Current-mode CMOS Galois Field Circuits

Zeljko Zilic and Zvonko Vranesic

Department of Electrical and Computer Engineering
University of Toronto

Abstract

Use of current-mode CMOS circuits for implementation of multiple-valued logic (MVL) functions has been considered in a number of recent papers. In this paper, we present an application of these circuits in realization of Galois field operations. We also give a new algorithm for determination of polynomial representations for arbitrary functions over a class of Galois fields implementable with presently available MVL circuits.

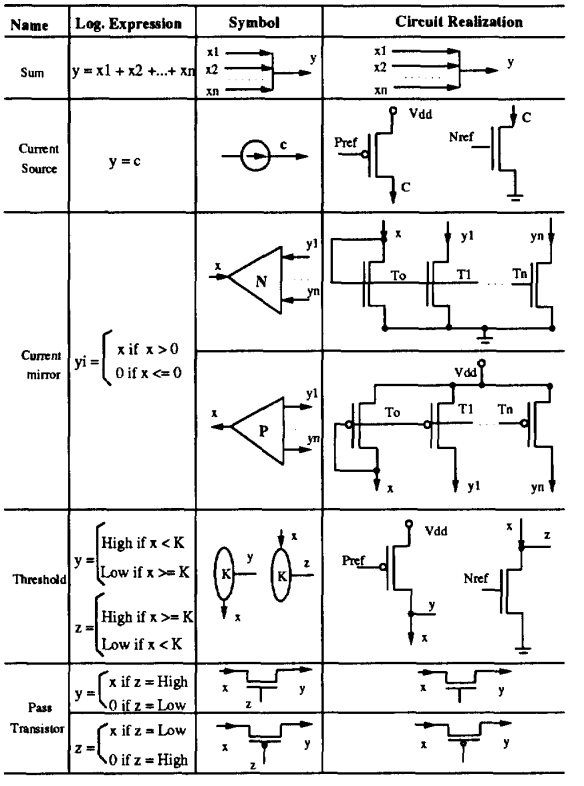
1.0 Current-Mode MVL Functions

Recent experience shows that current-mode circuits are attractive for implementation of MVL functions, particularly when the radix is greater than 3. Most of the circuits and synthesis techniques in the literature have been intended for the 4-valued environment [2], [4], [5]. Current-mode circuits offer several advantages, but they also have some disadvantages. Perhaps the most important of these are the ease of summation of signals and the difficulty in distribution of signals caused by the fanout being equal to one. In practice, it is useful to augment the current-mode circuits with some intermediate voltage-mode circuits, which often results in more effective designs. Figure 1 gives the basic blocks used in our design. All of these blocks have been used before [2], [4], [5]. Here, we will only summarize their characteristics.

Currents are summed by means of a simple wired connection. Current sources are realized as an N-type or a P-type transistor with the gate connected to a reference voltage. The amount (value) of current is proportional to the ratio of W/L.

Signal distribution can be performed using *current mirrors*, which can be of either N-type or P-type. We will use both types. The current produced at the output is determined by the input current and the ratio between output and input W/L values. In addition to signal distribution, the current mirror will be used for multiplication with a constant and for sign reversal.

FIGURE 1. Basic Current Mode Blocks



A *threshold detector* block takes a multiple-valued input current signal and produces a voltage signal. The output signal is High if the input exceeds a predefined reference value; otherwise the output signal is Low. Threshold detectors generate signals that can be used with normal binary logic gates and for control of pass-transistor networks.

A *pass transistor* is a voltage-controlled device, which acts as a switch, depending on the gate voltage. We will use both N and P-type pass transistors.

0195-623X/93 $03.00 © 1993 IEEE

2.0 Galois Field Circuits

Binary Galois field circuits have been investigated by many researchers. Much less work has been done on MVL versions of such circuits. We note the I^2L implementations of 4-valued Galois field operations reported by Dao [8].

