Linear-feedback shift registers for encoding and decoding cyclic codes

Cyclic codes and linear-feedback shift registers lend themselves to similar mathematical treatment and as we shall see the encoding and decoding of cyclic codes can be realized in terms of linear-feedback shift registers. The task is simplified by the fact that part of the decoding circuit is the same as that of the encoding circuit, this being the polynomial-division register. As we shall see, once the polynomial-division register is established, it is relatively simple to construct circuits for encoding and for decoding.

4.1 Linear-feedback shift registers

Figure 4.1 shows an *r-stage shift register* capable of storing r bits in its stages $b_0, b_1, \ldots, b_{r-1}$. Each stage is a 1-bit memory element which, when *shifted*, forwards its contents to the next stage and is then updated by the contents of the previous stage. Each time the register is shifted a bit b_{in} enters the register and all the bits in the register move along by one stage, with the bit in the last stage b_{r-1} leaving the register and forming the output b_{out}. At each shift the contents of the ith stage is fed to the $(i+1)$th stage and then the ith stage is updated with the contents of the $(i-1)$th stage. This is expressed as

$$\begin{aligned} b_{i+1} &= b_i \\ b_i &= b_{i-1} \end{aligned} \tag{4.1}$$

where b_i is used to represent the ith stage or the contents of the ith stage (this should not cause confusion).

The order in which the two expressions in eqns 4.1 are carried out is significant. If the order is reversed, so that the stage b_i is first updated with the contents of b_{i-1} and then b_{i+1} is updated, the original contents of b_i will be lost and b_{i+1} will end up containing the contents of b_{i-1}. For an r-stage shift register eqns 4.1 can be extended to the whole register to describe the changes in the stages at each shift, this gives

$$\begin{aligned} b_{out} &= b_{r-1} \\ b_{r-1} &= b_{r-2} \\ &\vdots \quad \vdots \\ b_2 &= b_1 \\ b_1 &= b_0 \\ b_0 &= b_{in} \end{aligned} \tag{4.2}$$

where b_{in} and b_{out} are the bits entering and leaving the shift register respectively. Equations 4.2 describe the operation of the register and give the *state* of the register

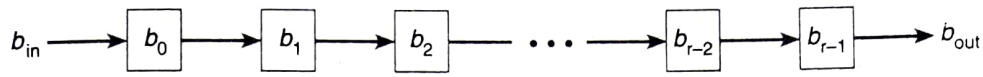

Fig. 4.1 An r-stage shift register.

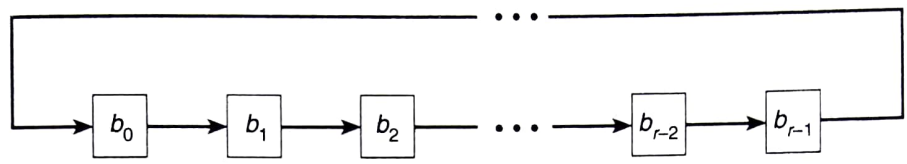

Fig. 4.2 An r-stage linear-feedback shift register.

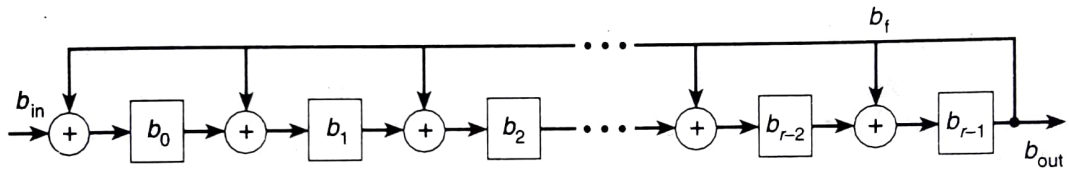

Fig. 4.3 An r-stage LFSR with feedback to all stages.

at each shift. The stages b_0 and b_{r-1} are referred to as the *low-order* and *high-order* sides of the register respectively.

