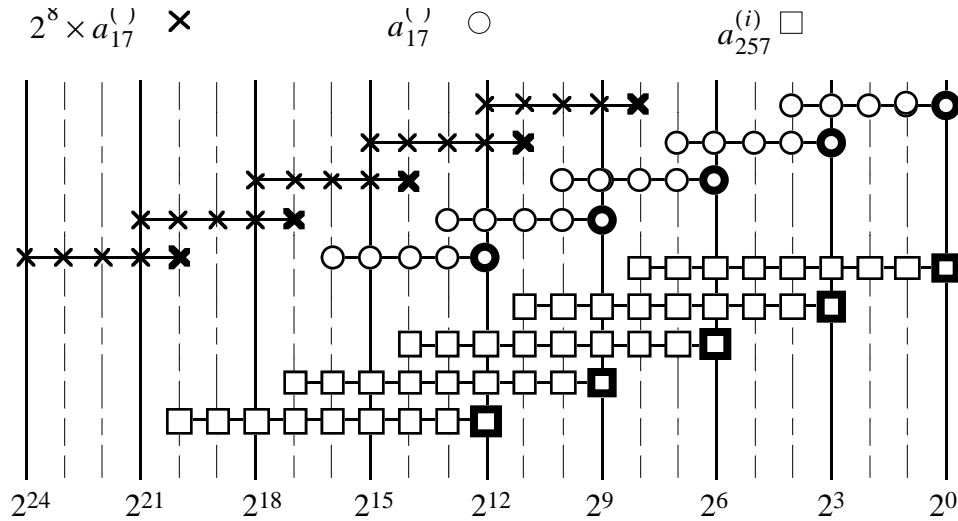


Figure 7. Bit-map of the final addition array

5 VLSI CIRCUIT DETAILS

The efficacy of the replication mapping can only really be explored at the level of full custom VLSI circuitry. For this paper we will restrict ourselves to the implementation of the basic Fermat ALU (multiplier/accumulator), and provide some preliminary comparisons with a binary MAC. Our design style is based on dynamic logic and TSPC pipeline latches. There are some specific design details that we introduce below.

5.1 Overview

The circuit design techniques of our implementation of the Fermat ALU are embodied in the following aspects:

1. Design of a dynamic adder using modular EMODL circuit techniques.
2. Design of a dynamic ROM with a domino-style sense amplifier.
3. Design of a TSPC dynamic DFF.

We will discuss each of these separately.

5.2 Design of the dynamic adder with EMODL and modularity.

EMODL (Enhanced Multiple-Output Domino Logic) was first introduced in [16] with application to the design of efficient, fast, modular elements for carry-lookahead adders. The pseudo complement adder tree, proposed in [16], is built to a maximum height of 4 in this design. We are able to cascade these trees to obtain small bit-length (<16) adders with low power dissipation, high speed, and design modularity. The very fast but irregular architecture reported in [16] (2.7ns critical path for a 32-bit adder) is replaced by a more modular, lower power construction that attains a sufficiently fast evaluation time to enable a 3-stage pipeline to be efficiently utilized for the complete Fermat ALU. The structure of a single level in the adder tree is shown in Figure 8.

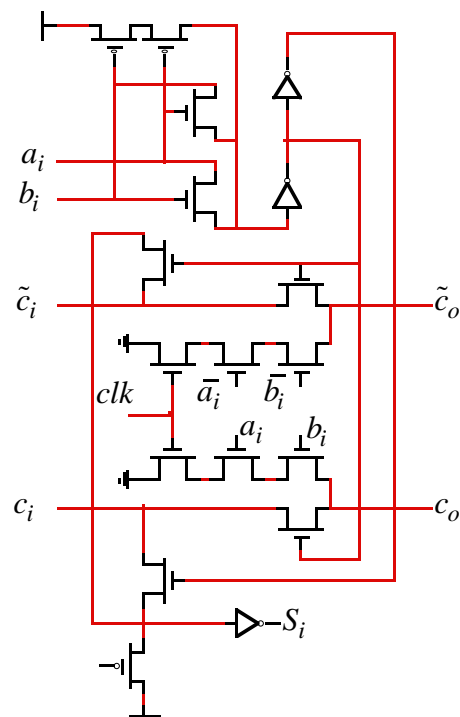


Figure 8. A single-bit level in the EMODL tree.

