Computers Math. Applic. Vol. 35, No. 6, pp. 27–31, 1998

Polynomial Division Using Left Shift Register

P. SARKAR, B. K. ROY AND P. P. CHOUDHURY
Computer Science Unit, Indian Statistical Institute
203, B.T. Road, Calcutta 700035, India

R. BARUA
Stat/Math Unit, Indian Statistical Institute
203, B.T. Road, Calcutta 700035, India

(Received December 1996; accepted October 1997)

Abstract—In this short note, we describe a simple polynomial division circuit based on a left shift register. The circuit essentially performs the modulo operation $f(x) \bmod p(x)$. It is shown how the same circuit can be used to perform $f(x)g(x) \bmod p(x)$. Applications to standard basis multiplication and encoding and decoding of systematic cyclic codes are also described.

Keywords—VLSI, Shift register, Polynomial division, Finite field arithmetic.

1. INTRODUCTION

Polynomial division over $GF(2)$ are of fundamental importance in designing circuits for error correcting codes. Presently, this is done using Linear Feedback Shift Register (LFSR) [1,2]. In this short note, we describe simple circuits using a left shift register to perform the operations $f(x) \bmod p(x)$ and $f(x)g(x) \bmod p(x)$. The main advantage of our scheme is that the circuit for performing $f(x) \bmod p(x)$ (which we will call the MOD circuit) is independent of $p(x)$ and the same circuit can be used for dividing by another polynomial of the same degree (requiring only a change in stored values). Moreover, this MOD circuit can also be used to perform $f(x)g(x) \bmod p(x)$ for arbitrary polynomials $f(x)$ and $g(x)$. For corresponding LFSR circuits which perform modulo multiplication, $p(x)$ along with one of $f(x)$ or $g(x)$ must be fixed. This flexibility allows the MOD circuit to be used as a standard basis multiplier. The trade-off being that our circuit require additional flip-flops and EX-OR gates. A second advantage is that compared to the corresponding LFSR circuit, the MOD circuit takes less clock cycles to perform the operation $f(x) \bmod p(x)$.

In what follows, all operations are over $GF(2)$.

2. POLYNOMIAL DIVISION

Let $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0$ be a fixed polynomial of degree n.
Then, $x^n \pmod{p(x)} = a_{n-1}x^{n-1} + \cdots + a_0$.
Let $x^{n+\alpha} \pmod{p(x)} = c_{n-1}x^{n-1} + \cdots + c_0$, $\alpha \geq 0$.

Then,

$$\begin{aligned} x^{n+\alpha+1} \pmod{p(x)} &= c_{n-1}x^n + \cdots + c_0 x \\ &= c_{n-1}(a_{n-1}x^{n-1} + \cdots + a_0) + c_{n-2}x^{n-1} + \cdots + c_0 x \\ &= (c_{n-1}a_{n-1} + c_{n-2})x^{n-1} + (c_{n-1}a_{n-2} + c_{n-3})x^{n-2} \\ &\quad + \cdots + (c_{n-1}a_1 + c_0)x + c_{n-1}a_0. \end{aligned}$$

Using the natural representation of polynomials as vectors, we can say that if $\underset{\sim}{x} = [c_{n-1}, \ldots, c_0]$ be the vector representing $x^{n+\alpha} \pmod{p(x)}$, then the vector $\underset{\sim}{y} = [c'_{n-1}, \ldots, c'_0]$ representing $x^{n+\alpha+1} \pmod{p(x)}$ is obtained as follows:

$$\begin{bmatrix} a_{n-1} & 1 & 0 & \ldots & 0 \\ a_{n-2} & 0 & 1 & \ldots & 0 \\ . & . & . & \ldots & . \\ a_1 & 0 & 0 & \ldots & 1 \\ a_0 & 0 & 0 & \ldots & 0 \end{bmatrix} \begin{bmatrix} c_{n-1} \\ . \\ . \\ . \\ c_0 \end{bmatrix} = \begin{bmatrix} c'_{n-1} \\ . \\ . \\ . \\ c'_0 \end{bmatrix}$$

which we can write as $A\underset{\sim}{x}^{\top} = \underset{\sim}{y}^{\top}$.

We first describe an algorithm for obtaining $f(x) \pmod{p(x)}$ for any polynomial $f(x) = f_m x^m + f_{m-1}x^{m-1} + \cdots + f_0$ with $f_m = 1$.

