

Shift Registers

♣ **Introduction.** To make error correcting codes easier to use and analyse, it is necessary to impose some algebraic structure on them. It is especially useful to have an alphabet in which it is possible to add, subtract, multiply and divide without restriction. In other words we wish to construct a *finite field*. Evarist Galois (1811-32), a French mathematician who died in a duel at the age of 20 introduced finite fields and proved that there exists a field of order q if and only if q is a prime power (i.e. $q = p^r$, where p is prime and r is a positive integer). Finite fields of order q are also known as *Galois fields* of order q and are denoted by $GF(q)$. In general elements of the finite field $GF(2^r)$ are represented by r -tuples of 0's and 1's, they are easy to manipulate using digital circuits or in a binary computer.

One reason that *cyclic codes* are so useful is that they can be efficiently encoded and decoded by means of *Linear shift registers*. The encoding process is efficient because no storage is required as the codewords are generated by shifting and adding. In linear finite-state switching circuits, information is assumed to be some representation of elements of $GF(2^r)$. In a linear shift register, the following devices are used:

1. *Registers (or delay elements)*, each capable of holding one bit (0 or 1) and each having one input and one output (Figure 1). The arrows indicate the input and output. The output of each register is always the same as the input was one unit of time earlier.
2. A *clock* or *shift signal* which controls the movement of shifting of the data contained in the registers. After each clock "tick", the output of each stage takes the value that the input took immediately before the shift signal appeared.
3. *Binary adders* which have two inputs and one output, (output is 1 if and only if odd number of inputs are 1). Also called *Exclusive-OR (XOR) gates* (Figure 2).
4. *Constant multipliers*, each has one input and one output, the output being simply the input multiplied by a constant. The constant multiplier for the constant 1 is simply a connection, and for the constant 0, simply no connection. The rule for interconnection of these devices is that any number of inputs may be connected to any output, but that two outputs are never connected together.



Figure 1. Register

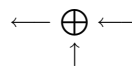


Figure 2. Binary adder

The connection of inputs and outputs must be serial; that is, it consists of elements entering an input line one at a time, one for each unit time. A shift register circuit may have several

inputs and outputs. The contents of registers at each time interval is called the *t-state*; the initial state is also called the *seed*.

When an input or output is a polynomial, as is often the case, only the coefficients appear on the input or output line. In the case of division of polynomials, high-order coefficients enter first. The reason is that in division the high-order coefficients of the dividend must be processed first. Thus the polynomial

$$a(x) = a_0 + a_1x + \cdots + a_nx^n$$

would be entered on an input line as a succession of n elements, with a_n coming first, then a_{n-1} one unit of time later, a_{n-2} after another unit of time, and so forth. In the case of multiplication of polynomials, coefficients may be entered either in high-order or low-order.

Consider an arbitrary linear shift register circuit with k registers, r input lines, and s output lines (r or s or both may be zero). Then the t -state of the circuit is the $1 \times k$ vector $u_t = [u_{t,0}, u_{t,1}, \dots, u_{t,k-1}]$. The circuit input is a $1 \times r$ vector v_t and the output is a $1 \times s$ vector denoted by w_t . The input u_t to the registers is a function of the output u_{t-1} of the registers and the circuit input v_t . Since the circuit is made up only of binary adders and constant multipliers, the process must be linear, and can be expressed in matrix form:

$$u_t = [u_{t,0}, u_{t,1}, \dots, u_{t,k-1}] = u_{t-1}S + v_tR,$$

where S is a $k \times k$ *shift matrix* and R is a $r \times k$ matrix. Similarly the output w_t is a linear function of the state vectors u_{t-1} and u_t :

$$w_t = u_{t-1}Q + u_tL,$$

where Q and L are $k \times s$ matrices.

The companion matrix of $g(x) = g_0 + g_1x + \cdots + g_{n-1}x^{n-1} + x^n$, denoted by C_g is an $n \times n$ matrix defined as follows:

$$C_g = \begin{bmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \vdots & \vdots & 0 & 1 \\ g_0 & g_1 & \cdots & \cdots & g_{n-2} & g_{n-1} \end{bmatrix} \quad \text{with} \quad C_g^t = \begin{bmatrix} 0 & 0 & \cdots & \cdots & \cdots & g_0 \\ 1 & 0 & 0 & \cdots & \cdots & g_1 \\ \vdots & \ddots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots \\ 0 & 0 & \vdots & \ddots & 0 & g_{n-2} \\ 0 & 0 & \cdots & \cdots & 1 & g_{n-1} \end{bmatrix}$$

It can be shown that $g(x)$ is the characteristic polynomial of both C_g and C_g^t ; that is

$$\det(xI_n + C_g) = \det(xI_n + C_g^t) = g(x).$$

We shall see that the companion matrix of a polynomial will be the shift matrix of the circuit for dividing polynomials while the transpose of a companion matrix will be the shift matrix of a circuit that encodes a cyclic code.

An irreducible polynomial over \mathbb{K} of degree n , $n > 1$ is said to be *primitive*, if it is not a divisor of $1 + x^m$ for $m < 2^n - 1$. For example, $g(x) = 1 + x + x^3$ is primitive, since it is not a factor of $1 + x^m$ for $m < 2^3 - 1 = 7$. However $h(x) = 1 + x + x^2 + x^3 + x^4$ is irreducible but not primitive since it divides $1 + x^5$ and $5 < 2^4 - 1 = 15$. It is known that any primitive polynomial has an odd number of nonzero coefficients; the converse is not true (see $h(x)$).

