MEMO: GFAU System Design Document
DATE: May 23, 2018
TO: Dr. E.F. Charles LaBerge
SUBJECT: Mathematical Background on the Galois Field Arithmetic Unit

1 Background

Galois fields (denoted $GF(p^n)$, where $p,n \in \mathbb{N}_{>0}$, and p is prime) are fields with finite orders of p^n. They are a key part of number theory, abstract algebra, arithmetic algebraic geometry, and cryptography. In error detection and correction, Galois fields are utilized in cyclic redundancy check (CRC) which are used in digital networks and storage devices to detect accidental changes to raw data.

The Galois Field Arithmetic Unit (GFAU) is a scalable arithmetic logic unit (ALU) capable of generating elements in the Galois field of an irreducible polynomial in $GF(2^n)$, where $2 \leq n <$ (# of memory address pins). GFAU supports addition, subtraction, multiplication, division and logarithm of these elements for low powered devices.

1.1 Purpose and Scope

This document will demonstrate the functionality of the unit by deriving all its functionality through mathematical approaches. The mathematical algorithms will be accompanied by the modules that implement them in digital design. The modules will contain brief documentation on their usages.

1.2 Terms and Keywords

1.2.1 Input Primitive Polynomials

Input primitive polynomials in the Galois Field are represented as:

$$c_n x^n + \ldots + c_2 x^2 + c_1 x^1 + c_0 x^0, \text{ where } c,x \in \mathbb{Z}_2$$

For convenience and simplicity, all the examples provided will refer to the polynomial: $x^3 + x^2 + x^0$.

1.2.2 Elements

The elements of an input polynomial refer to the $2^n - 1$ elements in the field.

1.2.3 Polynomial Form

The polynomial forms of the elements refer to the $2^n - 1$ symbolic representations of the input primitive polynomials in the field.

1.2.4 Example

An example of the elements and their corresponding polynomials is provided below:

Table 1: The 8 Element Vectors of $x^3 + x^2 + x^0$ in $GF(2)[x]$

Element	Symbol	Polynomial Form	Symbol
0 (NULL)	[1]	$0+0+0$	000
β^0	000	$0+0+\beta^0$	001
β^1	001	$0+\beta^1+0$	010
β^2	010	β^2+0+0	100
β^3	011	$\beta^2+0+\beta^0$	101
β^4	100	$\beta^2+\beta^1+\beta^0$	111
β^5	101	$0+\beta^1+\beta^0$	011
β^6	110	$\beta^2+\beta^1+0$	110
β^7	[2]	$0+0+\beta^0$	001

[1] The additive identity, 0 (zero), referred to as NULL, in its element form is reserved where its binary symbol does not represent its decimal value.

[2] Elements beyond the $(2^n - 1)$th element will be handled with special conditions since they cycle back to previous elements.

2 Design

Input polynomials will be represented as n-bit binary-coded decimal (BCD) arrays.

For example, the polynomial $x^3 + x^2 + x^0$ will be represented in a system with 16-bit data words as

$$< 0000\ 0000\ 0000\ 1101 > \text{(3rd, 2nd, and 0th bits)}$$

2.1 Primitive Polynomial

A polynomial is said to be irreducible if and only if there exists no roots for it. A primitive polynomial is an irreducible polynomial that generates all elements of an extension field from a base field. [1]

Determining irreducibility of a polynomial can be achieved by performing exclusive disjunctions on all the bits of the polynomial vector.

$$\text{irreducible} = \begin{cases} true, & \text{if } < v_n \oplus \ldots \oplus v_2 \oplus v_1 \oplus v_0 >= 1 \\ false, & \text{if } < v_n \oplus \ldots \oplus v_2 \oplus v_1 \oplus v_0 >= 0 \end{cases}$$

However, determining if a polynomial is primitive requires precomputing the elements and checking those that are prime factors to $2^n - 1$ in a brute forced manner.