In Figure 8, a_i , b_i are the inputs of the bit position, S_i is the sum of the output; c_i and c_o constitute the carry chain. \bar{a}_i is a normal complement and \tilde{c}_i is a *pseudo-complement* [16]. The use of pseudo-complements allows symmetrical domino circuits to be built, with minimum height [16]. The XOR/XNOR gate at the top of the schematic of Figure 8 is based on a new design proposed in [17].

The use of EMODL eliminates the p transistors associated with separate domino stages to provide an efficient logic subfunction evaluation. The modularity associated with *growing* the adder bit-length is shown in Figure 9 for a 4-bit adder module; note the multiple output sums in the single tree and the pseudo-complement carries generated by the cascaded subfunctions.

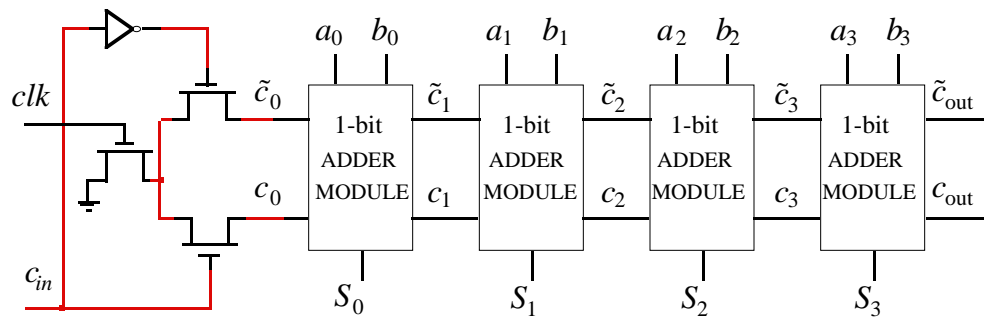


Figure 9. A 4-bit EMODL adder tree

For multiple sections of 4-bits (e.g. the GF(257) MAC uses 8-bit adders) a connecting component (*X-connector*) is inserted between the EMODL trees to accelerate the discharge during the evaluation phase. The X-connector is functionally similar to carry regenerating buffers [18] which were used in static Manchester adders; in our case, where dual carry chains are used and the circuits are dynamic, two chains of carries need to be restored, and the ground clock switch and domino inverters are used to generate the carry signals dynamically. The X-connectors decrease the loading capacitances for the propagating carries (and their pseudo-complements), and so greatly reduce the worst-case delay. Simulation results of an 8-bit adder (made up of the two 4-bit adder trees) show a reduction in delay from 4.9ns to 3.3ns when the X-connector is inserted. An X-connector is shown in Figure 10, with a 4n tree cascade shown in Figure 11.

Post-layout simulation shows that, using a target 1.5 μ DLM CMOS process (Mitel 1.5 μ available to our laboratory through the Canadian Microelectronics Corporation), the evaluation delay of an 8-bit adder constructed using two 4-bit EMODL trees is about 3.3ns. The area consumed by a 1-bit adder module is 4251 μm^2 , with the 1.5 μm design rules. This is a very small area compared with the average bit area consumption of modern adder designs [19]. In [19], the area consumption per bit is about $3.22 \times 10^4 \mu\text{m}^2$ with 1 μm CMOS design rules.

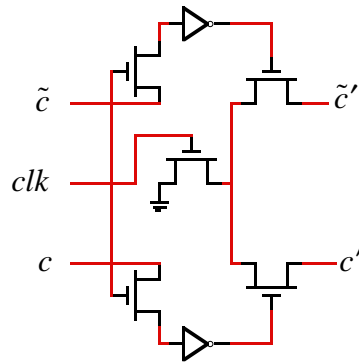
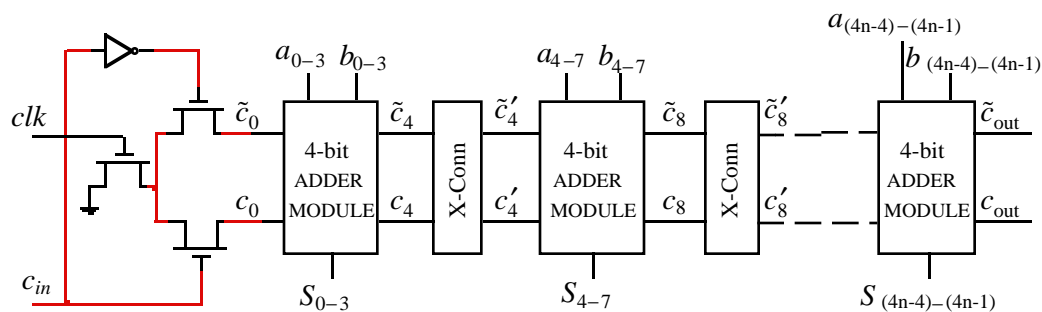


