ISOMORPHIC TRANSFORMATION AND ITS APPLICATION TO THE MODULO \((2^n + 1)\) CHANNEL FOR RNS BASED FIR FILTER DESIGN

NEGOVAN STAMENKOVIĆ

1Faculty of Natural Sciences and Mathematics, University of Priština, Kosovska Mitrovica, Serbia

ABSTRACT

In this paper, the implementation of a Finite Impulse Response (FIR) filter in the Residue Number System (RNS), is presented, in which a modulo multiplier based on the isomorphism technique is used to perform multiplication in the \((2^n + 1)\) channel. An RNS modular multiplication in the Galois Field \(GF(2^n + 1)\) is presented in detail in this paper. The multiplication is based on the isomorphic mapping technique adapted to the residue arithmetic. The isomorphic encoder and decoder look-up tables in the \(GF(2^n + 1)\) are given. An architecture for FIR filter design based on distributed arithmetic for multiplication and accumulation in mentioned \((2^n + 1)\) channel is also presented. This architecture is discussed in details and compared with with architecture based on isomorphing technique.

Keywords: Galois field, Multiplication, Lookup table, Modular arithmetic, Distributed arithmetic.

INTRODUCTION

Modulo \((2^n + 1)\) multipliers of various types have been considered in literature (a) both inputs in standard representation, (b) one input in standard form and another in diminished-1 form and (c) both inputs in diminished-1 representation.

This paper develops an enhanced algorithm for the arithmetic modular \((2^n + 1)\) multiplication problem in the Residue Number System. The proposed algorithm is based on Galois finite field theory (Pradhan, 1978). Galois field \((GF(m))\) is a number system with a finite number of elements, \(m\), and two main arithmetic operations, called addition and multiplication. Other operations such as division can be derived from those two (Chen et al., 2007). Some of the formal properties of a finite field the following. They consist of a set number of \(GF(m)\), and two operations, modular addition (+) and modular multiplication (*). The result of adding or multiplying two numbers from the finite field is always an element of the field.

Mapping the arithmetic multiplication problem over the Galois field \(GF(m)\) eliminates many of the limitations of existing algorithms for modular \((2^n + 1)\) multiplication. And advantage of the proposed algorithm is that it has no restriction on the multiplier and the multiplicand, no diminished one multiplication, and no based extension operation.

A prime Galois field as a multiplier

A prime Galois field \(GF(m)\) is a finite field of order \(m\) \((m\) is the number of elements\) where \(m\) is a prime positive integer (Kitsos et al., 2003; Chen et al., 2014). They consist of two operations, modular addition (denoted by +) and modular multiplication (denoted by \(*\)), both operations are communicative and associative, that satisfies the usual arithmetic properties:

(a) The \((GF, +)\) is an Abelian group with an additive neutral element denoted by \(0\), such that \(a + 0 = a\) for any element \(a \in GF(m)\).
(b) The \((GF, 0, +)\) excluding the zero element is an Abelian group with a multiplicative neutral element denoted by \(1\), such that \(a * 1 = a\) for any element \(a \in GF(m)\).
(c) For every element \(a \in GF(m)\), there is an additive inverse element \(-a\), such that \(a + (-a) = 0\).
(d) For every nonzero element \(b \in GF(m)\) there is a multiplicative inverse element \(b^{-1}\) such that \(b * b^{-1} = 1\).
(e) Multiplication distributes across addition as: \((a + b)c = ac + bc\) and \(ca + cb = ca + cb\) for all \(a, b, c \in GF(m)\).

These properties can be satisfied if the field size is any prime number or any integer power of a prime.

The organization of the paper is as follows: Section 2 gives a brief overview of the index mapping method; Section 3 gives a short explanation of the design \((2^n + 1)\) channel for RNS based FIR filter design using the index mapping method given in Section 2; Section 4 deals with distributed arithmetic and its comparison with isomorphing transformatin; Section 5 deals with the conclusion of the work. Isomorphing encoded and decoded tables for modulus \((m = 2^n + 1)\) are given in Appendix.