In a *linear-feedback shift register* (*LFSR*) the output b_{out} is fed back into the register. The simplest example of such a register is shown in Fig. 4.2 which consists of r stages. Here the output bit is fed back into the first stage b_0 and so the stage contents are cyclically moved around the register as the register is shifted. If the stages are initialized with a codeword $\boldsymbol{c}$ from a cyclic code of blocklength r, then $r-1$ shifts will produce the $r-1$ codewords that are cyclic shifts of the codeword $\boldsymbol{c}$. At the rth shift the stages will again contain $\boldsymbol{c}$.

The feedback within a register can be used to alter any of the stages directly and not just b_0. Figure 4.3 shows a linear-feedback shift register with r stages and feedback, via modulo-2 adders, to all stages. Each modulo-2 adder has two, or more, inputs and one output that is the sum modulo-2 of the inputs. In practice some stages will have feedback and others will not, depending upon requirements. Denoting the feedback bit by b_f then stages with a feedback link change according to

$$b_i = b_{i-1} + b_f \tag{4.3}$$

where addition is modulo-2, and

$$b_i = b_{i-1} \tag{4.4}$$

describes changes to stages without a feedback link.

Example 4.1

Figure 4.4 shows a LFSR with 5 stages and feedback to 3 of the stages. The operation of this register is given by

$$\begin{aligned} b_{out} &= b_f = b_4 \\ b_4 &= b_3 \\ b_3 &= b_2 + b_f \\ b_2 &= b_1 + b_f \\ b_1 &= b_0 \\ b_0 &= b_{in} + b_f. \end{aligned}$$

Table 4.1 gives the operation of the register when the input is the 10-bit word (1 0 1 0 1 0 0 0 1 1) and where the left-hand bit enters the register first. The stages are initialized to zero before the first bit enters the register. For the first 5 shifts the feedback is zero as the input has yet to reach the high-order stage b_4, and so at the end of the 5th shift the stages simply contain the first 5 bits of the incoming word (for clarity these are underlined). It is only when the first nonzero bit reaches the high-order stage that feedback begins. At the end of the 5th shift we have $b_4 = 1$ and therefore at the next shift $b_f = 1$ and there is feedback to b_3, b_2, and b_0. Once all the bits have entered the register, the final state of the register is (0 0 1 0 0) (also shown underlined).

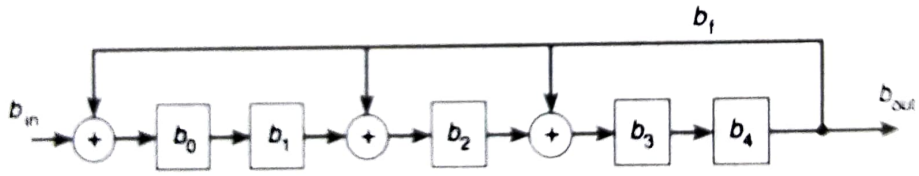

Fig. 4.4 A 5-stage LFSR.

Table 4.1
Step-by-step operation of the LFSR in Fig. 4.4

Input $v = (1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1)$

Shift	b_{in}	b_0	b_1	b_2	b_3	b_4	b_f	
		0	0	0	0	0	0	←Initial state
1	1	1	0	0	0	0	0	
2	0	0	1	0	0	0	0	
3	1	1	0	1	0	0	0	
4	0	0	1	0	1	0	0	
5	1	1	0	1	0	1	0	
6	0	1	1	1	0	0	1	
7	0	0	1	1	1	0	0	
8	0	0	0	1	1	1	0	
9	1	0	0	1	0	1	1	
10	1	0	0	1	0	0	1	←Final state

4.2 The polynomial-division register

The encoding and decoding of cyclic codes involves the division of a polynomial $p(x)$ by a generator polynomial $g(x)$ giving a remainder $r(x)$. When encoding, $p(x)$ is an information polynomial $i(x)$ and $r(x)$ gives the parity-check bits required to construct codewords. At the decoding stage $p(x)$ is the received polynomial $v(x)$ and $r(x)$ gives the syndrome polynomial. Encoding and decoding use the same *polynomial-division register* and so we consider this before addressing encoding and decoding specifically.