Figure 10. Accelerating X-connector

Figure 11. $4n$ Adder Tree cascade with X-connectors

5.3 Dynamic ROM Design

In the Fermat ALU, the ROM is a main component of area consumption and power dissipation, as well as the critical component limiting the throughput rate. In this section we present a final design from a complete (as yet unreported) design study. Our final design is a small dynamic ROM with a power dissipation very much lower than more traditional static designs. In general, small ROMs have a disadvantage over larger ROMs in that the power dissipation of the output circuitry is only dependent on the number of output bits; large ROMs are able to amortize this area and power over the much larger number of storage bits. Our preferred design replaces the usual static circuitry with a dynamic sense amplifier and decoder.

5.3.1 Dynamic Sense Amplifier

The dynamic sense amplifier is shown in Figure 12.

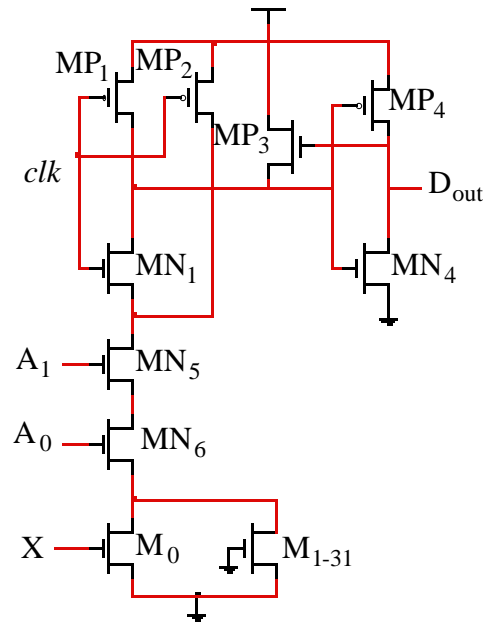


Figure 12. Dynamic sense amplifier

X is a word line from the decoder, M_0 is the selected ROM cell, and M_{1-31} are the unselected ROM cells of the same bit-line. MN_5 and MN_6 are one of the four column decoders, and connected to the true address bits A_1 and A_0 in this column. The presence of M_0 allows the pre-charged nodes to discharge producing a **1** at the output; the absence of M_0 produces a constant logic **0** at the output. The ROM programming is therefore carried out by selectively placing transistors in the storage transistor array. MP_3 is a weak p-channel pull-up transistor that reduces the charge sharing effect on the evaluation node. For this design we find that the power dissipation is very low and the area is smaller than similar ROM designs [3][20]. In [3] the 256 X 8 ROM area is $0.56\mu\text{m}^2$, using a $1.6\mu\text{m}$ CMOS process, our design consumes an area of $0.4\mu\text{m}^2$, using the target $1.5\mu\text{m}$ CMOS process. Our design reduces the power associated with a traditional differential output stage which need bias voltage and static current drain, where the static power dissipation is usually greater than the dynamic (working) power. So, in our design, the elimination of static power is a big saving for total power dissipation.

5.3.2 Dynamic Decoder

The Domino style decoder unit is shown in Figure 13. We have elected to build 32 separate stages rather than the usual tree decoder. The number of transistors is greater than the tree method, but the practical layout area is the same because of the non-rectangular decoder tree structure and the layout is made more modular; the power dissipation does not increase.

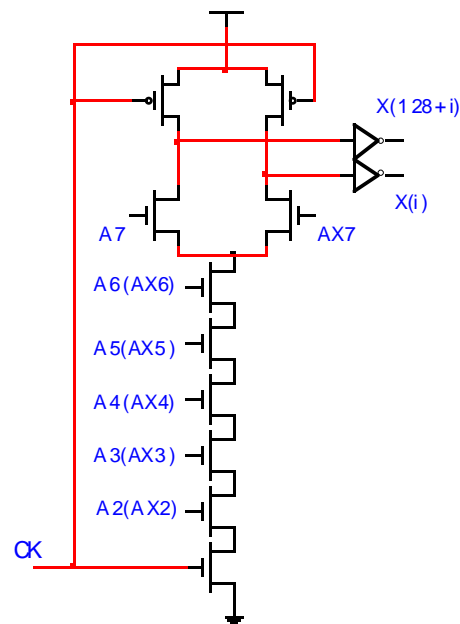


Figure 13. Dynamic decoder unit

5.4 The Modular Pipeline Structure

The True Single Phase Clock strategy for dynamic logic [13] is used throughout the whole design. TSPC is known to suffer from edge slew problems, and we have explored this issue quite fully in order to produce robust designs.

