TABLE 9.9 CRC-16 Parallel Logic

CRC Bits	16-Bit XOR Logic	8-Bit XOR Logic
C0	X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X15	C8 C9 C10 C11 C12 C13 C14 C15 D0 D1 D2 D3 D4 D5 D6 D7
C1	X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14	C9 C10 C11 C12 C13 C14 C15 D1 D2 D3 D4 D5 D6 D7
C2	X0 X1 X14	C8 C9 D0 D1
C3	X1 X2 X15	C9 C10 D1 D2
C4	X2 X3	C10 C11 D2 D3
C5	X3 X4	C11 C12 D3 D4
C6	X4 X5	C12 C13 D4 D5
C7	X5 X6	C13 C14 D5 D6
C8	X6 X7	C0 C14 C15 D6 D7
C9	X7 X8	C1 C15 D7
C10	X8 X9	C2
C11	X9 X10	C3
C12	X10 X11	C4
C13	X11 X12	C5
C14	X12 X13	C6
C15	X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X14 X15	X7 C8 C9 C10 C11 C12 C13 C14 C15 D0 D1 D2 D3 D4 D5 D6

there are a couple of ways that this can be done. One approach is to explicitly renumber the XOR terms to perform the flipping intrinsically. Another approach, the one taken here, is to maintain a consistent XOR nomenclature and simply flip the bits between the input and the XOR functions. It is convenient to adopt a common bit ordering convention across different CRC implementations, and the XOR terms shown for the CRC-16 are written with the same convention used in the HEC logic: MSB first. The actual bit-flipping in hardware is translated to a renumbering of the XOR input terms by the logic implementation software being used without any penalty of additional gates.

Once the bits have been clocked through the XOR functions in the correct order, industry convention is that the CRC register itself is flipped bit-wise and, depending on the implementation, byte-wise as well. The bit-wise flipping is always performed, and the byte-wise flipping is a function of whether big-endian or little-endian ordering is used. Since all of this talk of bit shuffling may seem confusing, Table 9.10 shows a step-by-step example of calculating a CRC-16 16 bits at a time across the 32-bit data set 0x4D41524B using the big-endian convention.

Yet another CRC is the ubiquitous 32-bit *CRC-32*, which is used in Ethernet, FDDI, Fibre Channel, and many other applications. The CRC-32 polynomial is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$, and its LFSR implementation is shown in Fig. 9.12. Similar to

TABLE 9.10 Step-by-Step CRC16 Calculation

Operation	Data
Initialize CRC-16 state bits	0x0000
First word to be calculated	0x4D41
Reorder bytes to end high-byte first after bit-flipping	0x414D
Flip bits for LSB-first transmission of high-byte then low-byte	0xB282
Clock word through XOR logic	0xAF06
Second word to be calculated	0x524B
Reorder bytes to end high-byte first after bit-flipping	0x4B52
Flip bits for LSB-first transmission of high byte then low byte	0x4AD2
Clock word through XOR logic	0x5CF4
Flip bits of CRC	0x2F3A
Optionally swap bytes of CRC for final result	0x3A2F

the CRC-16, parallel CRC-32 logic is commonly derived for data paths of one, two, or four bytes in width. A difference between the CRC-32 and those CRC schemes already presented is that the CRC32's state bits are initialized to 1s rather than 0s, and the final result is inverted before being used. Table 9.11 lists the CRC-32 XOR terms for handling one, two, or four bytes per cycle.

As noted, the CRC-32 state bits are initialized with 1s before calculation begins on a new data set. Words are byte-swapped and bit-flipped according to the same scheme as done for the CRC-16. When the last data word has been clocked through the parallel logic, the CRC-32 state bits are inverted to yield the final calculated value. Table 9.12 shows a step-by-step example of calculating a CRC-32 32 bits at a time using the same 32-bit data set, 0x4D41524B, as before.

CRC algorithms can be performed in software, and often are when cost savings is more important than throughput. Due to their complexity, however, the task is usually done in hardware when high-speed processing is required. Most modern networking standards place one or more CRC fields into

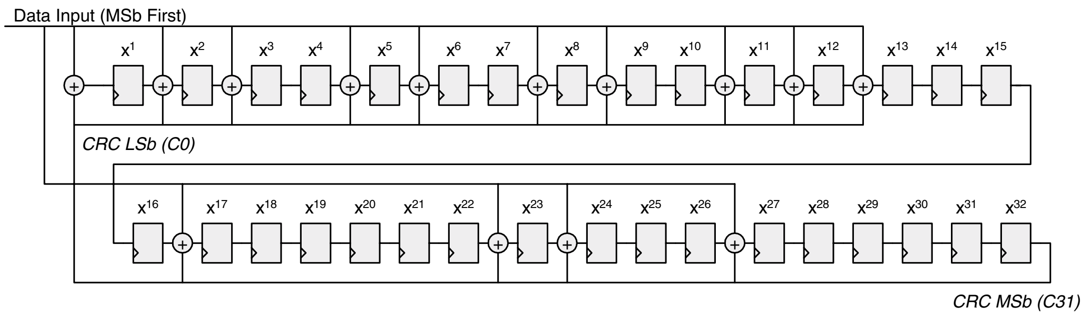

FIGURE 9.12 CRC-32 LFSR.