To determine if an irreducible polynomial of degree n is primitive, $2^n - 1$ must be the smallest element in its field to satisfy

$$x^{2^n - 1} \equiv 1 \pmod{p(x)}$$

If the prime factors, $\{p_0^{\alpha_0}, p_1^{\alpha_1}, \ldots, p_k^{\alpha_k}\}$, of $2^n - 1$ are known, it is sufficient to check that none of them would satisfy the condition listed above.

2.1.1 Modules

Checking for primitive polynomials is beyond the scope of the project. Therefore, modules for checking irreducibility and primitivity are not implemented in the design.

2.2 Symbols

Once a polynomial is determined primitive, its symbols may be generated. The order of the finite field generated by a primitive polynomial of degree n is $2^n - 1$.

Proof of the generation of the elements is not provided in this document. An example is provided in the appendix, and additional resources for formal proofs will be provided.

2.2.1 Modules

The `generator` module is responsible for generating all the elements in the field of the input polynomial. GFAU supports lookup of the elements by their element and polynomial forms. This conversion is achieved by the finite state machines (FSM) inside the `generator` swapping the memory addresses and data to the corresponding memory blocks.

For more details on the memory interface, see Memory Interface.

The design uses the following recurrence relation to generate all the elements:

$$\begin{aligned}\alpha^{n+m} &= \alpha^{n+(m-1)} \times \alpha^n \\ &= \begin{cases} \alpha^{n+(m-1)} \ll 1 & \text{if } \alpha^{n+(m-1)}[n-1] = 0 \\ (\alpha^{n+(m-1)} \ll 1) \oplus \alpha^n & \text{if } \alpha^{n+(m-1)}[n-1] = 1 \end{cases}\end{aligned}$$

Table 2: Generated Symbols for Primitive Polynomials of Degree n in Memory with $m > n$ Address Pins

Element	Polynomial Form	Symbol
0	$0_m + \ldots + 0_n + \ldots + 0_2 + 0_0 + 0_0$	$< \overleftarrow{0_m \ \ldots \ 0_n} \ \ldots \ 0_2 0_1 0_0 >$
α^0	$0_m + \ldots + 0_n + \ldots + 0_2 + 0_0 + \alpha_0^0$	$< \overleftarrow{0_m \ \ldots \ 0_n} \ \ldots \ 0_2 0_1 1_0 >$
α^1	$0_m + \ldots + 0_n + \ldots + 0_2 + \alpha_1^1 + 0_0$	$< \overleftarrow{0_m \ \ldots \ 0_n} \ \ldots \ 0_2 1_1 0_0 >$
α^2	$0_m + \ldots + 0_n + \ldots + \alpha_2^2 + 0_1 + 0_0$	$< \overleftarrow{0_m \ \ldots \ 0_n} \ \ldots \ 1_2 0_1 0_0 >$
$\ldots$	$\ldots$	$\ldots$
α^{n-1}	$0_m + \ldots + \alpha_{n-1}^{n-1} + \ldots + 0_2 + 0_1 + 0_0$	$< 0_m \ \ldots \ 1_{n-1} \ \ldots \ 0_2 0_1 0_0 >$
α^n	$0_m + \ldots + \alpha_{n-1}^{n-1} + \ldots + \alpha_2^2 + \alpha_1^1 + \alpha_0^0$	$< 0_m \ \ldots \ x_{n-1} \ \ldots \ x_2 x_1 x_0 >$
α^{2^n-1}	$0_m + \ldots + 0_n + \ldots + 0_2 + 0_0 + \alpha_0^0$	$< \overleftarrow{0_m \ \ldots \ 0_n} \ \ldots \ 0_2 0_1 1_0 >$

2.3 Operations

GFAU supports addition, subtraction, multiplication, division and logarithm of elements in Galois fields. The `operators` module consists of the `addsub`, `mul`, and `div` modules that are responsible for their corresponding binary operations. Additional helper module, `maskedtwoscmp`, is included to compute the two's complement of the second operand required for division. The other two modules, `outselect` and `outconvert` act as advanced multiplexers to determine the output of the operation requested.