This paper is primarily concerned with circuits operating over Galois fields with four elements *(GF4)*, because of the ease of implementation of 4-valued current-mode CMOS circuits. We will show that the more general class of Galois field circuits can be obtained using the basic blocks presented here.

A *field* [6] is a set with two operations that are closed with respect to that set. We will call them multiplication and addition. A certain number of axioms, ensuring the existence of neutral and inverse elements with respect to both operations, as well as commutativity and distributivity of multiplication with respect to addition, define the structure of a field.

FIGURE 2. GF4 Operations

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

All fields containing a finite number of elements must have the number of elements equal to a prime number (p) or some power of it (p^n). Such fields are known as *Galois fields*. In the case of a field with a prime number of elements, both operations are defined as usual modulo addition and multiplication. In the other cases, the operations are more complex. They are defined with respect to some irreducible polynomial over a simpler field. This means that if we have a field with p^n elements, then the operations are defined using some irreducible polynomial of degree n over the field with p elements. The usual notation for fields is *GF(p)*, or *GFp*, for the prime cardinality case, and $GF(p^n)$, or GFp^n, for the general case.

FIGURE 3. GF4 Addition Compared to Absolute Difference

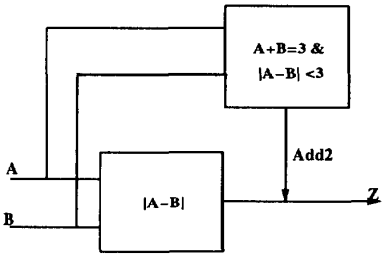
+	0	1	2	3
0	0	1	2	3
1	1	0	**3**	2
2	2	**3**	0	1
3	3	2	1	0

AD	0	1	2	3
0	0	1	2	3
1	1	0	**1**	2
2	2	**1**	0	1
3	3	2	1	0

The simplest possible case is to have a two-element field. Then, the addition is defined as the logical XOR operation, while the multiplication is defined as the logical AND. For a 4-element field, GF4, the addition and multiplication operations are defined in Figure 2. Subtraction is the same operation as the addition. This property is the consequence of the fact that the diagonal elements are zero (if $a + a = 0$, then $a = -a$) and for such fields, we will say that they are of *characteristic two.*

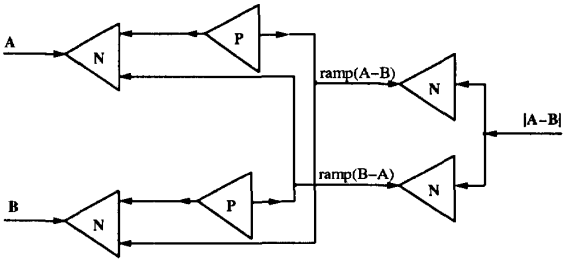
2.1 GF4 Adder

FIGURE 4. GF4 Adder



We will use the fact that GF4 is of characteristic two to derive a simple implementation of the addition circuit. Note that the absolute difference circuit will produce the output with diagonal elements equal to zero. We can see, from Figure 3, that the addition differs from the absolute difference in only two entries of the addition table. Therefore, an attractive realization of the addition operation is to use the absolute difference plus a correction circuit for the two entries outlined in bold in the figure.

FIGURE 5. Absolute Difference Circuit



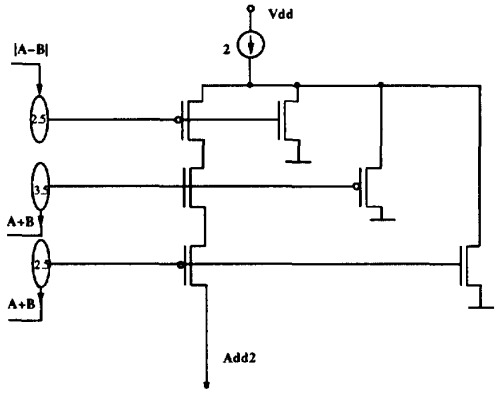
The absolute difference circuit can be implemented using the following equation:

$$|A-B| = ramp\,(A-B) + ramp\,(B-A)$$

where the *ramp* operation follows naturally from the current mirror operation. A current mirror forwards the input current if it is greater than zero, otherwise a zero output is produced. The two differences required can be produced by reversing the sign (with a current mirror) of the subtrahend and then by a simple wired-sum connection. Figure 5 shows the realization of the absolute difference circuit. Note that the arrowheads are used to indicate the actual directions of current.