THE INDEX MAPPING OVERVIEW

In the residue number system (RNS), an analogous method which can be, as logarithms multiplication used, to call index calculus (Padmavathy & Bhagvati, 2012). Using index mapping over

---

* Corresponding author: negovan.stamenkovic@pr.ac.rs

MATHEMATICS, COMPUTER SCIENCE AND MECHANICS
the Galois Field $GF(m)$, the multiplication operation can be implemented by the addition. The multiplication operation in RNS is a modular operation, therefore, multiplication can be done as an addition in RNS, which is easier than multiplication (Qi et al., 2012).

The groups $(G_1, *)$ and $(G_2, \odot)$ are said to be isomorphic if there is a one-to-one correspondence (bijection) $f : G_1 \rightarrow G_2$ that preserves the group operation, in other words, $f(a * b) = f(a) \odot f(b)$, for all $a, b \in G_1$.

The input and output index mapping of RNS numbers is based on the following definitions.

**Definition 1.** The Euler’s $\varphi(n)$ function or the totient function of a positive integer $n$ is the number of integers in the range $(1, 2, \ldots, n-1)$ which are relatively prime or co-prime to $n$. If $m$ is prime then $\varphi(m) = m - 1$.

For example, $\varphi(5) = 4$, the numbers 1, 2, 3, 4 are relatively prime to 5, but 5 is not. If $n = p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$, where $p_1, p_2, \ldots, p_m$ are distinct prime divisors of $n$ and $k_i \geq 1$, then

$$\varphi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})\ldots(1 - \frac{1}{p_m})$$

To calculate Euler’s function, the Matlab string given in Listing 1 can be used.

**Listing 1.** Euler’s $\varphi(n)$ function

```matlab
N = 48; % for example
n = 1:N-1;
ind = gcd(n,N)==1;
tot = n(ind)
```

**Definition 2.** Let $a \neq 0$ and $m > 0$ be relatively prime. Then $x$ is called the order of $a$ modulo $n$, denoted $x = \text{ord}_m(a)$ if $x$ is the smallest natural number so that $\langle a^x \rangle_m = 1$.

For example, order of $a = 3$ modulo $m = 11$ is $\text{ord}_{11}(3) = 5$ because $(3^5)_{11} = 1$.

**Definition 3.** Let $m \in \mathbb{N}$ and $g \in \mathbb{Z}$ be such that $\gcd(g, m) = 1$. Then $g$ is called a primitive root modulo $m$ if $\text{ord}_m g = \varphi(m)$, i.e. if the order of $g$ is equal to the maximal possible value. In other words, an integer $g$ is a primitive root modulo $m$ if the powers of $g$ generate all residue classes coprime to $m$.

For example, let $m = 17$ be a second order Fermat number, then the number $g = 3$ is the primitive roots of the prime $m$.

