Efficient Arithmetic in $GF(2^n)$ through Palindromic Representation

Ian F. Blake, Ron M. Roth, Gadiel Seroussi
Visual Computing Department
HPL-98-134
August, 1998

finite field representation, optimal normal basis, palindromic representation

A representation of the field $GF(2^n)$ for various values of n is described, where the field elements are palindromic polynomials, and the field operations are polynomial addition and multiplication in the ring of polynomials modulo $x^{2n+1}-1$. This representation can be shown to be equivalent to a field representation of Type-II optimal normal bases. As such, the suggested palindromic representation inherits the advantages of two commonly-used representations of finite fields, namely, the standard (polynomial) representation and the optimal normal basis representation. Modular polynomial multiplication is well suited for software implementations, whereas the optimal normal basis representation admits efficient hardware implementations. Also, the new representation allows for efficient implementation of field inversion in both hardware and software.

Internal Accession Date Only

1 Introduction

In this note, we consider representations of extension fields $GF(2^n)$ that lend themselves to efficient arithmetic implementation over the binary field $GF(2)$. Familiarity with basic concepts of finite field theory is assumed; these facts can be recalled, for instance, from [1].

The finite field $GF(2^n)$ is a vector space of dimension n over binary field $GF(2)$. As such, it can be represented using any basis of n linearly independent elements of $GF(2^n)$ over $GF(2)$. Therefore, elements of $GF(2^n)$ are represented by binary vectors of length n. Field addition is realized in all bases by a bit-wise exclusive OR (XOR) operation, whereas the structure of field multiplication is determined by the choice of basis for the representation.

Two families of bases are commonly used to represent the field $GF(2^n)$:

Standard (polynomial) representation:

The basis elements have the form $1, \omega, \omega^2, \ldots, \omega^{n-1}$, where ω is a root in $GF(2^n)$ of an irreducible polynomial $P(x)$ of degree n over $GF(2)$. In an equivalent interpretation of this representation, the elements of $GF(2^n)$ are polynomials of degree at most $n-1$ over $GF(2)$, and arithmetic is carried out modulo an irreducible polynomial $P(x)$ of degree n over $GF(2)$.

Optimal normal basis (ONB) representation:

The basis elements have the form $\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}$ for a certain element $\alpha \in GF(2^n)$. This defines a *normal basis*. In addition, if for all $0 \leq i_1 \neq i_2 \leq n-1$ there exist j_1, j_2 such that, $\alpha^{2^{i_1}+2^{i_2}} = \alpha^{2^{j_1}} + \alpha^{2^{j_2}}$, the basis is called *optimal*. The element α is called the *generator* of the basis. Optimal normal bases exist for an infinite subset of values of n defined below.

The standard representation lends itself to efficient *software* implementations of the field arithmetic. In particular, multiplication can be made very efficient if the polynomial $P(x)$ is sparse, and inversion can be realized using the extended Euclidean algorithm. On the other hand, the ONB representation allows for efficient *hardware* implementations of field multiplication (see [2, Ch. 5]). Inversion, however, remains a difficult operation in this case.

Large finite fields are the basis of many modern cryptographic algorithms, e.g., in elliptic curve cryptography. In these applications, the field arithmetic is the computational bottleneck, and efficient implementations are essential. On the other hand, as the use of

cryptography becomes widespread, hardware and software implementations are required to inter-operate and use common representations.

It is known [2, Ch. 5] that ONB's exist in $GF(2^n)$ only in the following cases:

Type-I ONB: $n+1$ is a prime p and and 2 is primitive modulo p (namely, the multiplicative order of 2 modulo p is n).

Type-II ONB: $2n+1$ is a prime p and either

(i) 2 is primitive modulo p, or —

(ii) $p \equiv 3 \pmod 4$ (i.e., -1 is a quadratic nonresidue modulo p) and the multiplicative order of 2 modulo p is n (namely, 2 generates the quadratic residues modulo p).