5.4.1 Pipeline Latch Design

In the TSPC strategy, the clock related problems are mainly self-skewing problems, and the design of a robust latch has to involve detailed simulations with chains or counters.

The TSPC latch was carefully studied in [13][21] and the clock slope dependent problems explored in [22]. In [22], the problems related with clock slope and self skew are analyzed and sizing of transistors was suggested as a suitable way to deal with the inherent problems. In partic-

ular, an example of sizing N and P blocks concludes that the size of the precharge transistor is an important parameter in preventing edge and skew failures.

Figure 14 is the familiar form of TSPC latch with the addition of an inverter buffer; this buffer proves indispensable to isolate the self loads of the internal sized transistors from any output circuitry. A qualitative analysis of the non-ideal clock problems with the latch, leads to the conclusion that M2-3 should be very weak to prevent false paths through M4-5 for a slow clock edge, and also to prevent problems with the input data changing during a slow clock transition. M6 should also be weak to prevent false discharge through M7-8 [22], and also M7-8 should be weak to provide sufficient hold time for a following TSPC latch stage.

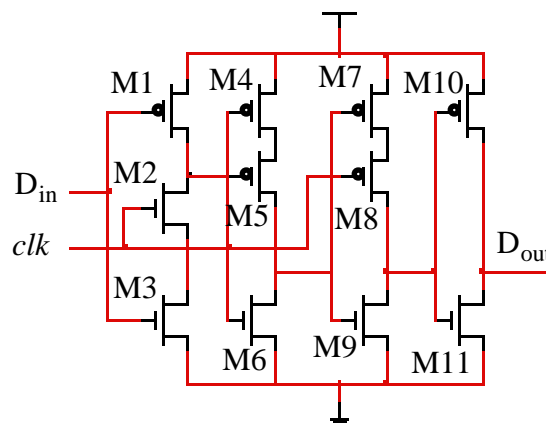


Figure 14. A Buffered TSPC Latch

Through repetitive simulation experiments with Spectre (both pre- and post- layout), we have determined the set of transistor dimensions shown in Table 2. These have been normalized to the width/length ratio of the n and p transistors in the final buffer inverter.

Table 2. Normalized Transistor Dimensions for a Robust TSPC Latch Design

Device	Size	Device	Size	Device	Size
M1	1/2	M2	1/7	M3	1/7
M4	1	M5	1	M6	1/4
M7	1/2	M8	1/2	M9	1
M10	1	M11	1		

It is clear that the resizing of transistors other than the p-channel pull-ups are important for a robust design. For our target $1.5\mu\text{m}$ technology, M10 is $6.6\mu/1.5\mu$ and M11 is $3.3\mu/1.5\mu$

5.4.2 Pipeline design template

For an *all TSPC* design, we consider the robust implementation of pipelined logic in dynamic circuits. In [23], the subject of skew and logic flexibility are explored, a set of rules for dynamic block connections are suggested, and various timing strategies provided. From this work we have produced a pipeline design template, as shown in Figure 15, which is suitable for the long pipeline cascades projected as implementation architectures for the Fermat ALU. In Figure 15 all of the logic functions are implemented in the N switching trees [24] of the preceding domino stages. The output signals of the latches may also be inverted to provide extra logic flexibility in the domino switching trees [24]. The Fermat ALU is designed as a three pipeline stage in this fashion, with the adders and ROM components formed as the n-channel logic blocks in Figure 15.