**Listing 2.** The number $g = 3$ is a primitive root modulo 17

```matlab
m = 17; % for example
k = 1:m-1;
g = 3;
ind = mod(g.*k,m);
ind = sort(ind)
ind =
    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

**Definition 4.** Let $m$ be any prime number, and let $g$ be any primitive root of $m$, then to each integer $a$, relative prime to $m$, there is a unique integer (index) $i$, denoted as $i = \text{ind}_g a$, such that

$$a = \langle g' \rangle_m, \quad 0 \leq i < m - 1$$

Indexes over Galois field $GF(m)$ have the following important properties:

1. $\text{ind}_g 1 = 0$,
2. $\text{ind}_g (a \times b) = (\text{ind}_g a + \text{ind}_g b)_{m-1}$,
3. $\text{ind}_g a^n = (n \times \text{ind}_g a)_{m-1}$,
4. $\text{ind}_g a = (\text{ind}_g g' + \text{ind}_g a)_{m-1}$, where $g'$ is any other primitive root.

**Definition 5.** For integer numbers $a, b$ and $m$, notation $b = \langle a \rangle_m$ means $a - b$ is divisible by $m$, that is $m| (a - b)$ or, equivalently, $a - b = km$ for some integer $k$.

For any integer $a, r = \langle a \rangle_m$ shall denote the unique integer remainder $r, 0 \leq r \leq m - 1$, obtained upon dividing $a$ by $m$; this operation is called reduction modulo $m$.

A special technique, based on isomorphic transformations (Jullien, 1980), can be used in RNS to transform the modular multiplication into a simpler modular addition. It is based on the concept of indices that are similar to logarithms, and primitive roots $g$ which are similar to logarithm bases. It is possible to demonstrate that if the number $m$ is a prime there exists a number of primitive roots (the number of the primitive roots can be computed by using the Euler's function) that share the following property: every element of the field $GF(m) = \{0, 1, \ldots, m - 1\}$ excluding the zero element can be generated by using the following equation

$$x = \langle g^k \rangle_m$$

where $k$ (index) is an integer number and $g$ a primitive root. In this way, an isomorphism exists between the multiplicative group $\{x\} = \{1, 2, \ldots, m - 1\}$ with the multiplication modulo $m$, and the additive group $\{k\} = \{0, 1, \ldots, m - 2\}$ with the addition modulo $m - 1$. Multiplication of two integers can now be performed by adding the corresponding indices mod $(m - 1)$, and then finding its inverse index value. Thus, the product of $x$ and $y$ is given by

$$\langle x \times y \rangle_m = \langle g^k \rangle_m \times \langle g^k \rangle_m = g^{\langle k + k \rangle_{m-1}}$$
This approach is known as index calculus. By using isomorphisms, the product of the two residue numbers is mapped into the sum of their indexes which are obtained by an isomorphic mapping. The scheme for an index calculus multiplier is shown in Figure 1. This multiplication needs three ROM look-up tables and an addition modulo \((m − 1)\).

\[
x = g^x \\
y = g^y
\]

\[x = g^x \rightarrow \text{Index ROM} \rightarrow k_x \rightarrow (k_x + k_y)_{m−1} \rightarrow \text{Inverse Index ROM} \rightarrow (x \times y)_m \]

**Figure 1.** An index calculus multiplier.

The modulo \((m − 1)\) adder has two \(n\)-bit inputs and one \(n\)-bit output, were \(n = \lfloor \log_2(m - 2) \rfloor\).

Proposed applications can only be computed with only index ROM and inverse index ROM look-up tables and addition modulo \(m − 1\).

**APPLICATION ISOMORPHING TRANSFORMATION TO THE MODULO \((2^4 + 1)\) CHANNEL FIR FILTER**

An \(m\) channel of \(N\) taps (degree) for RNS based FIR filters is described by the ordinary expression

\[
y_n = \sum_{k=0}^{N-1} A_k x_{n-k} \tag{3}\]

where \(x(n)\) is the input to the filter, \(A_k\) represents the filter coefficients and \(y_n\) is the output of the filter. This can be implemented using a single Multiply Accumulate (MAC) engine, but it would require \(N\) MAC cycles, before the next input sample can be processed. Clearly, it is necessary to apply modular arithmetic.

Two direct isomorphic transformations to obtain \(A_n \rightarrow k_n\) and \(x_{n-k} \rightarrow k_{x,n-k}\), and one inverse isomorphic transformation to obtain \(y_n \rightarrow (k_{x,n} + k_{y,n-k})_{16}\), are performed. Because of the complexity of modular multiplication, we used the isomorphism technique to implement the product of residues.