The reciprocal of the polynomial $g(x) = g_0 + g_1x + \cdots + g_{m-1}x^{m-1} + x^m$, denoted by $\hat{g}(x)$ is defined as

$$\hat{g}(x) = x^m g\left(\frac{1}{x}\right) = g_0x^m + g_1x^{m-1} + \cdots + g_{m-1}x + 1.$$

It is not difficult to show that $\hat{g}(x)$ is irreducible (resp. primitive) if and only if $g(x)$ is irreducible (resp. primitive).

Here is a list of some “well-known” primitive polynomials used in common systems.

Used in Cyclic Redundancy Check codes:

$$\text{CRC-12: } x^{12} + x^{11} + x^3 + x + 1$$

$$\text{CRC-16: } x^{16} + x^{15} + x^2 + 1$$

$$\text{CCIT: } x^{16} + x^{12} + x^5 + 1$$

$$\text{AUTODIN II: } x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Permitted (under regulations which are now out of date) for use in generating spread-spectrum sequences by radio amateurs:

$$\text{7-bit: } x^7 + x + 1$$

$$\text{13-bit: } x^{13} + x^4 + x^3 + x + 1$$

$$\text{19-bit: } x^{19} + x^5 + x^2 + x + 1$$

Originally alleged as used in the A5 European cellular telephone algorithm:

$$x^{19} + x^5 + x^2 + x + 1$$

$$x^{22} + x^9 + x^5 + x + 1$$

$$x^{23} + x^4 + x^3 + x + 1$$

Actually used in the A5 cellular telephone algorithm, according to more recent information:

$$x^{19} + x^5 + x^2 + x + 1$$

$$x^{22} + x + 1$$

$$x^{23} + x^{15} + x^2 + x + 1$$

$$x^{17} + x^5 + 1$$

Used by GPS satellites:

$$x^{10} + x^3 + 1$$

$$x^{10}x^9 + x^8 + x^6 + x^3 + x^2 + 1$$

♣ **Linear Feedback Shift Register.** A linear shift register with the output fed back into the device is called a *linear feedback shift register* \mathcal{LFSR} . Here is an example of \mathcal{LFSR} :

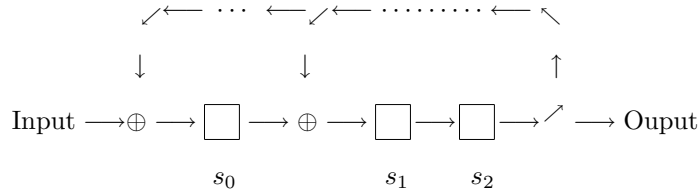


Figure 3. \mathcal{LFSR}

At each clock tick the input and contents of the registers are shifted and then the output digit c_t , is added into selected registers. We shall see that Figure 3 represents division by $g(x) = 1 + x + x^3$.

Example 1. An element of $GF(2^4)$ generated by the primitive polynomial $g(x) = 1 + x + x^4$ is represented by 4 0's and 1's, which can be stored in a row of 4 storage elements (*registers*). For example,

$$\boxed{0} \quad \boxed{0} \quad \boxed{1} \quad \boxed{1}$$

contains the element $0011 \leftrightarrow \alpha^2 + \alpha^3$. The circuit in Figure 4 multiplies the contents of registers by α in $GF(2^4)$.

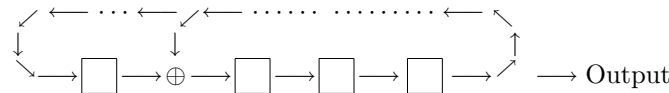


Figure 4. Generating elements of $GF(2^4)$

The initial state is

$$\boxed{a_0} \longrightarrow \boxed{a_1} \longrightarrow \boxed{a_2} \longrightarrow \boxed{a_3}$$

then one unit time later the state is

$$\begin{matrix} \square & \longrightarrow & \square & \longrightarrow & \square & \longrightarrow & \square & \leftrightarrow & \alpha(a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3) \\ a_3 & & a_0 + a_3 & & a_1 & & a_2 & = & a_3 + (a_0 + a_3)\alpha + a_1\alpha^2 + a_2\alpha^3. \end{matrix}$$

The initial state u_0 , the shift matrix S_g , and L_g corresponding to circuit generated by the polynomial $g(x) = 1 + x + x^4$ are as follows:

$$u_0 = [a_0, a_1, a_2, a_3], \quad S_g = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \quad Q_g = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad L_g = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Since there is no circuit input, $R_g = [0 \ 0 \ 0 \ 0]$. Thus we have

$$u_t = u_{t-1} S_g = [u_{t-1,0}, u_{t-1,1}, u_{t-1,2}, u_{t-1,3}] \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = [u_{t-1,3}, u_{t-1,0} + u_{t-1,3}, u_{t-1,1}, u_{t-1,2}]$$

and $w_t = u_t L_g = [u_{t,0}, u_{t,1}, u_{t,2}, u_{t,3}] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = u_{t,3}$.

If initially the registers contain $1000 \leftrightarrow 1$, then at successive time instant they contain $1, \alpha, \alpha^2, \dots, \alpha^{14}, \alpha^{15}, 1, \alpha, \dots$. So the output of the circuit is periodic with period 15. This is the maximum possible period with 4 storage elements (since there are just $2^4 - 1 = 15$ nonzero states).

Remark 1. Any \mathcal{LFSR} with no input containing all zeros will never move to any other state, and so the initial state consisting of all zeros must be excluded. Also note that only primitive polynomial of degree n can produce maximum period of $2^n - 1$.

♣ Encoding Linear Cyclic Codes with Shift Register Generators. Consider an (n, k) linear cyclic code generated by the polynomial $g(x) = g_0 + g_1x + \dots + g_{k-n}x^{n-k}$ and let

$$h(x) = \frac{x^n + 1}{g(x)} = h_0 + h_1x + \dots + h_kx^k.$$

Encoding this code can be accomplished by using the following linear shift register circuit (\mathcal{LFSR}) known as a k -stage shift register generator.