The correction factor involves adding a current of value 2 when the inputs (A, B) have the values (1, 2) or (2, 1). A suitable correction circuit is given in Figure 6.

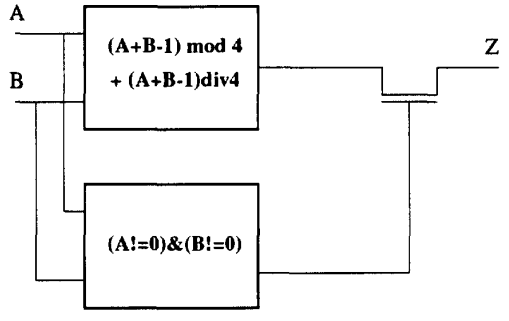
FIGURE 6. Correction Circuit



2.2 The GF4 Multiplier

The multiplier circuit can be designed as shown in Figure 7. The product is zero whenever one or both inputs are zero. The pass transistor at the output performs this function.

FIGURE 7. GF4 Multiplier



The top block in Figure 7 can be obtained using an adder with a correction (bias) factor of -1 at the output. When this output is greater than 3, another correction factor is switched on. This time, 3 is subtracted. Figure 8 shows the corresponding circuit.

The bottom block comprises two threshold detectors set up to detect zero, which control the output pass transistor.

2.3 The GF16 Multiplier

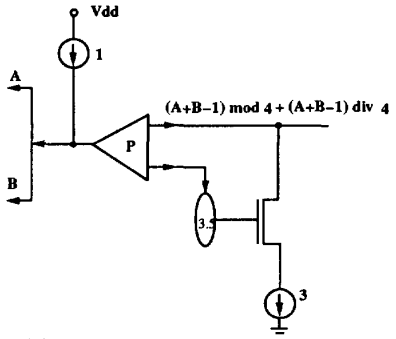
We can produce the GF16 multiplier using the four-valued primitives by representing an element of GF16 in form of an ordered couple of GF4 elements. Thus, every GF16 element will be represented in the form:

$$A = a_1 + \beta a_2$$

where β is a root of an irreducible polynomial over GF4. One such polynomial in GF4 is

$$f = x^2 + x + 2$$

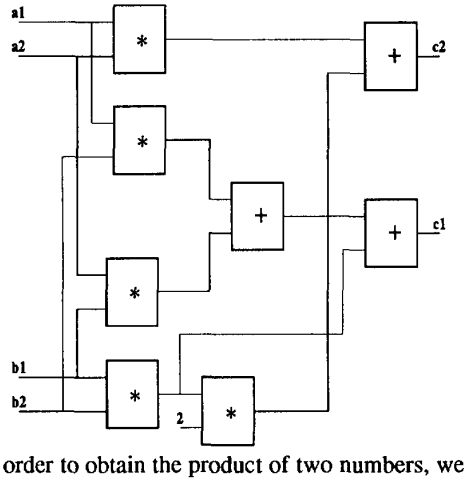
FIGURE 8. Implementation of the Top Block in Figure 7



The addition is then performed simply by adding the respective components of the GF4 representation, i.e. if we have two GF16 numbers A and B, then the following holds

$$A + B = a_1 + a_2 + (b_1 + b_2)\beta$$

FIGURE 9. GF16 Multiplier



In order to obtain the product of two numbers, we will multiply these two representations to obtain

$$A \times B = a_1 a_2 + (a_1 b_2 + a_2 b_1)\beta + b_1 b_2 \beta^2$$

and then replace the squared term with the remainder of the polynomial found above, since $\beta^2 = \beta + 2$. Thus, we can write the multiplication expression as:

$$A \times B = a_1 a_2 + 2b_1 b_2 + (a_1 b_2 + a_2 b_1 + b_1 b_2)\beta$$

The circuit for performing GF16 multiplication, based on this expression, is shown in Figure 9.