Figure 4.5 shows a linear-feedback shift register for dividing an arbitrary polynomial

$$p(x) = p_{n-1}x^{n-1} + p_{n-2}x^{n-2} + \cdots + p_2x^2 + p_1x + p_0$$

by a fixed polynomial

$$g(x) = g_rx^r + g_{r-1}x^{r-1} + \cdots + g_2x^2 + g_1x + g_0$$

where $p(x)$ and $g(x)$ are referred to as the dividend and divisor respectively. The switches in Fig. 4.5 represent the polynomial coefficients g_i and are open or closed depending on the coefficients. If $g_i = 1$ then the corresponding switch is closed so giving a feedback link to the ith stage, and changes in the ith stage are given by $b_i = b_{i-1} + b_f$ (eqn 4.3). If $g_i = 0$ then there is no feedback to the ith stage, the adder feeding into b_i can then be omitted, as it has only 1 input, and so $b_i = b_{i-1}$ (eqn 4.4). The feedback link g_r is always present in the register, as a polynomial $g(x)$ of degree r has $g_r = 1$ by definition.

The polynomial coefficients p_i are shifted into the register starting with the high-order coefficient p_{n-1}, followed by p_{n-2}, p_{n-3}, and so forth. For the first r shifts the feedback and output are zero as the dividend has yet to reach the high-order stage b_{r-1}. Once the first nonzero bit of the dividend reaches b_{r-1} feedback commences and it is at this point that division starts. The high-order stage b_{r-1} represents an x^{r-1} term within $p(x)$, and a 1 leaving this stage means that an x^r term needs to be subtracted from $p(x)$. This is achieved by the feedback circuit whose links to the stages represent the divisor $g(x)$. This is analogous to the way in which the divisor is subtracted from the dividend when carrying out long division. If the output of b_{r-1} is zero then no subtraction takes place. On shifting the last bit p_0 into the register the remainder of $p(x)$ divided by $g(x)$ is left in the stages. Note that because the stages $b_0, b_1, \ldots, b_{r-2}, b_{r-1}$

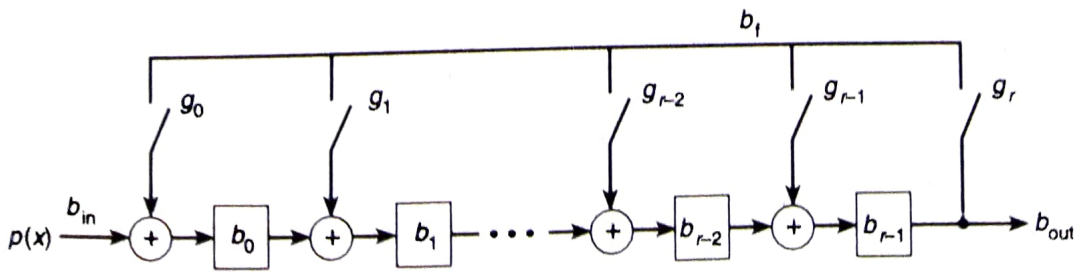

Fig. 4.5 A polynomial-division register.

contain the remainder of the input divided by $g(x)$ we refer to them as the *remainder stages*. If we let

$$r(x) = b_{r-1}x^{r-1} + b_{r-2}x^{r-2} + \cdots + b_2x^2 + b_1x + b_0$$

be a polynomial representing the stages then, using the notation introduced in Section 3.2, we have

$$r(x) = R_{g(x)}[p(x)].$$

Furthermore, the output from the register gives the quotient $q(x)$, where the first nonzero output determines the degree of $q(x)$. Let's consider dividing

$$p(x) = x^7 + x^6 + x^2 + x + 1$$

by

$$g(x) = x^4 + x^2 + 1.$$

The coefficients of the divisor $g(x)$ are $g_4 = 1$, $g_3 = 0$, $g_2 = 1$, $g_1 = 0$, and $g_0 = 1$. The linear-feedback shift register required is shown in Fig. 4.6. The dashed lines show the absence of feedback links due to $g_1 = 0$ and $g_3 = 0$. With each shift the stages change according to

$$\begin{aligned} b_{\text{out}} &= b_{\text{f}} = b_3 \\ b_3 &= b_2 \\ b_2 &= b_1 + b_{\text{f}} \\ b_1 &= b_0 \\ b_0 &= b_{\text{in}} + b_{\text{f}}. \end{aligned}$$