ALGORITHM $\mathcal{A}$.

input $\underset{\sim}{f} = (f_m, \ldots, f_0)$ a vector representing the polynomial $f(x)$.
output $\underset{\sim}{r} = (r_{n-1}, \ldots, r_0)$ a vector representing the polynomial $f(x) \pmod{p(x)}$
method
begin
    $\underset{\sim}{r} = (f_{n-1}, \ldots, f_0)$
    $\underset{\sim}{t} = (a_{n-1}, \ldots, a_0)$
    for $i = 0$ to $(m - n)$ do
        if $(f_{n+i} = 1)$ $\underset{\sim}{r} = \underset{\sim}{r} + \underset{\sim}{t}$
        $\underset{\sim}{t} = A\underset{\sim}{t}$
    od
end

REMARK 2.1.

1. Correctness of the algorithm follow from the previous discussion.
2. Assuming that the loop can be executed in one clock cycle, the algorithm takes $m - n + 1$ clock cycles. Next, we show how to execute the loop in one clock cycle.

Let $\underset{\sim}{x} = (c_{n-1}, \ldots, c_0)$ and $\mathcal{C}$ be an $(n + 1)$ cell left shift register. Let the contents of $\mathcal{C}$ be $(d, c_{n-1}, \ldots, c_0)$, where d means don't care. Now $\mathcal{C}$ is left shifted once. If the content of the first cell is 1, then $(a_{n-1}, \ldots, a_0)$ is added bitwise to the contents of $\mathcal{C}$ leaving out the first cell. Then the contents of $\mathcal{C}$ (leaving out the first cell) gives the vector $\underset{\sim}{y} = A\underset{\sim}{x}$.

Using this implementation of the operation $\underset{\sim}{t} = A\underset{\sim}{t}$, it is easy to implement Algorithm $\mathcal{A}$ in VLSI. We essentially compute the powers $x^i \pmod{p(x)}$ and add up the partial results in another register. The entire MOD circuit is shown in Figure 1 which operates as follows.

Initially, $\mathcal{C}$ is loaded with $(0, a_{n-1}, \ldots, a_0)$ and $\mathcal{R}$ is loaded with $(f_{n-1}, \ldots, f_0)$. The register $\mathcal{P}$ contains the fixed vector $(a_{n-1}, \ldots, a_0)$. The coefficient f_{n+i} $0 \leq i \leq m - n$ is available in the i^{th} clock cycle. In the positive half of the i^{th} clock cycle, $\mathcal{C}$ is evolved (left shifted) once and if $f_{n+i} = 1$, then simultaneously, the contents of $\mathcal{C}$ are added bitwise to the contents of $\mathcal{R}$. In the negative half of the i^{th} cycle, if the leftmost bit of $\mathcal{C}$ is 1, then the contents of $\mathcal{P}$ are added bitwise to the contents of $\mathcal{C}$.

REMARK 2.2.

1. The flexibility mentioned in the introduction arises due to the fact that to divide by another polynomial only requires a change of values in register $\mathcal{P}$ (which stores the coefficients of polynomial $p(x)$).
2. Using the above implementation Algorithm $\mathcal{A}$ can be completed in $m-n+1$ clock cycles. This time is an improvement over LFSR circuits which require m clock cycles.
3. To implement Algorithm $\mathcal{A}$, $2n$ 2-input EXOR gates and 3 n-bit registers are required. For LFSR circuits, one n bit register is required and the number of gates required is $\leq n$.

3. POLYNOMIAL MULTIPLICATION MODULO A POLYNOMIAL

In this section, we describe how the MOD circuit (Figure 1) can be used to perform $f(x)g(x) \pmod{p(x)}$, where $f(x)$ and $g(x)$ are arbitrary polynomials and $p(x)$ is a fixed polynomial of degree n.

Let,

$$f(x) = f_m x^m + \cdots + f_0,$$
$$g(x) = g_m x^m + \cdots + g_0.$$

The algorithm for modulo multiplication is presented as Algorithm B.

ALGORITHM $\mathcal{B}$.