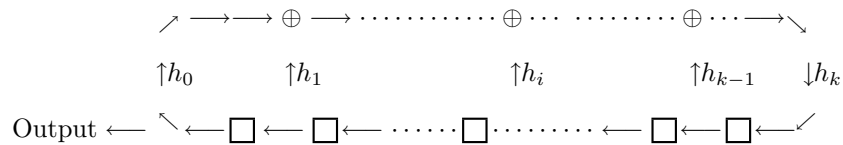


Figure 5. A shift register generator

The information digits (message) a_0, a_1, \dots, a_k are initially stored in k registers s_0, s_1, \dots, s_k , and then the circuit is made to shift n times. The first k digits that come out will be the message, and the $n - k$ digits that follow will be check digits that make the whole n -digit word a codeword. The shift matrix $S_h = C_h^t$, where C_h is the companion matrix of $h(x)$. The initial state $u_0 = (a_0, a_1, \dots, a_k)$ with $u_{t+1} = S_h u_t$ and the output $q_t = u_{t-1,0}$.

Example 2. Consider the (7, 4) cyclic Hamming code generated by the polynomial

$$g(x) = 1 + x + x^3 \quad \text{with} \quad h(x) = h_0 + h_1x + h_2x^2 + h_3x^3x^4 \frac{x^7 + 1}{1 + x + x^3} = 1 + x + x^2 + x^4$$

The shift register shown in Figure 6 can be used for the encoding.

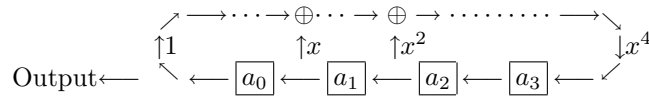


Figure 6. Shift register generator for a (7,4) Hamming code

At time zero, four binary elements a_0, a_1, a_2, a_3 are placed in s_0, s_1, s_2 , and s_3 . After one time interval a_0 is output, a_1 is shifted into s_0 , a_2 into s_1 , and a_3 into s_2 ; and the new element is entered into s_3 . In our example this element is the sum $a_0 + a_1 + a_2$ corresponding to the polynomial $h_0 + h_1 + h_2$. So we have:

$$u_{t+1} = u_t S_g = [u_{t,0}, u_{t,1}, u_{t,2}, u_{t,3}] \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad q_t = u_{t-1,0}.$$

Suppose that the digits 1101 are placed in s_0, s_1, s_2 , and s_3 and follow the outputs and inputs for seven time intervals.

$h(\mathbf{x}) = 1 + \mathbf{x} + \mathbf{x}^2 + \mathbf{x}^4$					
output	s_0	s_1	s_2	s_3	time
—	1	1	0	1	t_0
1	1	0	1	0	t_1
1	0	1	0	0	t_2
0	1	0	0	1	t_3
1	0	0	1	1	t_4
0	0	1	1	1	t_5
0	1	1	1	0	t_6
1	1	1	0	1	t_7

Table 1. Generating a codeword for the message 1 1 0 1

If this process is continued, the vector (1, 1, 0, 1, 0, 0, 1) will be repeated.

♣ Pseudo-Random Numbers. Normally random numbers are made by using linear feedback shift-register random number generators. These kind of generators are easy to implement

and produce fairly good *pseudo-randomness*. To produce these numbers, we proceed as follows:

1. Use a primitive polynomial $g(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} + x^n$.
2. Use $g(x)$ to design a \mathcal{LFSR} described in Figure 5.
3. Initialize the registers with a nonzero seed vector.

The output sequence will be a pseudo-random sequence with a period of $2^n - 1$. Here is a table for $g(x) = 1 + x + x^3$ with the seed 110 and period $2^3 - 1 = 7$:

$g(x) = 1 + x + x^3$				
output	s_0	s_1	s_2	time
—	1	1	0	t_0
1	1	1	1	t_1
1	1	0	1	t_2
1	0	0	1	t_3
0	0	1	0	t_4
0	1	0	0	t_5
1	0	1	1	t_6
0	1	1	0	t_7
1	1	1	0	t_8
1	1	1	1	t_9

Table 2. Generating a pseudo-random sequence

The primitive polynomial $g(x) = 1 + x + x^{22}$ generates a pseudo-random sequence of maximum period $2^{22} - 1$. The polynomial $h(x) = 1 + x + x^2 + x^4$ used in Table 1 is not primitive since it has only a period of 7 instead of $2^4 - 1 = 15$. Notice that $h(x)$ has an even number of nonzero coefficients h_0, h_1, h_2 , and h_4 , so it can not be primitive.

♣ Polynomial Multiplication. Linear shift registers can be designed to multiply any polynomial $a(x) = a_0 + a_1x + \dots + a_mx^m$ by a fixed polynomial $g(x) = g_0 + g_1x + \dots + g_kx^k$. The following diagram represents a shift register circuit for multiplication.

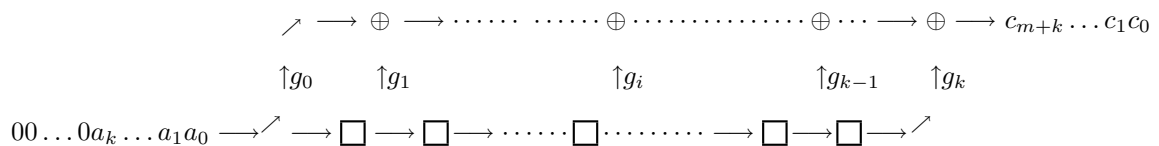


Figure 7. A circuit for multiplying $a(x)$ by $g(x)$

The registers are assumed to contain 0's initially, and the coefficients of $a(x)$ are assumed to enter low order first and to follow by k 0's; that is:

$$\text{Input } 0, 0, \dots, 0, a_m, a_{m-1}, \dots, a_1, a_0 \rightarrow \dots \rightarrow c_{m+k}, c_{m+k-1}, \dots, c_1, c_0 \text{ Output}$$

The shift matrix and the vectors for the circuit in Figure 7 are as follows:

$$S_g = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & 1 & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}, R_g = [1 \ 0 \ 0 \ \dots \ 0], Q_g = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \text{ and } L_g = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix}.$$

Example 3. Consider the 3-stage shift register in Figure 8 generated by $g(x) = 1 + x + x^3$.