The coefficients of the dividend are $p_7 = 1$, $p_6 = 1$, $p_5 = 0$, $p_4 = 0$, $p_3 = 0$, $p_2 = 1$, $p_1 = 1$, and $p_0 = 1$, and these enter the register high-order bits first, p_7 followed by p_6 and so forth. Table 4.2 shows the division step-by-step for all the 8 shifts required to feed $p(x)$ into the register, the polynomial representation of the remainder stages, $r(x)$, is also shown in Table 4.2. The input is underlined for the first 4 shifts so that we can clearly see its progress into the register. It takes the first 4 shifts to move p_7 to the high-order stage b_3, during this time there is no feedback or output. On the 5th shift p_7 leaves b_3 therefore giving a nonzero feedback and output, so that $b_{\text{f}} = 1$ and $b_{\text{out}} = 1$, division has now started. By the end of the 8th shift all the polynomial coefficients have entered the register and the stages contain the remainder

$$r(x) = b_3x^3 + b_2x^2 + b_1x + b_0.$$

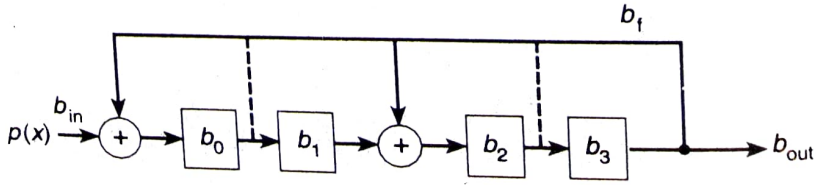

Fig. 4.6 An LFSR for dividing by $x^4 + x^2 + 1$.

Table 4.2
Operation of the LFSR in Fig. 4.6

Dividing $x^7+x^6+x^2+x+1$ by x^4+x^2+1

Shift	b_{in}	b_0	b_1	b_2	b_3	b_f	$r(x)$
		0	0	0	0	0	0
1	1	1	0	0	0	0	1
2	1	1	1	0	0	0	$x+1$
3	0	0	1	1	0	0	x^2+x
4	0	0	0	1	1	0	x^3+x^2
5	0	1	0	1	1	1	x^3+x^2+1
6	1	0	1	1	1	1	x^3+x^2+x
7	1	0	0	0	1	1	x^3
8	1	0	0	1	0	1	x^2

As we can see from Table 4.2 (last row, underlined) only b_2 is nonzero and therefore $r(x)=x^2$ is the remainder of $p(x)=x^7+x^6+x^2+x+1$ divided by $g(x)=x^4+x^2+1$. Furthermore the register output is (0 0 0 0 1 1 1 1) which gives the quotient $q(x)=x^3+x^2+x+1$. We can check the answer by constructing $p(x)$ from $r(x)$, $q(x)$, and $g(x)$. Using the Euclidean division algorithm

$$p(x) = q(x)g(x) + r(x)$$

and substituting $q(x)$, $g(x)$, and $r(x)$ gives

$$\begin{aligned} p(x) &= (x^3+x^2+x+1)(x^4+x^2+1)+x^2 \\ &= x^7+x^6+x^2+x+1 \end{aligned}$$

as required. We could have checked the answer by dividing $p(x)$ by $g(x)$ using the long division method, however the Euclidean division algorithm involves multiplication, and not division, and is therefore a bit easier.

Example 4.2

By considering the step-by-step operation of the register shown in Fig. 4.7 find the remainder and quotient when the input to the register is $p(x)=x^6+x^5+x^4+x+1$.

From Fig. 4.7 we can see that the register has 3 stages and therefore the divisor has degree 3 and is of the form $g(x)=x^3+g_2x^2+g_1x+g_0$. Stages b_0 and b_1 have feedback links whereas b_2 does not and so $g_0=g_1=1$, $g_2=0$ which gives $g(x)=x^3+x+1$ as the divisor. Also from Fig. 4.7 we get

$$\begin{aligned} b_{out} &= b_f = b_2 \\ b_2 &= b_1 \\ b_1 &= b_0 + b_f \\ b_0 &= b_{in} + b_f. \end{aligned}$$

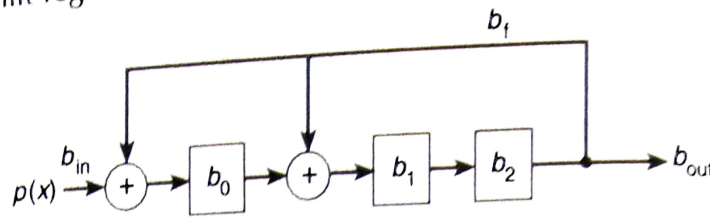

Fig. 4.7 An LFSR for dividing by $g(x) = x^3 + x + 1$.