The prime number \(m = 2^4 + 1 = 17\) is a second order Fermat number and it has 7 primitive roots. The complete list of primitive roots for \(GF(17)\) is: \([3, 5, 6, 7, 10, 11, 12]\). In isomorphism Table 1 the elements of prime field \(GF(17)\), which are generated by using mapping equation, (1), for primitive root \(g = 3\), are given.

**Table 1.** The isomorphism table for \(m = 17\) and \(g = 3\).

| \(GF(17)\) | 1 | 3 | 9 | 10 | 13 | 5 | 15 | 11 | 16 | 14 | 8 | 7 | 12 | 2 | 6 |
|------------|---|---|---|----|----|---|----|----|----|----|----|---|----|---|---|
| \(k\)      | 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |

The modular product of two integer elements \(x\) and \(y\) belonging to the Galois field with \(m\) elements is implemented in the following way

1. Forward mapping of \(x \in GF(m)\) and \(y \in GF(m)\) in the corresponding indices \(k_x\) and \(k_y\).
2. Addition modulo \((m − 1)\) of the two indexes.
3. Reverse mapping of the result of the addition to obtain the final result of the modular product.

The block diagram of a typical isomorphic implementation of the three tap modulo 17 channel of RNS FIR filter is shown in Fig. 2.

![Figure 2.](image)

**Figure 2.** Isomorphic multiplication based Implementation of a 3-tap modulo 17 channel for RNS based FIR Filter.

Product \(A_k x_{n-k}\) is transferred into Register R. The modulo \((2^4 + 1)\) adder in the next stage adds the present sum to the previous sum fed back from Register ACC, which is initialized to zero, thus accumulating the summation of the products \(A_k x_{n-k}\), over the interval \(i = 1, \ldots, N\). The final sum is left in Register ACC.

The architecture shown in figure 2 is also suitable for the modulo \(2^8 + 1\) channel of the RNS based FIR filter design. Isomorphic encoded and decoded tables for modulus \((m = 2^8 + 1)\) are given in Appendix.

For example, for the modular multiplication in the GF (17) of the integers \(x = 15\) and \(y = 16\) the corresponding indices are \(k_x = 6\) and \(k_y = 8\) respectively, and these can be found in Table 1. Addition modulo 16 is \(k_x + k_y)_{m−1} = \langle 6 + 8\rangle_{16} = 14\).

Reverse isomorphic mapping of the index 14 gives product \(\langle 15 \times 16\rangle_{17}\) in the field GF(17) which is equal to 2. Thus \(\langle 3^{14}\rangle_{17} = \langle 15 \times 16\rangle_{17} = 2\).
Although theoretically multiplication by zero can not be performed using isomorphing technique, notice that by using used look-ups one can solve the problem by adding an additional code to every ROM.

**DISTRIBUTED ARITHMETIC ARCHITECTURE**

Distributed arithmetic (DA) (Nagaljyothi & SriDevi, 2017) is a well known method for the calculation of the sum of products to perform Multiplication and Accumulation (MAC). It is a very common method in many Digital Signal Processing (DSP) Algorithms. It should be noted that the DA method is applicable only to cases where the (MAC) operation involves fixed coefficients.

Let the variable \( y \) hold the result of an inner product operation between a integer data vector \( x_i \) and a integer coefficient vector \( A_n, i = 0, 1, 2, \ldots, N - 1 \). The distributed arithmetic representation of the inner modular product operation is as follows:

\[
y = \sum_{n=0}^{N-1} A_n x_n \tag{4}
\]

where \( A_n \) are constant coefficient values (e.g. coefficients of FIR filter) and \( x_n = [b_{n,0}, b_{n,1}, \ldots, b_{n,K-1}] \) is the corresponding data vector with \( N \) inputs, each binary encoded with bit length of \( n \). Using the standard multiply and accumulate approach, it is obvious that the calculation of this inner product will take \( N \) multiply and accumulate execution cycles, corresponding to the number of coefficients used in (4).

Now consider expressing each input in the data vector, \( x_n \), in the unsigned binary number form as

\[
x_n = \sum_{k=0}^{K-1} b_{n,k}2^k, \quad b_{n,k} \in [0.1] \tag{5}
\]

where \( K - 1 \) is binary word length.

The inner product \( y \) in (4) can then be written in the form associating it directly with the bit values of the inputs in the data vector

\[
y = \sum_{k=0}^{K-1} \left( \sum_{n=0}^{N-1} A_n b_{n,k} \right)2^k = \sum_{k=0}^{K-1} f(A_n, b_{n,k})2^k \tag{6}