We assume a function modulo$(f(x))$ exists which returns $f(x) \pmod{p(x)}$.

begin

$q(x) =$ modulo$(f(x))$
if $(g_0 = 0)$ then $r(x) = 0$ else $r(x) = q(x)$
for $i = 1$ to m do
$q(x) =$ modulo$(x * q(x))$
if $(g_i = 1)$ then $r(x) = q(x) + r(x)$
od

end

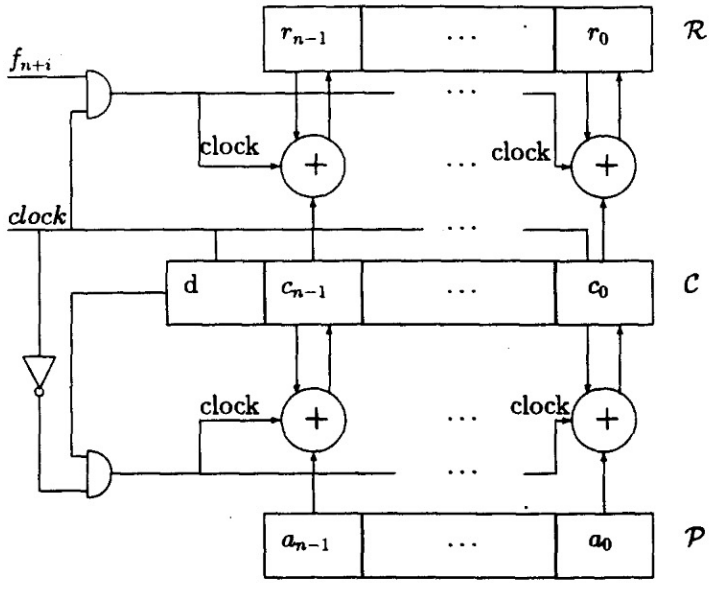

Figure 1. The MOD circuit.

To see that the algorithm works, note that the following holds.

1. $x^{i+1} * h(x) \bmod p(x) = x * (x^i * h(x) \bmod p(x)) \pmod{p(x)}$,
2. $f(x)g(x) \bmod p(x) = (g_m x^m * h(x) \pmod{p(x)} + \cdots + g_i x^i * h(x) \pmod{p(x)} + \cdots + g_0 * h(x) \pmod{p(x)}) \pmod{p(x)}$,

where $h(x) = f(x) \bmod p(x)$.

The MOD circuit can be used to implement Algorithm $\mathcal{B}$ in the following way. In the first phase of operation, we compute $f(x) \bmod p(x)$ as usual. This takes $m - n + 1$ clock cycles. In the next clock cycle, $\mathcal{C}$ (leaving out the leftmost cell) is loaded with $\mathcal{R}$ and if $g_0 = 0$, then simultaneously register $\mathcal{R}$ is reset to 0. Starting from the next clock cycle and continuing for the next m clock cycles, the coefficients $g_1 \ldots g_m$ are input to the circuit (in place of f_{n+i}s). So after $2m - n + 2$ clock cycles, the register $\mathcal{R}$ contains the result.

As in the case of Algorithm $\mathcal{A}$, dividing by another $p(x)$ (of same degree), will only require a change in stored values of $\mathcal{P}$. Also note that for a fixed $p(x)$, both $f(x)$ and $g(x)$ are arbitrary. For LFSR circuits, $p(x)$ along with one of $f(x)$ or $g(x)$ must be fixed. Thus, our implementation provides more flexibility in design.

4. APPLICATIONS

4.1. Standard Basis Multiplication

In this section, we describe the use of the MOD circuit for finite field multiplication. Arithmetic over finite fields has important applications. See [3–8] for related work in this area. We first describe some finite field terminology all of which can be found in [1,2].

Let $GF(2^n)$ be the field extension of degree n over $GF(2)$. Then it is also a vector space of dimension n over $GF(2)$. Let $p(x)$ be an irreducible polynomial of degree n over $GF(2)$ and let α be one of its roots. Then $1, \alpha, \ldots, \alpha^{n-1}$ forms a basis (called a *standard basis*) for $GF(2^n)$ over $GF(2)$. Any element β of $GF(2^n)$ can be written as $\beta = f(\alpha)$, where $f(x)$ is a polynomial of degree at most n over $GF(2)$. If $\beta, \gamma \in GF(2^n)$, where $\beta = f(\alpha)$ and $\gamma = g(\alpha)$, then $\beta + \gamma = f(\alpha) + g(\alpha)$. So addition over $GF(2^n)$ is simply the polynomial addition of $f(x)$ and $g(x)$, and hence is simple to perform. The multiplication is more complicated and it is the multiplication which can be easily performed by the MOD circuit. Let,

$$\beta = f_0 + f_1\alpha + \cdots + f_{n-1}\alpha^{n-1},$$
$$\gamma = g_0 + g_1\alpha + \cdots + g_{n-1}\alpha^{n-1}.$$

Then β and γ are, respectively, represented in a unique way by the tuples $(f_0, \ldots, f_{n-1})$ and $(g_0, \ldots, g_{n-1})$, with respect to the standard basis $1, \alpha, \ldots, \alpha^{n-1}$. Then $\beta\gamma = h(\alpha)$, where, $h(x) = f(x)g(x) \bmod p(x) = h_0 + h_1 x + \cdots + h_{n-1}x^{n-1}$

Again $\beta\gamma$ is also uniquely represented by the tuple $(h_0, \ldots, h_{n-1})$. Note that in the whole operation the role played by α is only that of a placeholder. Hence to obtain the element $\beta\gamma$ all we have to do is to perform the modulo multiplication $f(x)g(x) \bmod p(x)$. This is done by the MOD circuit in the way described in the previous section and takes a total of $n + 2$ clock cycles.