Table 4.3
Operation of the LFSR in Fig. 4.7

Dividing $x^6 + x^5 + x^4 + x + 1$ *by* $x^3 + x + 1$

Shift	b_{in}	b_0	b_1	b_2	b_f	$r(x)$
	0	0	0	0	0	
1	1	1	0	0	0	1
2	1	1	1	0	0	$x+1$
3	1	1	1	1	0	x^2+x+1
4	0	1	0	1	1	x^2+1
5	0	1	0	0	1	1
6	1	1	1	0	0	$x+1$
7	1	1	1	1	0	x^2+x+1

Table 4.3 shows the step-by-step operation as $p(x)$ enters the register. After the 7th shift $p(x)$ has completely entered the register, and the remainder stages are (1 1 1) giving the remainder $r(x) = x^2 + x + 1$. The output, as given by the feedback bit, gives the quotient $q(x) = x^3 + x^2$. The reader may wish to check the answers, using either the long division method or the Euclidean division algorithm. □

In the polynomial-division register the remainder at each shift can be determined from the previous remainder. If $r_i(x)$ is the remainder after the ith shift, then at the next shift the contents of the stages move 1 place towards the high-order side of the register, which is equivalent to multiplying $r_i(x)$ by x, and the feedback circuit divides by $g(x)$. The remainder after the $(i+1)$th shift is therefore

$$r_{i+1}(x) = R_{g(x)}[xr_i(x)] \tag{4.5}$$

where it is assumed that the input to the register is 0. If the input is 1, then 1 is added to the remainder. The degree of $r_i(x)$ is always less than or equal to $r-1$ and therefore the degree of $xr(x)$ cannot exceed the degree r of the divisor $g(x)$. Whenever $xr(x)$ results in an x^r term, so that it has the same degree as $g(x)$, the feedback circuit has the effect of adding (or subtracting, remember that we are using modulo-2 addition) $g(x)$ to $xr(x)$ and therefore removing the x^r term, this corresponds to division by $g(x)$. Equation 4.5 can be evaluated by multiplying $r_i(x)$ by x, and then adding $g(x)$ if $xr_i(x)$ contains an x^r term. If the register input is 1, then this is added to the remainder.

The polynomial-division register shown in Fig. 4.5 requires r shifts before the dividend reaches the high-order side of the register, which represents idle or wasted time as no division takes place during these shifts. This can be avoided by feeding the input directly into the high-order side of the register. By doing so feedback, and therefore division, commences immediately with the input of the first nonzero bit. However, this does not give the remainder of the input $p(x)$ divided by $g(x)$ but rather of $p'(x) = x^r p(x)$ divided by $g(x)$. This can be useful when carrying out encoding, as it corresponds to multiplication by x^{n-k} required when generating systematic codewords. At the decoding stage there is no requirement for multiplication by x^r and a register with input to the high-order side will give the wrong error-syndrome. Despite this, however, it is still possible to feed the input into the high-order side when decoding (this is discussed further in Section 4.5). Registers with inputs to the high-order side and low-order side are referred to as having *high-order input* and *low-order input* respectively.