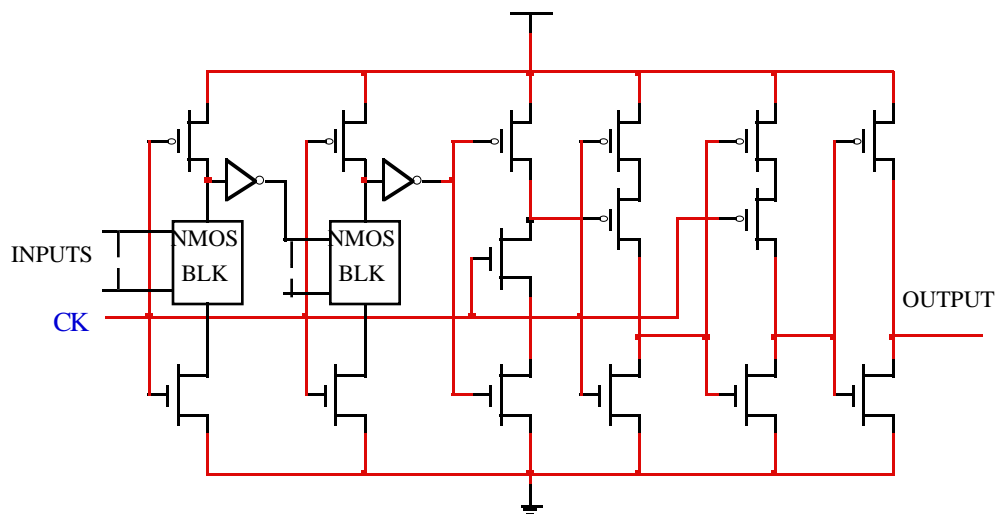


Figure 15. TSPC Design Template

Post-layout simulation results of the Fermat 257 unit, show the worst case propagation delay (evaluation delay) of each pipeline stage to be less than 3.7ns. So a 100 MHz safe working throughput rate is expected when the pulse width ratio is 50%, and up to 150 MHz is possible for a pulse width around 80%. A layout of the GF(257) part of the Fermat ALU is shown in Figure 16.

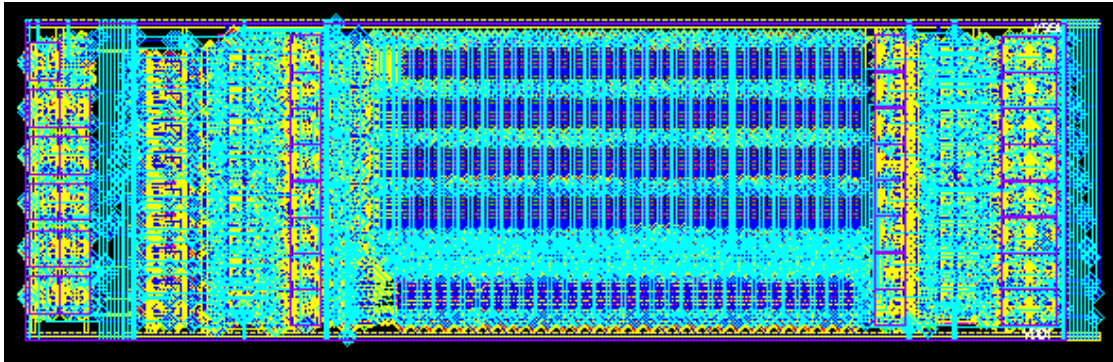


Figure 16. Layout of the Fermat ALU

6 CONCLUSIONS

In this paper we have discussed the application of modulus replication theory to the implementation of pipelined DSP computations. We have emphasized the importance of the VLSI structure that independent residue computations provide, and we have briefly reviewed the theory behind modulus replication, and the advantages that replication brings to the implementation of independent residue modules. We have demonstrated the algebraic advantages by discussing the construction of a Fermat ALU in some detail. The final design employs TSPC pipelined logic and we provide some new results in terms of VLSI design details. Our final design, in a 1.5μ CMOS process, is able to robustly pipeline at 100MHz with a conversion overhead of only 6% for a 150 tap filter.

7 REFERENCES

- [1] Schroeder, M.R., 1986. "Number Theory in Science and Communication." Springer-Verlag. Berlin.
- [2] Barraclough, S.R., Sotheran, M., Burgin, K., Wise, A.P., Vadher, A., Robbins, W.P. and Forsythe, R.M., 1989. "The Design and Implementation of the IMS A110 Image and Signal Processor." *IEEE Custom Integrated Circuits Conf.*, pp. 24.5.1-24.5.4
- [3] Mellott, J.D., Smith, J.C. and Taylor, F.J., 1993. "The Gauss Machine: A Galois-Enhanced Quadratic Residue Number System Systolic Array." *Proceedings of the 11th IEEE Symposium on Computer Arithmetic.*, Windsor, Canada. pp. 156-162.