We have:
$$\begin{aligned} c(x) &= a(x)g(x) = (a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4)(1 + x + x^3) \\ &= a_0 + (a_1 + a_0)x + (a_2 + a_1)x^2 + (a_3 + a_2 + a_0)x^3 \\ &\quad + (a_4 + a_3 + a_1)x^4 + (a_4 + a_2)x^5 + a_3x^6 + a_4x^7 \end{aligned}$$

The circuit shown in Figure 8 outputs $c(x) = a(x)(1 + x + x^3)$.

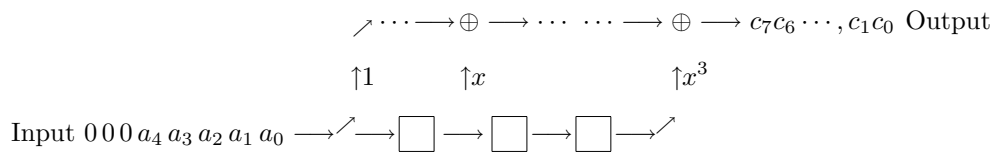


Figure 8. Multiplying $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$ by $g(x) = 1 + x + x^3$.

The following table shows how $a(x)$ and $g(x)$ are multiplied.

		$g(x) = 1 + x + x^3$			
time	input	s_0	s_1	s_2	output
0	—	0	0	0	— — —
1	a_0	a_0	0	0	a_0
2	a_1	a_1	a_0	0	$a_1 + a_0$
3	a_2	a_2	a_1	a_0	$a_2 + a_1$
4	a_3	a_3	a_2	a_1	$a_3 + a_2 + a_0$
5	a_4	a_4	a_3	a_2	$a_4 + a_3 + a_1$
6	0	0	a_4	a_3	$a_4 + a_2$
7	0	0	0	a_4	a_3
8	0	0	0	0	a_4

Table 3. Multiplying $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$ by $g(x) = 1 + x + x^3$

The next table shows how $a(x) = 1 + x + x^3 + x^4$ is multiplied by $g(x) = 1 + x + x^3$.

		$g(x) = 1 + x + x^3$			
time	input	s_0	s_1	s_2	output
0	—	0	0	0	— — —
1	1	1	0	0	1
2	1	1	1	0	$1 + 1 = 0$
3	0	0	1	1	$0 + 1 = 1$
4	1	1	0	1	$1 + 0 + 1 = 0$
5	1	1	1	0	$1 + 1 + 1 = 1$
6	0	0	1	1	$1 + 0 = 1$
7	0	0	0	1	1
8	0	0	0	0	1
					$1 + x^2 + x^4 + x^5 + x^6 + x^7$

Table 4. Multiplying $a(x) = 1 + x + x^3 + x^4$ by $g(x) = 1 + x + x^3$.

The following diagram describes another shift register circuit for multiplying any polynomial $a(x)$ by a fixed polynomial $g(x) = g_kx^k + g_{k-1}x^{k-1} + \dots + g_1x + g_0$.

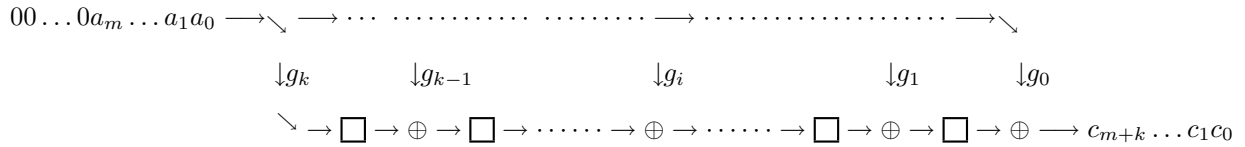


Figure 9. Another circuit for multiplying $a(x)$ by $g(x)$

The shift matrix S_g and the vectors $R_g, L_g,$ and Q_g are as follows:

$$S_g = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}, R_g = [g_k, g_{k-1}, g_{k-2}, \dots, g_1], Q_g = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad \text{and} \quad L_g = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Note that the shift matrix S_g and the vector Q_g for both circuits shown in Figure 7 and Figure 9 are identical.

Remark 2. In the multiplication circuits of Figure 7 and Figure 9, if the coefficients of $a(x)$ are entered high order first followed by k 0's, then the same circuits multiply $a(x)$ by the reciprocal polynomial $\hat{g}(x)$ of $g(x)$. So if $d(x) = d_0 + d_1x + \dots + d_{m+k}x^{m+k} = a(x)\hat{g}(x)$, then

$$\text{Input } 0, 0, \dots, 0, a_0, a_1, \dots, a_{m-1}, a_m \rightarrow \dots \rightarrow d_0, d_1, \dots, d_{m+k-1}, d_{m+k} \text{ Output}$$

Hence the connections $\uparrow g_0, \uparrow g_1, \dots, \uparrow g_k$ in Figure 7 and $\downarrow g_k, \downarrow g_{k-1}, \dots, \downarrow g_0$ in Figure 9 must be replaced by $\uparrow g_k, \uparrow g_{k-1}, \dots, \uparrow g_0$ and $\downarrow g_0, \downarrow g_1, \dots, \downarrow g_k$ respectively.