REMARK 4.1.

1. For a modulo multiplication circuit to be used as a standard basis multiplier, it is essential for the circuit design to be independent of $f(x)$ and $g(x)$. Hence, LFSR based modulo multiplication circuits cannot be used for standard basis multiplication.
2. In the use of the MOD circuit for standard basis multiplication, the register $\mathcal{P}$ stores the the coefficients of $p(x)$. If one were to choose some other irreducible polynomial, then all that is required is a change in the stored values of register $\mathcal{P}$.

4.2. Systematic Cyclic Codes

One of the major application areas for polynomial division circuit is encoding and decoding of error correcting codes [1,2,9]. Here, we briefly describe how the above circuits can be used for encoding and decoding of systematic cyclic codes [1]. See [10] for a discussion on Reed-Solomon code and its VLSI implementation.

Let $d(x)$ denote the data polynomial (of degree n) and $g(x)$ the generator polynomial (of degree r) for the code space. Then the codeword $u(x)$ is formed as follows:

$$u(x) = d(x)x^r - r(x), \tag{1}$$

$$\text{where } r(x) = d(x)x^r, \qquad (\text{mod } g(x)). \tag{2}$$

The operation in (2) can be done using the MOD circuit. The actual method of obtaining $u(x)$ on line is discussed below.

Decoding is done by generating the syndrome for the received codeword and then using it for possible error correction. Let $v(x)$ be the received codeword. Then the syndrome of $v(x)$ is formed as follows.

$$s(v(x)) = v(x), \qquad (\text{mod } g(x)).$$

Again the above operation can be done using the MOD circuit.

For LFSR based designs, the data polynomial is fed to the encoding circuit, high order bit first. Since LFSR based division circuit require input with high order bit first, the data polynomial can be input to the division circuit and transmitted simultaneously. After the remainder has been formed, it is also transmitted with high order bit first. Thus, the polynomial $d(x)x^r + r(x)$ can be transmitted high order bit first with no delay.

For the MOD circuit that we have described, input is required low order bit first. Therefore, $d(x)$ is also transmitted low order bit first. After $r(x)$ have been computed it is transmitted low order bit first. At the receiving end $d(x)$ is obtained before $r(x)$. This however does not cause any delay in syndrome generation, since $d(x)x^r$ mod $g(x)$ is first computed and then $r(x)$ added to the result.

REFERENCES

1. T.R.N. Rao and E. Fujiwara, *Error Control Coding for Computer Systems*, Prentice Hall, (1989).
2. W.W. Peterson and E.J. Weldon, *Error Correcting Codes*, MIT Press, Cambridge, MA, (1972).
3. M.A. Hasan, Division-and-accumulation over $GF(2^m)$, *IEEE Transactions on Computers* **46** (6), (1997).
4. R. Barua and S. Sengupta, *Proceedings of 10^{th} Int. Conf. on VLSI Design*, pp. 465–468, IEEE Press, (1994).
5. E.D. Mastrovito, VLSI Architectures for computations in galois fields, Ph.D. Thesis, Dept. of Elec. Eng., Linkoping Univ., Linkoping, Sweden, (1991).
6. C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura and I.S. Reed, VLSI architectures for computing multiplications and inverses in $GF(2^m)$, *IEEE Transactions on Computers* **C-34**, 709–717, (1985).
7. B.B. Zhou, A new bit-serial systolic multiplier over $GF(2^m)$, *IEEE Transactions on Computers* **C-37**, 749–751, (1988).
8. P. Pal Choudhury and R. Barua, Cellular automata based VLSI architectures for computing multiplication and inverses in $GF(2^m)$, In *Proceedings of 7^{th} Int. Conf. on VLSI Design*, pp. 279–282, IEEE Press, (1994).
9. E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, (1968).
10. *Algorithms and Architectures for the Design of a VLSI Reed-Solomon Code*, Edited by S.B. Wicker and V.K. Bhargava, IEEE Press, (1994).