\]

where

\[
f(A_n, b_{n,k}) = \sum_{n=0}^{N-1} A_n b_{n,k} \tag{7}
\]

The function (6) contains values representing the sum of products \( A_n \) with the individual binary bit value \( b_{n,k} \) of the data vector \( x_n \). Since the \( b_{n,k} \) bit value is either 0 or 1, while the value of each \( A_n \) is constant, there are \( 2^N \) possible combination values of \( f(A_n, b_{n,k}). \)

Applying RNS arithmetic using a moduli set, for example RNS modulo basis is \( B = \{m_1, m_2, \ldots, m_L\} \) where one of them is \( m_l = 2^k + 1 \), for the inner product in (6), it can be rewritten in terms of its residue \( m_l \), i.e.

\[
y_i = \left( \sum_{k=0}^{K-1} f(A_n, b_{n,k})2^k \right)_{m_l} \tag{8}
\]

By applying the algebra of RNS we get follows:

\[
y_i = \left( \sum_{k=0}^{K-1} f_m(\lambda_n, b_{n,k})(2^k)_{m_l} \right)_{m_l} \tag{9}
\]

Hence values of \( f_m(\lambda_n, b_{n,k}) \) can be pre-computed and stored in the Look Up Table LUT, which can be subsequently clocked out by using the bit-serial stream of the input vector for the accumulation operation. However each of the value needs to be first scaled with the \( (2^2)_{m_l} \) factor, which is difficult to be implemented in hardware due to its modulo operation with respect to modulo \( m_l \).

The evaluation of a polynomial \( y_i \) of degree \( N \) allows only \( N \) multiplications and \( N \) additions. This is optimal, since there are polynomials of degree \( N \) that cannot be evaluated with fewer arithmetic operations.

\[
y_i = \left( f_m(\lambda_n, b_{n,0}) + f_m(\lambda_n, b_{n,1})2 + f_m(\lambda_n, b_{n,2})2^2 + \cdots + f_m(\lambda_n, b_{n,K-1})2^{K-1} \right)_{m_l} \tag{10}
\]

Using Horner's method for evaluating a polynomial we can rewrite

\[
y_i = \left( f_m(\lambda_n, b_{n,0}) + \left[ f_m(\lambda_n, b_{n,1}) + \left[ f_m(\lambda_n, b_{n,2}) + \cdots + \left[ f_m(\lambda_n, b_{n,K-2}) + f_m(\lambda_n, b_{n,K-1}) \right]2 \right] \cdots 2 \right] \right)_{m_l} \tag{11}
\]

The basic distributed arithmetic architecture of a three tap \((N = 3)\) FIR filter is shown in Fig. 3. The bank of shift registers in Fig. 1 stores four consecutive input samples. The concatenation of the rightmost bits of the shift registers becomes the address of the LUT. The shift registers are shifted right at every clock cycle. The corresponding LUT entries are also shifted and accumulated \( N \) consecutive times where \( N \) is the precision of the input data.

Example. Let \( N = 3 \) and \( K = 5 \) for \( m_1 = 17 \). Equation (11) is reduced to

\[
y_i = f_{17}(A_n, b_{k,0}) + 2f_{17}(A_n, b_{k,1}) + 2f_{17}(A_n, b_{k,2}) + 2f_{17}(A_n, b_{k,3}) + 2f_{17}(A_n, b_{k,4}) + \left[ b_{2,4}, b_{1,4}, b_{0,4} \right] \tag{12}
\]

where \( [b_{2,4}, b_{1,4}, b_{0,4}] \) create a memory address which is loaded into the memory address register, and

\[
f_{17}(A_n, b_{k,n}) = \sum_{k=0}^{K-1} A_n b_{n,k} \tag{13}
\]

for \( n = 0, 1, 2, 3, 4 \). For \( n = 4 \) we have

\[
f_{17}(A_n, b_{k,n}) = \left( A_0 b_{0,4} + A_1 b_{1,4} + A_2 b_{2,4} \right)_{17}
\]

The DA of FIR filter consists of the LUT, shift registers and scaling accumulator. The block diagram of a typical distributed arithmetic implementation of the three taps RNS FIR filter for modulo 17 channel is shown in Fig. 3.
We can store data in a look-up table of $2^K$ words addressed by $K$-bits. The multiplication by 2 can be implemented with a one-bit shift to the left.

The look-up-table size increases exponentially with the filter coefficients. A smaller number of coefficients can be realized very easily with a LUT of a smaller size. When dealing with larger coefficients, they will take up a lot of storage space in the LUT, for implementation and also reduce the calculation speed.

CONCLUSIONS

The proposed algorithm over Galois field GF($m$) provides an efficient algorithm for the modulo $2^8 + 1$ multiplication problem. Efficient procedures were proposed to convert the multiplication problem to the addition problem. The proposed algorithm and mapping procedure can be implemented using lookup tables, which means that multiplication in RNS can be computed very fast. The results of this research research can be used to design a general purpose multimoduli ALU.

