commonly called the *Header Error Check* (HEC) field, is defined by the polynomial $x^8 + x^2 + x + 1$. The HEC is implemented with an eight-bit *linear feedback shift register* (LFSR) as shown in Fig. 9.10. Bytes are shifted into the LFSR one bit at a time, starting with the MSB. Input data and the last CRC bit feed to the XOR gates that are located at the bit positions indicated by the defining polynomial. After each byte has been shifted in, a CRC value can be read out in parallel with the LSB and MSB at the positions shown. When a new CRC calculation begins, this CRC algorithm specifies that the CRC register be initialized to 0x00. Not all CRC algorithms start with a 0 value; some start with each bit set to 1.

The serial LFSR can be converted into a set of parallel equations to enable practical implementation of the HEC on byte-wide interfaces. The general method of deriving the parallel equations is the same as done previously for the scrambling polynomial. Unfortunately, this is a very tedious process that is prone to human error. As CRC algorithms increase in size and complexity, the task can get lengthy. LFSRs may be converted manually or with the help of a computer program or spreadsheet. Table 9.7 lists the XOR terms for the eight-bit HEC algorithm wherein a whole byte is clocked through each cycle. Each CRC bit is referred to as Cn, where n = [7:0]. Once the equations are simplified, matching pairs of CRC and data input bits are found grouped together. Therefore, the convention Xn is adopted where Xn = Cn XOR Dn to simplify notation. Similar Boolean equations can be derived for arbitrary cases where fractions of a byte (e.g., four bits) are clocked through each cycle, or where multiple bytes are clocked through in the case of a wider data path.

TABLE 9.7 Simplified Parallel HEC Logic

CRC Bits	XOR Logic
C0	X0 X6 X7
C1	X0 X1 X6
C2	X0 X1 X2 X6
C3	X1 X2 X3 X7
C4	X2 X3 X4
C5	X3 X4 X5
C6	X4 X5 X6
C7	X5 X6 X7

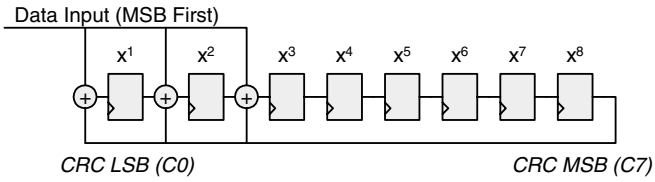

FIGURE 9.10 HEC LFSR.

When a new HEC calculation is to be started, the CRC state bits are reset to 0. Each byte is then clocked through the parallel logic at the rate of one byte per cycle. Following the final data byte, the HEC is XORed with 0x55 to yield a final result. An arbitrary number of bytes can be clocked through, and the CRC value will change each cycle. The one exception to this is the case of leading 0s. Because the HEC specifies a reset state of 0, passing 0x00 data through the CRC logic will not result in a nonzero value. However, once a nonzero value has been clocked through, the LFSR will maintain a nonzero value in the presence of a stream of 0s. This property makes the HEC nonideal for checking arbitrary strings of leading 0s, and it is a reason why other CRC schemes begin with a nonzero reset value. Table 9.8 shows an example of passing four nonzero data bytes through the parallel HEC logic and then XORing with 0x55 to determine a final CRC value.

TABLE 9.8 Examples of HEC Calculation

Data Input	HEC Value
(Initialization)	0x00
0x11	0x77
0x22	0xAC
0x33	0xD4
0x44	0xF9
XOR 0x55	0xAC

Another common CRC is the 16-bit polynomial appropriately called *CRC-16*. Its polynomial is $x^{16} + x^{15} + x^2 + 1$, and its LFSR implementation is shown in Fig. 9.11. As with the HEC, a CRC-16 can be converted to a parallel implementation. Because the CRC-16 is two bytes wide, its common implementations vary according to whether the data path is 8 or 16 bits wide. Of course, wider data paths can be implemented as well, at the expense of more complex logic. Table 9.9 lists the CRC-16 XOR terms for handling either one or two bytes per cycle.

Properly calculating a CRC-16 requires a degree of bit shuffling to conform to industry conventions. While this shuffling does not intrinsically add value to the CRC algorithm, it is important for all implementations to use the same conventions so that one circuit can properly exchange CRC codes with another. Unlike the HEC that shifts in data bytes MSB to LSB, the CRC-16 shifts in data bytes LSB to MSB. In the case of a 16-bit implementation, the high-byte, bits [15:8], of a 16-bit word is shifted in before the low-byte, bits [7:0], to match the standard order in which bytes are transmitted. What this means to the implementer is that incoming data bits must be flipped before being clocked through the parallel XOR logic. This doesn't actually add any logic to the task, and

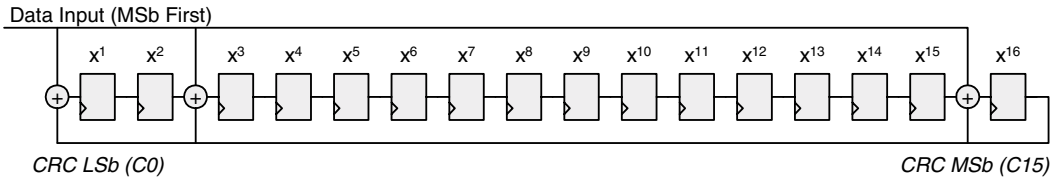

FIGURE 9.11 CRC-16 LFSR.