Example 4. Consider again the polynomial $g(x) = x^3 + x + 1$. The shift register circuit for the multiplication is shown in the following diagram.

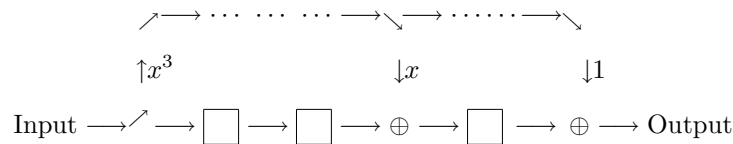


Figure 10. Circuit for multiplying $a(x)$ by $x^3 + x + 1$

We have $u_t = u_{t-1} S_g = u_{t-1} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} + v_t [1 \ 0 \ 1]$ and $w_t = u_{t-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + u_t \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$

$$\text{Thus } U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{bmatrix} = \begin{bmatrix} a_0 & 0 & a_0 \\ a_1 & a_0 & a_1 \\ a_2 & a_1 & a_0 + a_2 \\ a_3 & a_2 & a_1 + a_3 \\ a_4 & a_3 & a_2 + a_4 \\ 0 & a_4 & a_3 \\ 0 & 0 & a_4 \\ 0 & 0 & 0 \end{bmatrix}, \quad \text{and} \quad W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_0 + a_1 \\ a_1 + a_2 \\ a_0 + a_2 + a_3 \\ a_1 + a_3 + a_4 \\ a_2 + a_4 \\ a_3 \\ a_4 \end{bmatrix}.$$

♣ **Encoding Linear Cyclic Codes with (n-k)-Stage Shift Register Circuits.** A codeword in a linear cyclic code of length n and dimension k can be generated by a generator polynomial $g(x)$ of degree $n - k$. This can be mechanized with a shift register circuit shown in Figure 7 or in Figure 9. With this method, contrary to the method used with the shift register generator, the message (information digits) will be altered in the codeword, but can be recovered by division by $g(x)$.

Example 5. Consider the (7,4) Hamming cyclic code generated by $g(x) = 1 + x + x^3$. We can use the shift register in Figure 8 to encode the message $a = 1101\ 000$ into the codeword $c(a) = 1010001$ as follows:

$$g(x) = 1 + x + x^3$$

time	input	s ₀	s ₁	s ₂	output
0	—	0	0	0	— — —
1	1	1	0	0	1
2	1	1	1	0	0
3	0	0	1	1	1
4	1	1	0	1	0
5	0	0	1	0	0
6	0	0	0	1	0
7	0	0	0	0	1
					1010001

Table 5. Encoding a = 1 1 0 1 with $g(x) = 1 + x + x^3$.

Notice that the first 4 digits of $c(a) = 1010\ 001$ are different from $a = 1101$.

Remark 3. It should be noted that for a linear cyclic code of length n and dimension k , if the number of parity-check digits ($n - k$) is larger than the number of information digits (k), then encoding with a shift register generator circuit would be preferable, while for codes with fewer parity-check digits than information digits this method would generally be more economical. Either will produce the same cyclic code.

♣ **Multiple-Input Shift Registers.** Circuits of type shown in Figure 9 can have more than one input. Suppose for some fixed polynomials

$$g(x) = g_k x^k + g_{k-1} x^{k-1} + \dots + g_1 x + g_0 \quad \text{and} \quad h(x) = h_k x^k + h_{k-1} x^{k-1} + \dots + h_1 x + h_0,$$

we need to find $c(x) = a(x)g(x) + b(x)h(x)$, for any given polynomials

$$a(x) = a_0 + a_1 x + \dots + a_{m-1} x^{m-1} \quad \text{and} \quad b(x) = b_0 + b_1 x + \dots + b_{n-1} x^{n-1} + b_n x^n.$$

The next diagram describes the shift register circuit.

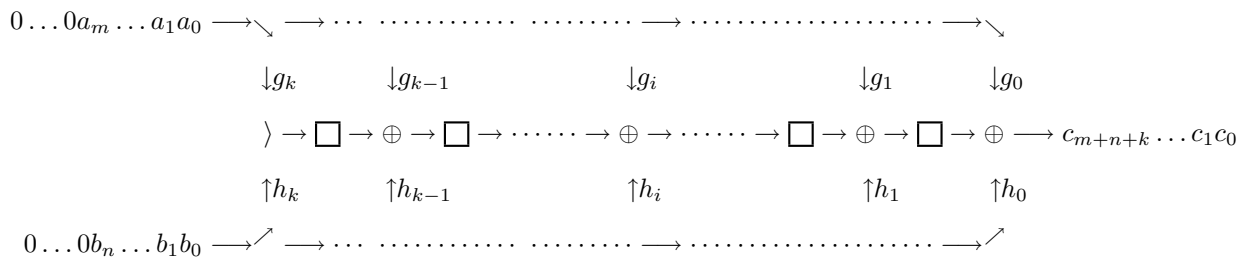


Figure 11. A two-input multiplier

Note that in this circuit we assume that $g(x)$ and $h(x)$ have both the same degree k , but in case the degrees are not equal, k can be taken as the larger degree, and the high-order coefficients of one polynomial can be 0. If $m < n$ (resp. $n < m$), then we set

$$a_{m+1} = a_{m+2} = \dots = a_n = 0 \text{ (resp. } b_{n+1} = b_{n+2} = \dots = b_m = 0).$$

The registers are assumed to contain 0's initially, and the coefficients of $a(x)$ and $b(x)$ are entered low order first followed by k 0's.