A modulo multiplier based on the isomorphism technique is compared with those realized as the distributed arithmetic. Isomorphism technique has the following advantages. It does not contain shift registers and memory size is not in the correlation with the FIR filter degree.

APPENDIX

Isomorphic transformation for $m = 2^8 + 1$

Prime number $2^8 + 1 = 257$ is a Fermat number order three and it has 128 primitive roots. The smallest primitive root of modulo 257 is 3, because ord$_{257}(3) = \varphi(257) = 256$. According to Definition 2, $(3^{256})_{257} = 1$ is calculated

The Encoder in Table 2 is determined from the mapping $(3^k)_{257}$ for indices $k = 1, 2, \ldots, 256$.

Table 2. The isomorphism encoder table for $m = 257$, and primitive root $g = 3$.

| $(3^{256})_m$ | $k$ | $(3^{256})_m$ | $k$ | $(3^{256})_m$ | $k$ | $(3^{256})_m$ | $k$ |
|--------------|-----|--------------|-----|--------------|-----|--------------|-----|
| 1            | 0   | 1            | 199 | 2            | 199 | 3            | 199 |
| 2            | 48  | 4            | 199 | 46           | 134 | 5            | 24 |
| 3            | 1   | 5            | 199 | 47           | 61 | 6            | 199 |
| 46           | 134 | 7            | 199 | 52           | 28 | 8            | 199 |
| 55           | 69  | 9            | 199 | 56           | 51 | 10           | 199 |
| 6            | 49  | 11           | 199 | 57           | 98 | 12           | 199 |
| 69           | 29  | 13           | 199 | 58           | 142 | 14          | 199 |
| 73           | 44  | 15           | 199 | 59           | 251 | 16          | 199 |
| 102          | 83  | 17           | 199 | 60           | 152 | 18          | 199 |
| 121          | 138 | 19           | 199 | 61           | 118 | 20          | 199 |
| 145          | 202 | 21           | 199 | 62           | 209 | 22          | 199 |
| 164          | 41  | 23           | 199 | 63           | 194 | 24          | 199 |
| 185          | 170 | 25           | 199 | 64           | 96  | 26          | 199 |
| 206          | 142 | 27           | 199 | 65           | 141 | 28          | 199 |
| 227          | 178 | 29           | 199 | 66           | 140 | 30          | 199 |
| 248          | 122 | 31           | 199 | 67           | 139 | 32          | 199 |
| 270          | 138 | 33           | 199 | 68           | 138 | 34          | 199 |
| 292          | 137 | 35           | 199 | 69           | 137 | 36          | 199 |
| 314          | 136 | 37           | 199 | 70           | 136 | 38          | 199 |

Prime number $2^8 + 1 = 257$ is a Fermat number order three and it has 128 primitive roots. The smallest primitive root of modulo 257 is 3, because ord$_{257}(3) = \varphi(257) = 256$. According to Definition 2, $(3^{256})_{257} = 1$ is calculated

The Encoder in Table 2 is determined from the mapping $(3^k)_{257}$ for indices $k = 1, 2, \ldots, 256$.

Table 2. The isomorphism encoder table for $m = 257$, and primitive root $g = 3$. 

| $(3^{256})_m$ | $k$ | $(3^{256})_m$ | $k$ | $(3^{256})_m$ | $k$ | $(3^{256})_m$ | $k$ |
|--------------|-----|--------------|-----|--------------|-----|--------------|-----|
| 1            | 0   | 1            | 199 | 2            | 199 | 3            | 199 |
| 2            | 48  | 4            | 199 | 46           | 134 | 5            | 24 |
| 3            | 1   | 5            | 199 | 47           | 61 | 6            | 199 |
| 46           | 134 | 7            | 199 | 52           | 28 | 8            | 199 |
| 55           | 69  | 9            | 199 | 56           | 51 | 10           | 199 |
| 6            | 49  | 11           | 199 | 57           | 98 | 12           | 199 |
| 69           | 29  | 13           | 199 | 58           | 142 | 14          | 199 |
| 73           | 44  | 15           | 199 | 59           | 251 | 16          | 199 |
| 102          | 83  | 17           | 199 | 60           | 152 | 18          | 199 |
| 121          | 138 | 19           | 199 | 61           | 118 | 20          | 199 |
| 145          | 202 | 21           | 199 | 62           | 209 | 22          | 199 |
| 164          | 41  | 23           | 199 | 63           | 194 | 24          | 199 |