2.3.1 Addition and Subtraction

Binary addition and binary subtraction are synonymous in Galois fields. Addition and subtraction of Galois operands may be done by computing the bitwise exclusive disjunction of the operands.

$$\begin{aligned}
\alpha^i \pm \alpha^j &= \{x_{i,n}, \ldots x_{i,2}, x_{i,1}, x_{i,0}\} \pm \{x_{j,n}, \ldots x_{j,2}, x_{j,1}, x_{j,0}\} \\
&= \{(x_{i,n} \oplus x_{j,n}), \ldots, (x_{i,2} \oplus x_{j,2}), (x_{i,1} \oplus x_{j,1}), (x_{i,0} \oplus x_{j,0})\} \\
&= \alpha^k
\end{aligned}$$

2.3.1.1 Modules

Only polynomial forms of inputs are valid for these operations.

The implementation of Galois addition and subtraction is computed with a single-level parallel array of XOR gates.

2.3.2 Multiplication

Binary multiplication of Galois operands is congruent to the sum of the indices of the operands. If the indices sum to greater than or equal to $2^n - 1$, then $2^n - 1$ is subtracted from the sum to prevent overflow.

$$\begin{aligned}\alpha^i \cdot \alpha^j &= \{x_{i,n-1}, \ldots, x_{i,2}, x_{i,1}, x_{i,0}\} \cdot \{x_{j,n-1}, \ldots, x_{j,2}, x_{j,1}, x_{j,0}\} \\ &= \alpha^{(i+j) \pmod{(2^n-1)}} \\ &= \begin{cases} \alpha^{(i+j)-(2^n-1)} & \text{if } (i+j) \geq 2^n - 1 \\ \alpha^{(i+j)} & \text{if } (i+j) < 2^n - 1 \end{cases}\end{aligned}$$

2.3.2.1 Modules

Only element forms of inputs are valid for this operation.

Galois multiplication requires multiple binary additions and condition checks. To find the product of α^i and α^j, an adder may be used to sum the elements $i + j$. The most significant bits of the sum and the sum $+1$ will then be OR-ed to compute a single-bit control signal. The control signal will be multiplexed with a binary 1 to be added to the sum.

$$\begin{aligned}\alpha^i \cdot \alpha^j &= \{x_{i,n-1}, \ldots, x_{i,2}, x_{i,1}, x_{i,0}\} \cdot \{x_{j,n-1}, \ldots, x_{j,2}, x_{j,1}, x_{j,0}\} \\ &= \begin{cases} \alpha^{(i+j)} & \text{if } \Big((i+j)[n] \ \vee \ (i+j+1)[n]\Big) = 0 \\ \alpha^{(i+j+1)} & \text{if } \Big((i+j)[n] \ \vee \ (i+j+1)[n]\Big) = 1 \end{cases}\end{aligned}$$

2.3.3 Division

Binary division of Galois operands is congruent to the difference of the indices of the operands. If the difference is negative, then the absolute value of the difference is subtracted from $2^n - 1$ to prevent

underflow. If the difference is zero, then the quotient is α^0.

$$\begin{aligned}\alpha^i/\alpha^j &= \{x_{i,n-1},\ldots x_{i,2},x_{i,1},x_{i,0}\}/\{x_{j,n-1},\ldots x_{j,2},x_{j,1},x_{j,0}\}\\ &= \alpha^{(i-j)\ (\mathrm{mod}\ (2^n-1))}\\ &= \begin{cases}\alpha^{(2^n-1)-(j-i)} & \text{if } (i-j)<0\\ \alpha^{(i-j)} & \text{if } (i-j)>0\\ \alpha^0 & \text{if } i=j,\ i\neq 0\\ ERROR & \text{if } j=0\end{cases}\end{aligned}$$

2.3.3.1 Modules

Only element forms of inputs are valid for this operation.