3.0 Circuit Performance

The described circuits have been simulated using HSPICE. The results are shown below. We have chosen the currents of 0, 20, 40 and 60 μA to represent logic val-

ues of 0, 1, 2 and 3. The sizes of all transistors have been scaled with respect to the basic transistor size of L/W= 3/6 μm .

Figure 10 gives the waveforms for the GF4 adder. This circuit shows some ringing in a case when the correction circuit is switched on and off, which should not cause undue problems if these circuits are used in synchronous designs.

FIGURE 10. GF4 Adder - SPICE Simulation Results

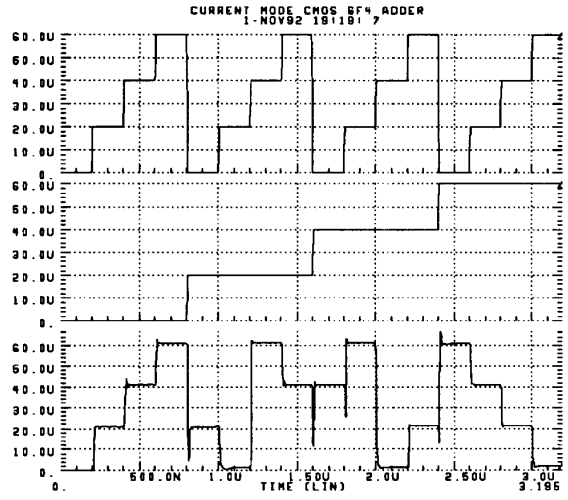
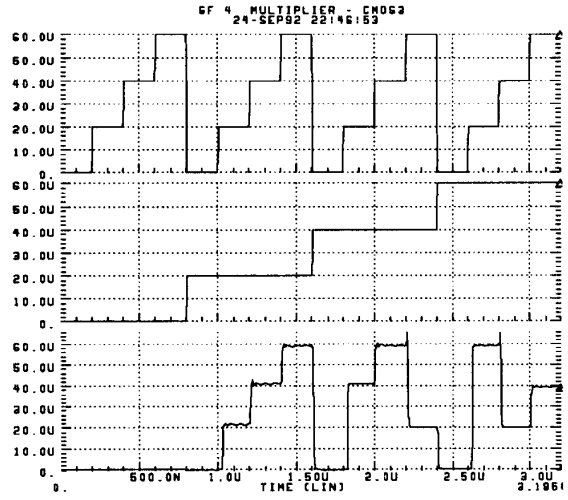


FIGURE 11. GF4 Multiplier- SPICE Simulation Results



As we can see from the figure, the delays are the largest when going to or leaving zero. These delays are twice as large as the other delays. This delay occurs because of the "dead time" of the current mirror. Figure 11 gives the simulation results for GF4 Adder. Again, the transitions from and to zero have the biggest delays. The transistor parameters are given in Figure 12:

FIGURE 12. Transistor Parameters

```
* PARAMETERS FOR PMOS(M1)
.MODEL M1 PMOS LEVEL=3 VTO=-0.787 GAMMA=0.667 PHI=0.7 IS=1E-16
+     PB=0.8 CGSO=3.2844E-10 CGDO=3.2844E-10 CGBO=1.5E-10
+     CJ=4.1E-4 MJ=0.54 CJSW=3.4E-10 MJSW=0.30 JS=1.0E-4
+     TOX=2.5E-8 NSUB=4.0E16 NFS=5.8601E11 TPG=+1 XJ=1.7403E-7
+     LD=1.3609E-7 UO=137.0 VMAX=271960 XQC=1.0 FC=0.5
+     DELTA=0 THETA=8.6257E-2 ETA=6.729E-2 KAPPA=8.125
*
* PARAMETERS FOR NMOS(M2)
.MODEL M2 NMOS LEVEL=3 VTO=0.736 GAMMA=0.366 PHI=0.61 IS=1E-16
+     PB=0.8 CGSO=1.9734E-10 CGDO=1.9734E-10 CGBO=1.5E-10
+     CJ=2.9E-4 MJ=0.486 CJSW=3.3E-10 MJSW=0.33 JS=1.0E-4
+     TOX=2.5E-8 NSUB=4.0E16 NFS=5.7651E11 TPG=-1 XJ=1.65E-7
+     LD=2.6508E-7 UO=436.7 VMAX=229170 XQC=1.0 FC=0.5
+     DELTA=1.0 THETA=0.2123 ETA=3.741E-2 KAPPA=0
*
```