The shift matrix $S_{g,h}$ and the vectors $R_{g,h}$, $Q_{g,h}$, and $L_{g,h}$ are as follows:

$$S_{g,h} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}, R_{g,h} = \begin{bmatrix} g_k & g_{k-1} & g_{k-2} & \dots & g_1 \\ h_k & h_{k-1} & h_{k-2} & \dots & h_1 \end{bmatrix}, Q_{g,h} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \text{ and } L_{g,h} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

So for $v_t = [a_t \ b_t]$ ($v_t = [0 \ 0]$ whenever $t > \max\{m, n\}$) we have

$$\begin{aligned} u_t &= u_{t-1} S_{g,h} + v_t R_{g,h} \\ &= [u_{t-1,0}, u_{t-1,1}, \dots, u_{t-1,k-1}] \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} + [a_t \ b_t] \begin{bmatrix} g_k & g_{k-1} & \dots & g_1 \\ h_k & h_{k-1} & \dots & h_1 \end{bmatrix} \\ &= [0 \ u_{t-1,0} \ u_{t-1,1} \ \dots \ u_{t-1,k-2}] + a_t [g_k \ g_{k-1} \ \dots \ g_1] + b_t [h_k \ h_{k-1} \ \dots \ h_1] \\ &= [a_t g_k + b_t h_k \ u_{t-1,0} + a_t g_{k-1} + b_t h_{k-1} \ \dots \ u_{t-1,k-2} + a_t g_1 + b_t h_1] \end{aligned}$$

and $w_t = u_{t-1} Q_{g,h} + u_t L_{g,h} = u_{t-1,k-2} + a_{t-1} g_1 + b_{t-1} h_1 + a_t g_k + b_t h_k.$

♣ Multiple-Output Shift Registers. In practice, the encoding of a message often uses more than one code. For example, two codes are used in the encoding of music on the compact discs, where both codes are *Reed Solomon* codes, and two codes are used by *NASA* and the European Space Agency, where one code is a *Reed-Solomon* code, the other a *convolution* code. A two-output encoder shown in Figure 12 produces two different codewords for the same message.

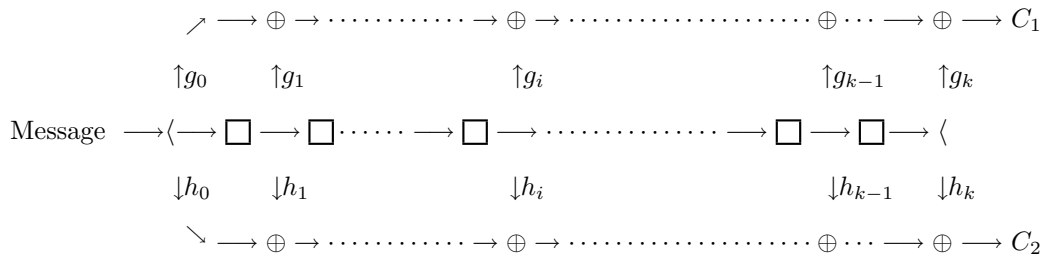


Figure 12. A message generated into two different codewords

The shift matrix and the vectors for the circuit in Figure 12 are as follows:

$$S_g = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}, R_g = [1 \ 0 \ 0 \ \dots \ 0], Q_g = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix}, \text{ and } L_g = \begin{bmatrix} g_0 & h_0 \\ g_1 & h_1 \\ \vdots & \vdots \\ g_{k-1} & h_{k-1} \end{bmatrix}.$$

♣ Polynomial Division. In order to decode a received word w , we must use a division at some point. Polynomial division (and thus polynomial decoding for cyclic codes) can be implemented by (\mathcal{LFSR}). A circuit for dividing $d(x) = d_n x^n + d_{n-1} x^{n-1} + \dots + d_1 x + d_0$ by a fixed polynomial $g(x) = g_0 + g_1 x + \dots + g_n x^n$ is shown in Figure 13.

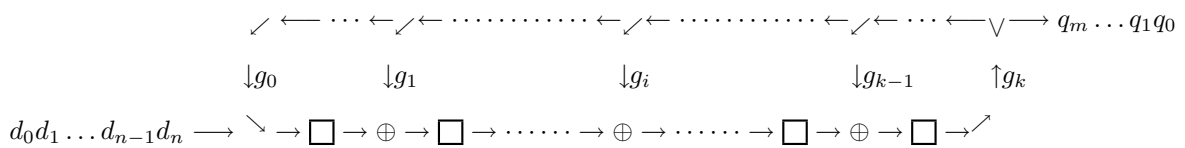


Figure 13. A circuit for dividing $d(x)$ by $g(x)$

High-order coefficients of the dividend polynomial $d(x)$ must be entered first, the output will be the quotient polynomial $q(x) = q_0 x^m + q_1 x^{m-1} + \dots + q_{m-2} x + q_{m-1}$, where the higher coefficients are coming out first (the last output q_m is not a coefficient of the quotient $q(x)$). The contents of the registers at the last step m will be the remainder $r(x)$; that is

$$u_m = [u_{m,0}, u_{m,1}, \dots, u_{m,k-1}] \implies r(x) = u_{m,0} + u_{m,1}x + \dots + u_{m,k-1}x^{k-1}.$$

The Companion matrix C_g of $g(x)$ is the shift matrix S_g and the state and output equations are as follows:

$$u_t = u_{t-1} S_g + v_t R_g = [u_{t-1,0}, u_{t-1,1}, \dots, u_{t-1,k-1}] \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 1 \\ g_0 & g_1 & \dots & g_{k-2} & g_{k-1} \end{bmatrix} + v_t [1 \ 0 \ 0 \ \dots \ 0]$$

$$\text{and } w_t = u_{t-1} Q_g = [u_{t-1,0}, u_{t-1,1}, \dots, u_{t-1,k-1}] \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Example 6. Consider the linear cyclic code generated by the polynomial $g(x) = 1 + x + x^3$ and let $w(x) = x + x^2 + x^4$ be the received polynomial. The \mathcal{LFSR} circuit shown in Figure 14 will find the syndrome $s(w)$.

