

Arithmetic Operations in a Power-of-Two Galois Field

Most modern 2D matrix symbologies use Reed-Solomon encoding in a power-of-two Galois field (also known as an “extension field”) to provide both error detection and correction. The closed field arithmetic needed both to encode and decode the Reed-Solomon blocks, described in the specifications as “bit-wise modulo 2 and word-wise modulo P”, is quite different from everyday arithmetic, and even from the straight modulo arithmetic used in prime Galois fields. This can make it difficult to replicate the encoding examples found in those specifications.

This paper attempts to teach the appropriate operations by building upon C-language encoding functions published in several AIM standards (e.g., “ISS - Aztec Code”). For illustration we will take the widely-used case of GF(256), the Galois field for 2^8 , with a “prime polynomial” of $x^8 + x^5 + x^3 + x^2 + 1$ whose equivalent value P is binary 100101101 or decimal 301. [Several possible prime polynomials can create a GF(256); 301 is the value selected for use with all GF(256) symbologies except QR Code, which uses 285.]

The nature of closed-field arithmetic is that all operations between values in the field produce another value in the field. GF(256) for example contains all the integers between 0 (zero) and 255. The “bitwise modulo 2” aspect of this arithmetic is best seen in the Addition and Subtraction operations, which BOTH are a bit-wise exclusive-or of the two numbers. For example:

given:	A =	141 ₁₀ =	10001101 ₂
and:	B =	43 ₁₀ =	00101011 ₂
then:	A + B =	A - B =	10100110 ₂ (= 166 ₁₀ , not the everyday result!).

Again, note that both the sum and difference between two numbers in GF(2^N) arithmetic is the exclusive-or of their corresponding bits, expressed in C by the caret, namely “A ^ B”. Thus they are realized in the following two functions:

```
int Sum (int A, int B) { return (A ^ B); }
int Difference (int A, int B) { return (A ^ B); }
```

[That the Sum and Difference are identical functions has the odd but true corollary that in such a Galois field each number is its own negative! That is, A equals minus A... which of course works out because then the sum of “A” and “-A” is the exclusive-or of the same two values which equals zero, as it must!]

Multiplication and Division operations within GF(2^N) are performed using Log and Antilog tables that are generated by the following C routine:

```
#define GF 256 // define the Size & Prime Polynomial of this Galois field
#define PP 301
int Log[GF], ALog[GF]; // establish global Log and Antilog arrays

// fill the Log[] and ALog[] arrays with appropriate integer values
void FillLogArrays (void) {
    int i;
    Log[0] = 1-GF; ALog[0] = 1;
    for (i=1; i<GF; i++) {
        ALog[i] = ALog[i-1] * 2;
        if (ALog[i] >= GF) ALog[i] ^= PP;
        Log[ALog[i]] = i;
    }
}
```

Then the Product of two values is the antilog of the mod (GF-1) sum of their logs, namely:

```
int Product (int A, int B) {
    if ((A == 0) || (B == 0)) return (0);
    else return (ALog[(Log[A] + Log[B]) % (GF-1)]);
}
```

...and the Quotient of two values is the antilog of the mod (GF-1) difference between their logs:

```

int Quotient (int A, int B) { // namely A divided by B
  if (B == 0) return (1-GF); // signifying an error!
  else if (A == 0) return (0);
  else return (ALog[(Log[A] - Log[B] + (GF-1)) % (GF-1)]);
}

```

These log and antilog operations closely resemble everyday arithmetic, but with modulo operations to keep all values between 0 and GF-1, and the results are distinctly different! For example, the product of 14 and 33 is not 206, the result from everyday modulo 256 arithmetic, but instead:

given: $\text{Log}[14] = 54$
 and: $\text{Log}[33] = 219$
 then: $(54 + 219) \bmod 255 = 273 \bmod 255 = 18$
 thus: $\text{ALog}[18] = 227$ (again, not the expected product!)

Employing these five routines - Sum(A,B), Difference(A,B), FillArrays(), Product(A,B), and Quotient(A,B) - will allow the encoding examples in the matrix symbology specifications to be confirmed.

Reference Books:

The following books include good references to assist on comprehension of this problem. Some of these books are no longer in print but may be found in libraries or on services such as Ebay etc.

- Error Coding Cookbook, C. Britton Rorabaugh, McGraw-Hill, ISBN 0-07-911720-1
- Blahut, Richard, E, "Principles and Practice of Information Theory", (c) 1987 by Addison-Wesley Publishing Company, Reading, MA, Reprinted with corrections, 1991. ISBN 0-201-10709-0
- Practical Error Correction, N. Glover & T. Dudley, Data Systems Technology, Corp., PO Box 1205, Bloomfield, Colorado, 80020, USA, +1 303 466-5228
- Error-Correction Coding for Digital Communications, G. Clark, Jr. & J. Cain, Plenum Press, NY, 233 Spring Street, New York, NY 10013, TK 5102.5.C52, ISBN 0-306-40615-2
- Error Control Coding: Fundamental and Applications, by Shu Lin and Daniel J. Costello, Jr., Prentice Hall, Inc., Englewood Cliffs, N.J. 07632, ISBN 0-13-283796-X

© AIM Inc. 2001

Published by:
 AIM Inc.
 634 Alpha Drive
 Pittsburgh, PA, 15238
 +1 412 963-8588
aidc@aimglobal.org
<http://www.aimglobal.org>

v1.0 Sep 22, 2001