Type-I ONB's are generated by elements $\alpha \in GF(2^n)$ of order $p = n+1$. Observe that the minimal polynomial of α is $f(x) = x^n + x^{n-1} + \cdots + x + 1$ and the sets $\{\alpha, \alpha^2, \alpha^{2^2}, \ldots, \alpha^{2^{n-1}}\}$ and $\{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^n\}$ are identical. Thus, after suitable permutation, we can operate on elements in ONB representation as polynomials modulo $f(x)$, or even simpler, modulo the *very sparse* polynomial $(x+1)f(x) = x^{n+1} + 1$. The latter will give results expressed in terms of $1, \alpha, \alpha^2, \ldots, \alpha^n$, which are brought back to the desired basis set by using, when needed, the equality $1 = \alpha + \alpha^2 + \cdots + \alpha^n$. So, in addition to being attractive for hardware applications, the Type I ONB representation inherits the advantages of the polynomial representation.

The field representation that we suggest in this note demonstrates how such an advantage can be obtained also for values of n for which a Type-II ONB exist. In our representation, the field elements will be a subset of the polynomials of degree at most $2n$ over $GF(2)$, and the arithmetic will be carried out modulo the (very sparse) polynomial $x^{2n+1} - 1$. On the other hand, our representation can be shown to be equivalent, up to a simple bit permutation and replication operation, to the Type-II ONB representation.

2 Palindromic representation of finite fields

We hereafter assume that $2n+1$ is a prime p and either condition (i) or (ii) of the Type-II ONB holds. For such values of n, let γ be a pth root of unity in $GF(2^{2n})$. It is known that $\alpha = \gamma + \gamma^{-1}$ generates a Type-II ONB [2, Section 5.2].

Let Φ denote the vector space of all polynomials over $GF(2)$ of the form $a(x) = \sum_{i=1}^{2n} a_i x^i$, where $a_i = a_{p-i}$ for $i = 1, 2, \ldots, n$. We call such polynomials *palindromic polynomials*. In our *palindromic representation* of $GF(2^n)$, each field element is represented

as a palindromic polynomial. Addition is defined as the ordinary polynomial addition of elements in Φ, and the product of two palindromic polynomials $a(x), b(x) \in \Phi$ is the unique polynomial $c(x) \in \Phi$ such that

$$c(x) \equiv a(x) \cdot b(x) \pmod{x^p - 1} . \tag{1}$$

Equation (1) suggests that multiplication can be implemented for the palindromic representation using standard modular polynomial multiplication.

When we substitute $x = \gamma$ in $a(x)$, we obtain

$$a(\gamma) = \sum_{i=1}^{2n} a_i \gamma^i = \sum_{i=1}^{n} a_i (\gamma^i + \gamma^{-i}) .$$

It follows from conditions (i) and (ii) above that for every $i \in \{1, 2, \ldots, n\}$, exactly one element in the pair $\{i, p-i\}$ can be written as 2^j modulo p, for some $0 \leq j \leq n-1$. Hence, we can write

$$a(\gamma) = \sum_{j=0}^{n-1} a_{2^j} (\gamma^{2^j} + \gamma^{-2^j}) = \sum_{j=0}^{n-1} a_{2^j} (\gamma + \gamma^{-1})^{2^j} = \sum_{j=0}^{n-1} a_{2^j} \alpha^{2^j} , \tag{2}$$

where all indexes are taken modulo p. It follows from (2) that, up to permutation, the elements $a_1, a_2, \ldots, a_n$ are the coefficients in the normal basis representation of $a(\gamma)$ that corresponds to the generator α. This simple relationship between the coefficients of $a(x)$ and the normal basis representation of $a(\gamma)$ implies that, in a hardware implementation, an efficient optimal normal basis representation multiplier can be used for the palindromic representation, provided the coefficients are permuted accordingly (this has no hardware cost, other than "re-wiring"). In software, we would use the polynomial interpretation induced by (1), and benefit from the efficient algorithms available in that case.

As for inversion, the palindromic representation allows for the use of the extended Euclidean algorithm to find the inverse of the palindromic polynomial $a(x)$ modulo $x^p - 1$, from which the inverse in ONB representation is easily derived. The Euclidean algorithm admits efficient implementations in both hardware and software.

3 References

[1] R. Lidl and H. Niederreiter, *Finite Fields,* in *Encyclopedia of Mathematics and its Applications,* G.-C. Rota, editor, Addison-Wesley, 1983.

[2] A.J. Menezes (Ed.), I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone, T. Yaghoobian, *Applications of Finite Fields,* Kluwer, Boston, 1993.