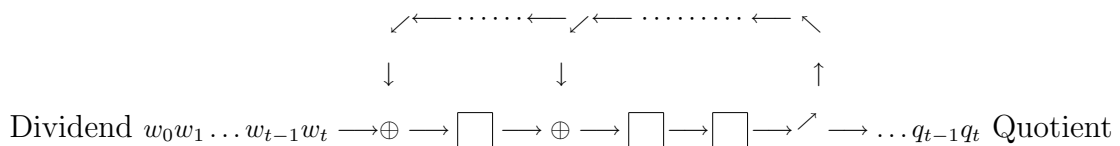


Figure 14. A circuit to divide by $g(x) = 1 + x + x^3$

So $10110 = w_4w_3w_2w_1w_0 \longleftrightarrow x^4 + x^2 + x$ will be fed into the \mathcal{LFSR} circuit. The result is suberized in the following table:

$g(x) = 1 + x + x^3$					
time	input	s_0+c_t	s_1+c_t	s_2	$c_t = \text{ouput}$
0	—	0	0	0	—
1	1	1	0	0	0
2	0	0	1	0	0
3	1	1	0	1	1 ↓
4	1	1+1	1+1	0	0
5	0	0 →	0 →	0	0
				remainder	quotient

Table 6. Division of $x^4 + x^2 + x$ by $x^3 + x + 1$

The quotient is $x = x + 0 \times 1 \longleftrightarrow 10$, corresponding to the output sequence. The remainder is $r(x) = s_0(5) + s_1(5)x + s_2(5)x^2 = 0 \longleftrightarrow 000$; so the syndrome $s(w) = 000$; thus w is a codeword.

Example 7. Now let $w(x) = 1 + x^3 + x^5$ be the received polynomial and 1001010 be the corresponding word. Then we have the following table:

$g(x) = 1 + x + x^3$					
time	input	s_0+c_t	s_1+c_t	s_2	$c_t = \text{ouput}$
0	—	0	0	0	—
1	1	1	0	0	0
2	0	0	1	0	0
3	1	1	0	1	1 ↓
4	0	0+1	1+1	0	0
5	0	0	1	0	0
6	1	1 →	0 →	1	0
				remainder	quotient

Table 7. Division of $x^5 + x^3 + 1$ by $x^3 + x + 1$

The quotient is $x^2 = x^2 + 0 \times x + 0 \times 1 \longleftrightarrow 100$ and the remainder is $1 + x^2 \longleftrightarrow 101 \neq 000$; in this case w is not a codeword.

♣ Simultaneously Multiplying and Dividing Polynomials. A single shift register circuit that multiplies a polynomial $a(x)$ by $g(x)$ and then divides by $h(x)$ can be made by combining the multiplication circuit of Figure 7 and the division circuit of Figure 13 as shown in Figure 15.

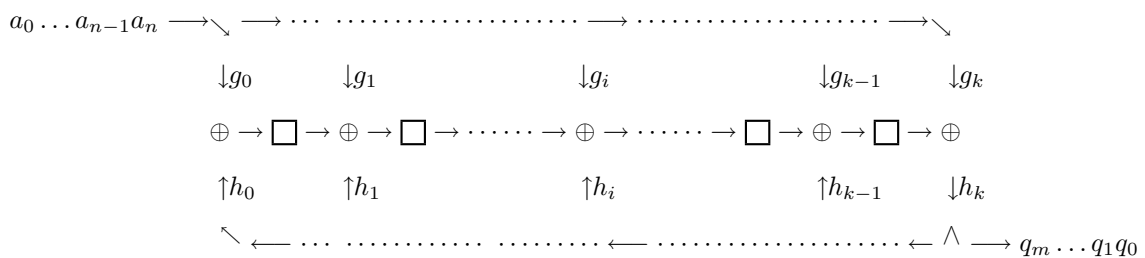


Figure 15. A circuit for $\frac{a(x)g(x)}{h(x)}$

In this circuit it is assumed that $\text{degree } g(x) \leq \text{degree } h(x)$. The shift matrix $S_{g,h}$ for this circuit is the companion matrix of the polynomial $h(x)$. High-order coefficients of $a(x)$ must be entered first, the output will be $q(x) = q_0x^m + q_1x^{m-1} + \dots + q_{m-1}x + q_m$ and the contents of the registers at the last step will be $r(x) = r_0 + r_1x + \dots + r_{k-1}x^{k-1}$ satisfying the equation:

$$a(x)g(x) = q(x)h(x) + r(x),$$

Example 8. A diagram of the shift register circuit for multiplying any polynomial $a(x)$ by $g(x) = 1 + x + x^5$ and then dividing by $h(x) = 1 + x^3 + x^4 + x^5 + x^6$ is shown in Figure 16.