- [4] Soderstrand, M.A., Jenkins, W.K., Jullien, G.A. and Taylor, F.J., 1986. "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing." IEEE Press. New York, NY.
- [5] Wigley, N.M. and Jullien, G.A., 1990. "On Modulus Replication for Residue Arithmetic Computations of Complex Inner Products." *IEEE Trans. Comp.* 39, August, pp. 1065-1076
- [6] Wigley, N.M. and Jullien, G.A., 1994. "Large Dynamic Range Computations Over Small Finite Rings." *IEEE Trans. Comp.* Vol. 43, No. 1, pp. 76-86
- [7] Chren, W.A.J., 1994. "Area and Latency Improvements for DDS Using the Residue Number System." *Proceedings of the 37th Mid-West Symp. on Circuits and Systems.*, Lafayette, LA. Paper 22.5 (in print).
- [8] Wang, Z., Jullien, G.A. and Miller, W.C., 1991. "Algorithms for Length 15 and 30 Discrete Cosine Transforms." *1991 Asilomar Conference on Circuits Systems and Computers.*, Pacific Grove, CA. pp. 111-115.
- [9] Jullien, G.A., Taheri, M., Bandyopadhyay, S. and Miller, W.C., 1990. "A Low-Overhead Scheme for Testing a Bit Level Finite Ring Systolic Array." *Journal of VLSI Signal Processing.* , Vol 2.3 pp. 131-138.
- [10] Jullien, G.A., 1980. "Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms." *IEEE Trans on Computers.* Vol. C-29, No.10, pp. 899-905
- [11] Zelniker, G. and Taylor, F.J., 1991. "A Reduced-Complexity Finite Field ALU." *IEEE Trans. on CAS.*, Vol. 38 , No. 12 pp.1571-1573.
- [12] Bayoumi, M.A., Jullien, G.A. and Miller, W.C., 1987. "A VLSI Implementation of Residue Adders." *IEEE Trans. on CAS.* , Vol. CAS-34 , No.3
- [13] Afghahi, M. and Svensson, C., 1990. "A Unified Single-Phase Clocking Scheme for VLSI Systems." *IEEE J. Solid-State Circuits.* 25, Feb., pp. 225-233
- [14] W. Luo, G.A. Jullien, N.M. Wigley, W.C. Miller, Z. Wang, 1995, "An Array Processor for Inner Product Computations Using a Fermat Number ALU", Proceedings of the 1995 Conference on Application Specific Array Processors, Strasbourg.
- [15] 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.

- [16] J. Wang, Z. Wang, G.A. Jullien, S.S. Bizzan, W. Luo and W.C. Miller, "Circuit Driven Delay Optimization of EMODL Carry Lookahead Adders," Proceedings of the Asilomar Conference on Signals, Systems and Computers, November, 1994, pp. 550-554
- [17] Jyh-Ming Wang, Ung-Chuan Fang, and Wu-Shiung Geng, "New Efficient Design for XOR & XNOR Functions on the Transistor Level," IEEE Journal of Solid State Circuits, pp.780-786, July 1994
- [18] C. Mead and L. Conway, "Introduction to Very Large Scale Integrated Systems", Addison-Wesley Publishing Co., 1980.
- [19] Thomas Lynch and Earl E Swartzlander, Jr, "A Spanning Tree Carry Lookahead Adder", IEEE Trans. on Computers, vol.41, no.8, pp.931-939, Aug.1992.
- [20] Jeremy Smith, Motorola Inc., private correspondence.
- [21] J.Yuan, I. Karlsson and C.Svensson, "A true sing-phase-clock dynamic CMOS circuit technique", IEEE J. of Solid-State Circuits, vol.22, no.5, pp.899-901, Oct 1987.
- [22] P.Larsson, C.Svensson, "Impact of Clock Slope on True Single Phase Clocked (TSPC) Circuits", IEEE J. of Solid-State Circuits, vol.39, no.6, pp.723-726, Jun 1994.
- [23] P.Larsson, PhD Dissertation, "Robustness of Digital CMOS Techniques with special Emphasis on the True Signal Phase Clocking Strategy", Linkoping University, Linkoping Studies in Science and Technology, Thesis No. 390, pp.29-35, Linkoping, Aug 1993.
- [24] G.A. Jullien, W.C.Miller, R.Gronin, L. Del Pul, S.S.Bizzan, and Dapeng Zhang, "Dynamic computational blocks for bit-level systolic arrays", Journal of Solid-State Circuits, vol.29, no.1, pp.14-22, Jan 94.

8 ACKNOWLEDGMENTS

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, software, and fabrication facilities from the Canadian Microelectronics Corporation.

PostScript error (--nostringval--, --nostringval--)