ARITHMETIC OF FINITE FIELDS *

T. R. N. RAO

DEPARTMENT OF COMPUTER SCIENCE
UNIV. OF SOUTHWESTERN LOUISIANA, LAFAYETTE, LA.

ABSTRACT: The arithmetic operations in finite fields and their implementation are important to the construction of error detecting and correcting codes. The addition, multiplication and division in the field GF(2^m) are implemented as polynomial operations using binary logic of flip-flops and EXOR's. For fields of non-binary characteristic, modular arithmetic (with modulus p, a prime) becomes important. This paper focuses on problems relating to the arithmetic of GF(p), and some recent results and new ideas on this topic are presented here.

I. INTRODUCTION AND BACKGROUND

Finite fields are often called Galois Fields. GF(q) denotes the Galois field of q elements and q must be of the form p^m for some prime p. Galois field arithmetic is employed extensively in the logic of error detection and correction (cyclic) codes [1,2]. The binary cyclic codes are defined in the algebra of polynomials over the field of two elements, namely, GF(2). The encoding and decoding logic of binary cyclic codes involves the addition, multiplication and division operations in the algebra of polynomials and operations of GF(2^m).

an mth degree extension of GF(2). The arithmetic of GF(2^m) is well known to coding theorists and logic designers but there are many interesting problems to be solved in the decoding of (multiple error correcting) cyclic codes [3].

The arithmetic of GF(p), a prime field, is important in the implementation of Reed-Solomon codes. If the arithmetic of GF(p) can be handled efficiently, then it is conceivable to obtain very efficient single and multiple error-correcting Reed-Solomon codes. We assume from the reader some background of finite field structure and Reed-Solomon codes [1,4]. We present arguments for the need for the development of suitable arithmetic logic in the prime fields, such as, GF(11) and GF(17). While the interest is in the arithmetic of GF(p), we focus special interest on primes of the form 2^m + 1. The motivation for that case is that each group or byte of m bits can be treated as an element of GF(p) if 2^m + 1 is a prime. For BCD numbers,
the base $b=10$ (and the radix $r=2$) but
each group of 4 bits representing a BCD
digit can also be treated as an element
of GF(11). By using that approach we
can construct Reed-Solomon codes
somewhat efficiently. Similarly
hexadecimal numbers can be treated as
elements of GF(17) and the error
correcting codes can be designed over
such a field.

II. REED-SOLOMON CODES

We consider here Reed-Solomon codes
generated by a polynomial of the form
\[
q(x) = (x-1)(x-a) \cdots (x-a^{d-2})
\]
\[
= \prod_{j=0}^{d-2} (x-a^j)
\]
where $a$ is an element of GF($p$) and $a$
has order $n$. That is, $a^n = 1$ for
smallest positive integer $n$. The
polynomial $g(x)$ generates a code of
length $n$ symbols (over GF($p$)) out of
which $n-d+1$ are information symbols and
d-1 are parity checks. The code has a
minimum distance of $d$ and is therefore
capable of detecting $(d-1)$ errors or
correcting \(\left\lfloor (d-1)/2 \right\rfloor\) errors. As
examples consider the Reed-Solomon
codes listed in Table I below.

<table>
<thead>
<tr>
<th>Code</th>
<th>$g(x)$</th>
<th>$p$</th>
<th>$n$</th>
<th>$k$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>$x-1$</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>$(x-1)(x-2)$</td>
<td>11</td>
<td>10</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>C3</td>
<td>$\frac{3}{x-1} \prod_{j=0}^{d-2} (x-a^j)$</td>
<td>11</td>
<td>10</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>C4</td>
<td>$(x-1)(x-3)$</td>
<td>17</td>
<td>16</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>C5</td>
<td>$\frac{3}{x-1} \prod_{j=0}^{d-2} (x-a^j)$</td>
<td>17</td>
<td>16</td>
<td>12</td>
<td>5</td>
</tr>
</tbody>
</table>

$g(x)$ represents the generator
polynomial and the number of check
symbols $n-k$ equals the degree of $g(x)$.
n is the code length and $k$ is the
number of information symbols. Each
code symbol is an element of GF($p$). The
Reed-Solomon cyclic codes over GF($p$)
will have a maximum code length of
$n=p-1$ if $(x-a)$ is a factor of $g(x)$ and
a is primitive in GF($p$). The codes C2
and C3 have $(x-2)$ as a factor and 2 is
primitive in GF(11). For codes C4 and
C5, $(x-3)$ is a factor of $g(x)$ and 3 is
primitive in GF(17). The minimum
distance, $d_{min}$ of the codes is related
to the number of factors in $g(x)$. If
$$g(x) = \prod_{i=0}^{d-1} (x-a^i), \text{ then } d_{min} \geq d$$

With only $d-1$ parity symbols, the codes have a minimum distance of $d$ and, in that sense, the Reed-Solomon codes are maximum distance separable and the information rates of these codes are very good.

Another important consideration in choosing $GF(p)$, instead of $GF(2^m)$, is the decoding logic. The roots of polynomials over $GF(p)$ can be obtained through explicit formulas rather than by a search or iteration. Finding the roots of polynomials over binary based fields (i.e. $GF(2^m)$) through explicit formulas is not known presently and a preliminary effort in this direction is appearing [3].

III. ARITHMETIC MODULO $2^n + 1$.

Not all integers of the form $2^n + 1$ are primes. However Fermat primes [5] are of the form
$$F_m = 2^{2^m} + 1$$
for $m=0,1,2,3,$ and 4.

Although Fermat conjectured that $F$ for all $n$ are primes it was shown for $n=5$, the Fermat number $2^{2^5} + 1$ is found to be composite. Our interest here will be restricted to Fermat primes, namely, 1, 5, 17, 257. There has been some interest in the modulo $2^n + 1$ arithmetic logic [5-7]. A novel format to represent $GF(2^n + 1)$ is derived in [6] as follows.

For simplicity we let $p = 2^n + 1$ and use $GF(p)$ instead of $GF(2^n + 1)$. The elements of $GF(p)$ cannot be represented as $n$-tuples. Therefore each $X \in GF(p)$ is represented by a binary $(n+1)$-tuple of the form
$$X = (x_0, x_1, \ldots, x_n, I)$$
where $x_i$ has the usual weight of 2 and $I$ has a weight of 1, the same as $x_0$. Here $I$ is called the zero indicator
and equals zero iff $X = \emptyset$.

Hence

$$X = I_x \left( \sum_{i=0}^{n-1} x^i \right).$$

Setting

$$x = \sum_{i=0}^{n-1} x^i,$$

we get

$$X = I_x (x + 1).$$

Using this representation it was shown [6] that addition and complementation operations can be obtained with only a minor modification to 1's complement logic. It is also easy to implement scaling operations i.e. multiplication or division by 2. However, further work is required to find efficient algorithms to multiply or divide numbers modulo p. That should lead to fast encoding and decoding logic for efficient multiple error correcting Reed-Solomon codes.

REFERENCES.