4.0 Polynomial Expansion Over GF

Any function can be represented as a polynomial over a suitable Galois field, which means that blocks presented here are sufficient for realizing any function. We will now present a new algorithm for determining the polynomial representation of functions over Galois fields. The algorithm is tailored for fields that can have feasible implementation with presently available MVL circuits.

Any function over Galois field, $f: GFk \rightarrow GFk$, where $k=p^n$, any power of a prime p, can be represented [3] in the form of a polynomial:

$$F(x) = \sum_{i=0}^{k-1} C_i x^i \qquad \text{(EQ 1)}$$

Benjahurit and Reed [1] proposed the use of Menger theorem [7] in order to determine the coefficients of the representing polynomial in the following form:

$$C_i = \sum_{g \neq 0} [F(0) - F(g)]\, g^{-i} \qquad \text{(EQ 2)}$$

With this method we have to make summations over all the elements for every coefficient of the polynomial.

We have investigated if there is a computationally more efficient method for the fields that can be used in practice with circuits described in previous sections. These are the fields of size 2, 3 and 4.

4.1 Direct Method for Polynomial Determination

Our approach is based on solving the system of linear equations directly and employing some algebraic properties common to fields that we are considering in order to get a computationally more effective algorithm.

Suppose that a given function has values $F_0, F_1, ..., F_t$ at points $0, 1, ..., t$. We want to have the function represented as a polynomial with coefficients $C_0, C_1, ..., C_t$.

Then, for a polynomial that fits these points, we have the system of linear equations:

$$C_0 + C_1 0 + C_2 0^2 + \ldots + C_t 0^t = F_0$$
$$C_0 + C_1 1 + C_2 1^2 + \ldots + C_t 1^t = F_1$$
$$\ldots$$
$$C_0 + C_1 t + C_2 t^2 + \ldots + C_t t^t = F_t$$

Solving this system in a conventional way is inefficient in terms of the number of operations required, compared to the application of Menger's theorem. However, we can exploit the properties of the fields under consideration to reduce the total number of operations needed.

Consider the following manipulation. It is obvious that coefficient C_0 is equal to the value of the function at point zero, F_0, from the first equation. Hence, we can subtract this equation from all the others, eliminating the coefficient C_0. After that, we can subtract any equation from all the others, to decrease the size of the problem. For example, after subtracting the equation defining F_1-F_0, the following system is obtained:

$$(2-1)C_1 + (2^2 - 1^2)C_2 + \ldots + (2^t - 1^t)C_t = F_2 - F_1$$
$$(3-1)C_1 + (3^2 - 1^2)C_2 + \ldots + (3^t - 1^t)C_t = F_3 - F_1$$
$$\ldots$$
$$(t-1)C_1 + (t^2 - 1^2)C_2 + \ldots + (t^t - 1^t)C_t = F_t - F_1$$