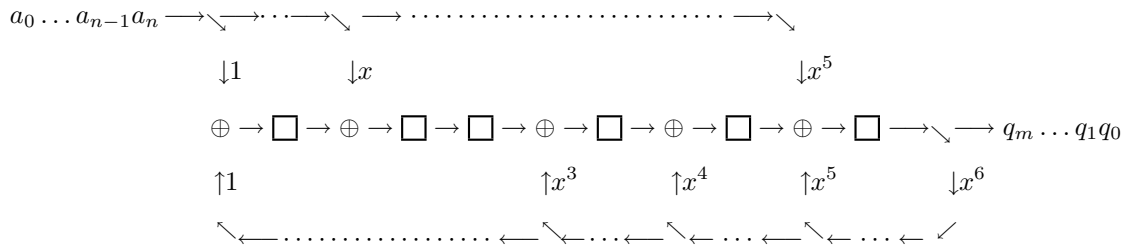


Figure 16. A circuit for $\frac{a(x)(1 + x + x^5)}{1 + x^3 + x^4 + x^5 + x^6}$

The different states and outputs of the circuit in Figure 15 for the input polynomial $a(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ are shown Table 8.

t	v_t	$s_{0,g,h}$	$s_{1,g}$	s_2	$s_{3,h}$	$s_{4,h}$	$s_{5,h}$	output
0	—	0	0	0	0	0	0	0
1	a_4	a_4	a_4	0	0	0	a_4	0
2	a_3	$a_3 + a_4$	$a_3 + a_4$	a_4	a_4	a_4	$a_3 + a_4$	a_4
3	a_2	$a_2 + a_3 + a_4$	$a_2 + a_3 + a_4$	$a_3 + a_4$	a_3	a_3	$a_2 + a_3$	$a_3 + a_4$
4	a_1	$a_1 + a_2 + a_3$	$a_1 + a_2 + a_3 + a_4$	$a_2 + a_3 + a_4$	$a_2 + a_4$	a_2	$a_1 + a_2$	$a_2 + a_3$
5	a_0	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3$	$a_1 + a_2 + a_3 + a_4$	$a_1 + a_3 + a_4$	$a_1 + a_4$	$a_0 + a_1$	$a_1 + a_2$

Table 8. $a(x)(1 + x + x^5)/(1 + x^3 + x^4 + x^5 + x^6)$

In the above table, $s_{0,g,h}$ indicates that the register s_0 accepts values from both $g(x)$ and $h(x)$ while s_1 accepts values from s_0 and $g(x)$, so it is marked $s_{1,g}$. All the other registers except s_2 accept values from $h(x)$. Note that s_2 has no input except from s_1 ; also note that the output is fed back into s_0 .

By using Table 8, we obtain the following equation:

$$(a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4)(1 + x + x^5) = q(x)(1 + x^3 + x^4 + x^5 + x^6) + r(x),$$

where the output is the quotient

$$q(x) = a_4x^3 + (a_3 + a_4)x^2 + (a_2 + a_3)x + (a_1 + a_2)$$

and the last row of Table 8 is the remainder

$$r(x) = (a_0 + a_1 + a_2) + (a_0 + a_1 + a_2 + a_3)x + (a_1 + a_2 + a_3 + a_4)x^2 + (a_1 + a_3 + a_4)x^3 + (a_1 + a_4)x^4 + (a_0 + a_1).$$

Here are the equations for the t-state u_t and the output w_t .

$$u_t = u_{t-1} S_{g,h} + v_t R_{g,h} = [u_{t-1,0}, u_{t-1,1}, \dots, u_{t-1,5}] \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} + v_t [1 \ 1 \ 0 \ 0 \ 0 \ 1]$$

and $w_t = u_{t-1} Q_{g,h} = [u_{t-1,0}, u_{t-1,1}, \dots, u_{t-1,5}] \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = u_{t-1,5}.$

To design a shift register circuit, where the constant factor $g(x)$ has higher degree than the divisor $h(x)$ (i.e., $d_g = \text{degree } g(x) < \text{degree } h(x) = d_h$), we add $d_g - d_h$ registers at the low-order end of the circuits. In order to complete the division, after entering all the coefficients of $a(x)$, we must input $d_g - d_h$ 0's.

Example 9. A diagram of the shift register circuit for multiplying any polynomial $a(x)$ by $g(x) = 1 + x + x^5$ and then dividing by $h(x) = 1 + x^3 + x^4 + x^5 + x^6$ is shown in Figure 16.

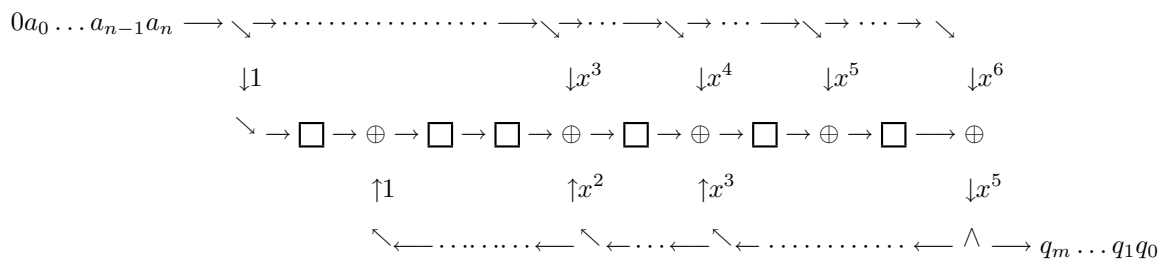


Figure 16. A circuit for $\frac{a(x)(1 + x^3 + x^4 + x^5 + x^6)}{1 + x^2 + x^3 + x^5}$

Exercises.

1. First draw the diagram for the shift registers corresponding to generator polynomial $g(x)$, then encode $a(x) = 1 + x + x^2$ and decode $w(x)$ using the proper $h(x)$:
 - (a) $g(x) = 1 + x^2 + x^3 \in \mathbb{K}^7[x]$ and $w(x) = 1 + x^4 + x^6$.
 - (b) $g(x) = 1 + x^3 + x^6 \in \mathbb{K}^9[x]$ and $w(x) = 1 + x^3 + x^5 + x^7$.
2. Generate pseudo-random numbers using the primitive polynomial $g(x) = 1 + x^3 + x^4$.