Galois division requires multiple binary additions and condition checks. To find the quotient of α^i and α^j, the two's complement of j has to first be summed with i. Converting to two's complement may be done with parallel inverters and an adder. The most significant bit of the sum will then be used as a control signal to a multiplexer. If activated, the multiplexer will add the two's complement of a binary 1 to the sum.

$$\begin{aligned}\alpha^i/\alpha^j &= \{x_{i,n-1},\ldots,x_{i,2},x_{i,1},x_{i,0}\}/\{x_{j,n-1},\ldots,x_{j,2},x_{j,1},x_{j,0}\}\\ &= \begin{cases}\alpha^{(i+j_{2's})} & \text{if } (i+j_{2's})[n]=1\\ \alpha^{(i+j_{2's}+1_{2's})} & \text{if } (i+j_{2's})[n]=0\end{cases}\end{aligned}$$

2.3.4 Logarithm

Logarithm is considered a unary operation in the Galois field, where only one operand is required. Logarithm here refers to the degree of the operand.

$$log_\alpha(\alpha^i) = \deg(\alpha^i) = i$$

2.3.4.1 Digital Design

Only element forms of inputs are valid for this operation.

The logarithm of a Galois operand is the index of the term itself.

2.4 Arithmetic Exceptions

2.4.1 Zero

Zero, or the null symbol, is the numerical zero used to fulfill its role as the additive identity in the Galois field. The following operations demonstrate the characteristics of zero in the Galois field.

Table 3: Operations with $\mathbf{0}$ in $GF(2)[x]$

Operand 1	Operation	Operand 2	Output
$\mathbf{0}$	$+/-$	β^x	β^x
β^x	$+/-$	$\mathbf{0}$	β^x
$\mathbf{0}$	$\times$	β^x	$\mathbf{0}$
β^x	$\times$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{0}$	$\div$	β^x	$\mathbf{0}$
β^x	$\div$	$\mathbf{0}$	$ERROR$
$\mathbf{0}$	deg	$DON'TCARE$	$ERROR$

2.4.1.1 Digital Design

To handle zero, multiplexers are used to decode the intended operation to determine the forms if the inputs. Once the form is determined, the module raises a `zero flag` which is handled by individual operations to raise exceptions.

Table 4: Operations with $\mathbf{0}$ in $GF(2)[x]$

Memory Block	Address	Data
$\mathbf{0}$	$< \overleftarrow{1_{m-1} \ \dots \ 1_2} 1_1 1_0 >$	$< \overleftarrow{0_{m-1} \ \dots \ 0_2} 0_1 0_0 >$
$\mathbf{1}$	$< \overleftarrow{0_{m-1} \ \dots \ 0_2} 0_1 0_0 >$	$< \overleftarrow{1_{m-1} \ \dots \ 1_2} 1_1 1_0 >$

2.4.2 Membership

Elements greater than or equal to the order of a generated field are not considered members of the field. Although any element α^i may be represented as a member element by computing $\alpha^i \pmod{2^{n-1}}$, GFAU avoids the step and generates an error to notify the user.

2.4.2.1 Digital Design

Although not a current feature, non-member elements may easily be computed as member elements by the existing `mul` module. Utilizing the output of `mul`$(NULL, \alpha^i)$ will guarantee the proper operations of all elements.

3 Control Unit

The `control unit` module utilizes 6-bit opcodes to determine the operation selected. The truth table of the opcodes are provided below:

Table 5: Truth Table of 6-bit Opcodes Provided to the Control Unit. IO type 0 is Element Form and 1 the Polynomial Form of an Operand or Output

Operation	IO type	Description
000	$-,-,-$	Generate the elements in the field
001	$0/1,0/1,0/1$	Add operands
010	$0/1,0/1,0/1$	Multiply operands
011	$0/1,0/1,0/1$	Divide operands
100	$0/1,-,0/1$	Compute the logarithm of an operand
101	$-,-,-$	Set mode

References

[1] E. W. Weisstein. Primitive polynomial. *MathWorld – A Wolfram Web Resource*, Dec 2017.