Now, for any two elements a and b in Galois fields that we are considering, namely of sizes 2,3 and 4, the following is true:

$$(a \pm b)^n = a^n \pm b^n, (0 < n < k-1)$$

Using this fact, each equation F_j -F_1 in the system can be divided by the term j-1, since $(j^2-1^2) = (j-1)^2$ leading to the following system:

$$C_1 + (2-1)C_2 + \ldots + (2-1)^{t-1}C_t = \frac{F_2 - F_1}{2-1}$$
$$C_1 + (3-1)C_2 + \ldots + (3-1)^{t-1}C_t = \frac{F_3 - F_1}{3-1}$$
$$\ldots$$
$$C_1 + (t-1)C_2 + \ldots + (t-1)^{t-1}C_t = \frac{F_t - F_1}{t-1}$$

By subtracting pairs of equations, a similar system of equations with a smaller number of unknowns is obtained. This procedure can be performed until there is only one unknown left. Then, it is straightforward to determine the rest of the unknowns.

The computation can be performed in a way similar to the Newton method of finite differences. A set of *formal differences* will be formed, in order to keep track of the right hand side of the equations, by the following definition:

$$[x_1 x_j] = \frac{F(x_j) - F(x_1)}{S_{1j}}$$

$$[x_1 x_i x_{i+1} \ldots x_{i+j-1}] = \frac{[x_1 x_{i+1} \ldots x_{i+j-1}] - [x_1 x_2 x_3 \ldots x_j]}{S_{1i(i+1)\ldots(i+j-1)}}$$

where the set of denominators S is determined by the following equations:

$$S_{1j} = j - 1$$
$$S_{1i(i+1)\ldots(i+j-1)} = S_{1(i+1)(i+2)\ldots(i+j-1)} - S_{12\ldots j}$$

The denominators are also used when calculating the coefficients C_l from the already known coefficients C_{l+1}, C_{l+2}, ... C_t, using the equation:

$$C_l = [x_1 x_2 \ldots x_l] - S_l C_{l+1} - S_l^2 C_{l+2} - \ldots - S_l^{t-1} C_t \quad \textbf{(EQ 3)}$$

where $S_l = S_{12\ldots l}$

In the case of a completely specified function, i.e. the one given by k points $(t = k - 1)$, a further simplification is possible. First, from the cyclicity property, it follows that $a^{k-1}=1$ for each element a of the field. This means that the last column of the starting system will be eliminated when subtracting any two equations for the first time. In order to recover this coefficient, we will employ the Menger formula. Indeed, by putting $i=k-1$ in Equation 2, we will get immediately the expression for the coefficient C_{k-1}:

$$C_{k-1} = -F_0 - F_1 - \ldots - F_{k-1} \quad \textbf{(EQ 4)}$$

4.2 The Performance

The total number of formal differences needed is $(k-2)(k-1)/2$. The total number of operations is the number of coefficients multiplied by 3, because we need to calculate the new divisor, the new difference in function values, and to divide these two numbers. So, to produce the difference table, we need $3/2k^2-9/2k+3$ operations.

The remaining part of the algorithm will produce the coefficients of the polynomial using Equation 3. In order to do this we have to perform

$$2+5+8+..+(2+3(k-2)) = 3/2(k-2)(k-3) + 2(k-2) = 3/2k^2 - 11/2k + 5$$

operations, since every new term will contribute three new operations, and in the beginning we need 2 operations for the first two coefficients. To determine the last coefficient we need k more operations, according to Equation 4. Adding all the operations, the total amounts to $3k^2-9k+8$ operations. This is somewhat better in comparison to Menger method which can be shown to require $3k^2-8k+6$ finite field operations. We note that Wesselkamper [9] proposed the use of Newton's method; his approach is general and it requires $7/2k^2-11/2k+2$ operations.