A linear-feedback shift register with high-order input has the same feedback circuit as that with the input to the low-order side. It is only the input to b_0 and the feedback bit b_f that are affected. The feedback bit now depends on the output from the high-order stage b_{r-1} and the input bit b_{in}, while the input to b_0 is taken solely from the feedback. Hence $b_f = b_{r-1} + b_{in}$ and $b_0 = b_f$ compared to $b_f = b_{r-1}$ and $b_0 = b_f + b_{in}$ when the input is to the low-order side. The inputs to all the other stages remain the same and bits are still shifted from the low-order side of the register to the high-order side. Let's reconsider the problem considered in Example 4.2, namely dividing $p(x) = x^6 + x^5 + x^4 + x + 1$ by $g(x) = x^3 + x + 1$ but this time using a linear-feedback shift register with a high-order input. The linear-feedback shift register required to achieve this is shown in Fig. 4.8. The operation of the register is now described by

$$\begin{aligned} b_{out} &= b_f = b_2 + b_{in} \\ b_2 &= b_1 \\ b_1 &= b_0 + b_f \\ b_0 &= b_f \end{aligned}$$

and using these or eqn 4.5 gives the results shown in Table 4.4. Note that if eqn. 4.5 is used to determine the remainder at each shift, then each nonzero input contributes an x^3 term to the remainder. During the first shift $p_6 = 1$ enters the register giving a nonzero feedback $b_f = 1$ and therefore division starts immediately. Note that because division starts with the first shift we cannot identify the information bits entry into the stages, so only the final remainder is underlined in Table 4.4. By the

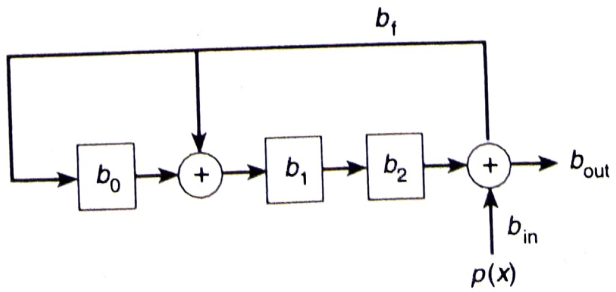

Fig. 4.8 An LFSR, with high-order input, for dividing by $g(x) = x^3 + x + 1$.

Table 4.4
Operation of the LFSR in Fig. 4.8

Dividing $x^6+x^5+x^4+x+1$ *by* $g(x)=x^3+x+1$

Shift	b_{in}	b_0	b_1	b_2	b_f	$r(x)$
	0	0	0	0	0	
1	1	1	1	0	1	$x+1$
2	1	1	0	1	1	x^2+1
3	1	0	1	0	0	x
4	0	0	0	1	0	x^2
5	0	1	1	0	1	$x+1$
6	1	1	0	1	1	x^2+1
7	1	0	1	0	0	x

end of the 7th shift the register contents are (0 1 0) giving the remainder $r(x)=x$ and the output gives the quotient $q(x)=x^6+x^5+x^2+x$. The remainder and quotient obtained are different from those obtained in Example 4.2 because $r(x)$ and $q(x)$ obtained here are not the remainder and quotient of $p(x)=x^6+x^5+x^4+x+1$ divided by $g(x)=x^3+x+1$, but rather of the polynomial $p'(x)=x^r p(x)=x^3p(x)$ divided by $g(x)$. We can illustrate this by using the Euclidean division algorithm to construct $p'(x)$ from $r(x)$, $q(x)$, and $g(x)$:

$$\begin{aligned} p'(x) &= q(x)g(x)+r(x) \\ &= (x^6+x^5+x^2+x)(x^3+x+1)+x \\ &= x^9+x^8+x^7+x^4+x^3. \end{aligned}$$

Furthermore taking out a factor of x^3 gives

$$p'(x)=x^3(x^6+x^5+x^4+x+1)$$

and so

$$p'(x)=x^3p(x)$$

as required. Therefore the results obtained with the register shown in Fig. 4.8 are the remainder and quotient of $x^3p(x)$ divided by $g(x)$, and not $p(x)$ divided by $g(x)$.

4.3 Registers for encoding

Encoding cyclic codes can be achieved in a variety of ways using shift registers. Here we look at two circuits based on the polynomial-division register considered in the previous section. The first circuit has an input to the low-order side and the second to the high-order side. Both circuits produce systematic codewords.

Figure 4.9 shows an encoder for an (n, k) cyclic code with generator polynomial

$$g(x)=g_{n-k}x^{n-k}+g_{n-k-1}x^{n-k-1}+\cdots+g_2x^2+g_1x+g_0$$