| 185          | 170 | 25           | 199 | 64           | 96  | 26          | 199 |
| 206          | 142 | 27           | 199 | 65           | 141 | 28          | 199 |
| 227          | 178 | 29           | 199 | 66           | 140 | 30          | 199 |
| 248          | 122 | 31           | 199 | 67           | 139 | 32          | 199 |
| 270          | 138 | 33           | 199 | 68           | 138 | 34          | 199 |
| 292          | 137 | 35           | 199 | 69           | 137 | 36          | 199 |

MATHEMATICS, COMPUTER SCIENCE AND MECHANICS 67
Table 3. The isomorphism decoder table for $m = 257$, and primitive root $g = 3$.

| $k$ | $(3^k)_m$ | $k$ | $(3^k)_m$ | $k$ | $(3^k)_m$ | $k$ | $(3^k)_m$ |
|-----|-----------|-----|-----------|-----|-----------|-----|-----------|
| 0   | 1         | 3   | 22       | 64  | 241       | 0   | 1         |
| 1   | 3         | 4   | 32       | 65  | 209       | 1   | 0         |
| 2   | 9         | 5   | 94       | 67  | 67        | 2   | 0         |
| 3   | 27        | 6   | 116      | 81  | 127       | 3   | 1         |
| 4   | 81        | 7   | 135      | 84  | 128       | 4   | 1         |
| 5   | 243       | 8   | 176      | 88  | 132       | 5   | 1         |
| 6   | 621       | 9   | 191      | 90  | 159       | 6   | 1         |
| 7   | 113       | 10  | 217      | 94  | 171       | 7   | 1         |
| 8   | 81        | 11  | 135      | 97  | 188       | 8   | 1         |
| 9   | 243       | 12  | 217      | 100 | 256       | 9   | 1         |
| 10  | 621       | 13  | 171      | 102 | 208       | 10  | 0         |

This represents the final product of 13 that is given in Table 3. It can easily verified that $(100 \times 234)_{257} = 13$. The scheme in Fig. 1 should be completed by a few gates to detect when one of the two operands is zero (no corresponding index in the isomorphism). When a zero is detected, the product is set to zero. Because isomorphic multipliers use modular adders in combination with two isomorphic tables, the modular adder in Fig. 1 should be replaced by a binary adder and incorporated in the inverse isomorphic mapping table. The resulting scheme will be faster and consumes less power, as detailed in (Nannarelli et al., 2003).

REFERENCES

Chen, G., Bai, G., & Chen, H. (2007). A New Systolic Architecture for Modular Division. IEEE Transactions on Computers, 56(2), pp. 282-286. doi:10.1109/tc.2007.20

Chen, R. J., Fan, J. W., & Liao, C. H. (2014). Reconfigurable Galois Field multiplier. In 2014 International Symposium on Biometrics and Security Technologies (ISBAST). Institute of Electrical and Electronics Engineers (IEEE), pp. 112-115. doi:10.1109/isbast.2014.7013104

Jullien, G. A. (1980). Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms. IEEE Transactions on Computers, C-29(10), pp. 899-905. doi:10.1109/tc.1980.1675473

Kitos, P., Theodoridis, G., & Koufopavlou, O. (2003). Theodoridis, G., & Koufopavlou, O. 2003. An efficient reconfigurable multiplier architecture for Galois field GF(2m). Microelectronics Journal, 34(10), pp. 975-980. doi:10.1016/s0026-2692(03)00172-1

NagaJyothi, G. & SriDevi, S. (2017). Distributed arithmetic architectures for FIR filters-A comparative review. In 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). Institute of Electrical and Electronics Engineers (IEEE), pp. 2684-2690. doi:10.1109/wispnet.2017.8300250

Nannarelli, A., Cardarilli, G. C., & Re, M. (2003). Power-delay tradeoffs in residue number system. In Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS ’03..Institute of Electrical and Electronics Engineers (IEEE), pp. 413-416. doi:10.1109/iscas.2003.1206300

Padmavathy, R. & Bhagvati, C. (2012). Discrete logarithm problem using index calculus method. Mathematical and Computer Modelling, 55(1-2), pp. 161-169. doi:10.1016/j.mcm.2011.02.022

Pradhan, D. K. (1978). A Theory of Galois Switching Functions. IEEE Transactions on Computers, C-27(3), pp. 239-248. doi:10.1109/tc.1978.1675077

Qi, H., Kim, Y. B., & Choi, M. (2012). A high speed low power modulo $2n + 1$ multiplier design using carbon-nanotube technology. In 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS). Institute of Electrical and Electronics Engineers (IEEE), pp. 406-409. doi:10.1109/mwscas.2012.6292043