4.3 Functions of Two Variables

We will now describe how this algorithm can be extended to functions of two variables $f:GF^2k \rightarrow GFk$. To solve this problem directly, the k^2 equations have to be set up. The values of the function will be given by the matrix $F_{ij}:0<i,j<k$, and we will be looking for a set of coefficients $C_{ij}:0<i,j<k$ of the multinomial representing the function:

$$F(x_1, x_2) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} C_{ij} x_1^i x_2^j \qquad \text{(EQ 5)}$$

First, let us consider the case when x_2 is equal to zero. Then, we can set up a system of equations in which the set of $F_{00}, F_{10}, \ldots, F_{(k-1)0}$ is given, and will be looking for a set of coefficients $C_{00}, C_{10}, \ldots, C_{(k-1)0}$, since all other coefficients C_{ij} will be multiplied by zero and will not contribute to the equation. This as the starting one-dimensional problem which we can solve using the algorithm above. Also, we can consider the symmetrical case, i.e. when x_1 is equal to zero and find a solution for the additional k-1 coefficients, since the coefficient C_{00} is already known. To solve the system for the remaining $(k-1)^2$ coefficients, a decomposition approach will be used.

First, assign constant values *1, 2, ..., k-1* to x_2 to obtain *k-1* functions of x_1:

$$F(x_1, i) = D_{0i} + D_{1i} x_1 + \ldots + D_{(k-1)i} x_1^{k-1}$$

and the original function is represented as a set of *k-1* of functions of only one variable. The coefficients D_{ij} can be obtained by solving the *k-1* systems given above, as the one-dimensional problem. The next step is to obtain the coefficients C_{ij} from the values D_{ij}, which can be done from:

$$D_{li} = C_{l0} + iC_{l1} + \ldots + i^{k-1} C_{l(k-1)}$$

According to this equation, the values of D_{ij} are obtained by the same transformation by which the values of the function in the one dimensional case are obtained from the coefficients of the polynomial. Therefore, the same inverse operation, i.e. our one-dimensional algorithm can be employed again. After solving the next *k-1* systems, all the coefficients of the multinomial will be known. The total number of executions of the one-dimensional algorithm, using this approach, is equal to $2k$.

It can be shown that this scheme can be employed in the general case of n-variable functions and that the number of systems to solve is equal to nk^{n-1}.

5.0 Concluding Remarks

This paper has presented current-mode CMOS circuits that implement addition and multiplication in a Galois field with four elements. It has also presented a computationally efficient method for determination of polynomial representations of arbitrary functions. While the method is given in terms of 2-variable synthesis, it can be generalized to the n-dimensional case. The method is restricted to Galois fields of small size; but, this size corresponds to the cases that are interesting in practice, namely those implementable with today's MVL technology.

6.0 References

[1]Benjauthrit, B. and Reed, I. S., *Galois switching Functions and their Applications,* IEEE Trans. on Computers, Vol. ***C-25*** No. 1, pp. 79-86, January 1976.

[2] Chang, Y. H. and Butler, J. T., *The Design of Current Mode CMOS Multiple-Valued Circuits,* Proceedings of the 21st ISVML, pp. 130-138 , May 1991.

[3] Davio, M., Deschamps, J-P. and Thayse, A. , *Discrete and Switching Functions,* McGraw-Hill, Reading, Massachusetts, 1978.

[4] Ishizuka, O.,Takarabe, H. ,Tang, Z. and Matsutomo, H., *Synthesis of Current-Mode Pass Transistor Networks,* Proceedings of the 21st ISMVL, pp. 139-146 , May 1991.

[5] Lei, K. and Vranesic, Z., *On the Synthesis of 4-Valued Current Mode Circuits,* Proceedings of the 21st ISMVL, pp. 147-155, May 1991.

[6] Lidl, R. and Niederreiter, H., *Finite Fields,* Addison-Wesley, Reading, Massachusetts, 1983.

[7] Menger, K. S., *A Transform for Logic Networks,* IEEE Trans. on Computers, Vol. ***C-18*** No. 3, pp. 241-251, March, 1969.

[8] Dao, T. T., *Multiple-Valued IIL, Applications and Extensions,* Computer Science and Multiple-Valued Logic, D. C. Rine, ed., pp. 449-501, North-Holland, 1984.

[9] Wesselkamper, T. C. , *Divided Difference Method for Galois Switching Functions,* IEEE Trans. on Computers, Vol. ***C-27*** No. 3, pp. 232-238, Mar., 1978.