Parity Code	Advantages	Disadvantages
Bit-per-word: one parity bit per data word	Detects all single-bit errors	Certain errors undetected, e.g., a word, including parity bit becomes all 1s, due to a failure of a bus or a set of data buffers
Bit-per-byte: each data portion (e.g., a byte) is protected by a separate parity bit; the parity of one group should be even and the parity of the other group should be odd	Detects all-1s and all-0s conditions	Ineffective for multiple errors, e.g., the whole-chip failure
Bit-per-multiple-chips: one bit from each chip is associated with a single parity bit	Detects failure of entire chip	Cannot locate failure of complete chip
Bit-per-chip: each parity bit is associated with one chip of the memory	Detects single-bit errors and identifies chip with erroneous bit	Susceptible to whole-chip failure, i.e., a single chip error can result in multiple bits to be corrupted and this may go undetected
Interlaced: similar to the bit-per-multiple-chips; must ensure that no two adjacent bits are from the same parity group	Detects errors in adjacent bits	Parity groups are not based on physical organization of the memory

Table 1: Comparison of Parity Codes

In high-speed memories, single-bit error correcting and double-bit error detecting (SEC-DED) codes are most commonly used. The data before writing to the memory are passed to a parity generator. The generated parity bit (or bits) is (are) then stored in the memory together with the data. On read operation the data bits are passed into the parity checker that regenerates the parity bit (or bits) and compares it with the parity bit(s) stored in the memory when the original data were written to the memory. The following description brings more details on Hamming codes.

In Hamming single-error correction code, c parity bits are added to a k-bit data word, forming a codeword of $k+c$ bits. The following expression can be used to determine number of necessary check (parity) bits to protect k bits of information: $2^c \geq c + k + 1$.

Consider a data word of four information bits (d_3, d_2, d_1, d_0). According to the above expression, three parity bits (p_3, p_2, p_1) are needed to protect the four bits of data. To illustrate how the parity (check) bits are generated and checked, assume that the bits in the codeword are numbered from 1 to $k+c$. Positions numbered as a power of two are reserved for the parity bits. The grouping of bits for parity generation and checking is determined based on a list of the binary numbers from 0 to (2^k–1), as illustrated in Figure 8.

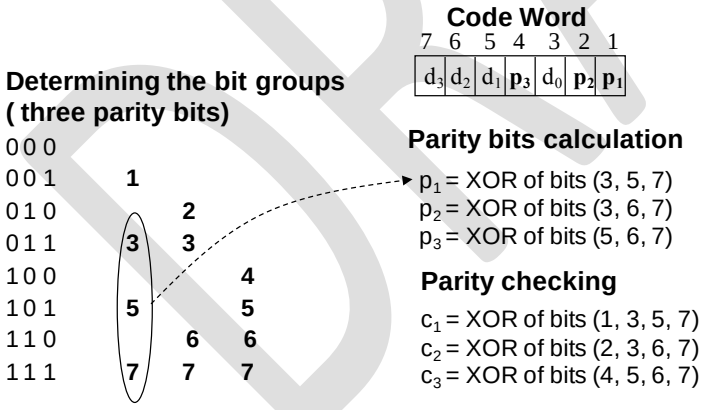

Figure 8: Determining Parity/Check Bits for Hamming Code

The first group is formed by the data bits in the positions corresponding to the 1-bits in the least significant bit of the binary count sequence (i.e., bits 1, 3, 5, 7). The second

group is formed by the data bits in the positions corresponding to the 1-bits in second significant bit of the binary count sequence (i.e., bits 2, 3, 6, 7), and so on. Note that each group of bits starts with the number that is a power of two. These numbers are also the position number (in a codeword) for the parity bits. The individual parity bits are calculated by performing an XOR operation on the data bits specified by a given group. For parity checking, the XOR operations also include the parity bit itself. As a result, the original data is encoded by generating a set of parity bits ($p_3p_2p_1$). To check correctness, the encoding process is repeated and a set of check bits ($c_3c_2c_1$) is generated. The binary word represented by the check bits $c_3c_2c_1$ forms a *syndrome*, which points directly to the position of the erroneous bit.

The Hamming code discussed above can only detect and correct single bit errors. By adding an extra parity bit, the Hamming code can be used to correct single bit errors and to detect double errors. In the example of a data word consisting of four information bits, the additional parity bit, p_0, can be calculated as parity (XOR) over the first seven bits of the codeword. For parity checking, the additional check bit c_0, is calculated over all eight bits of the codeword. Figure 9 illustrates the four cases distinguishable by single-error correction (SEC) and double-error detection (DED) Hamming code. In general, any code with an odd Hamming distance can be extended by adding a parity bit to form a code with a Hamming distance greater by one (and thus even).

c_3	c_2	c_1	c_0	
0	0	0	0	No errors
x_3	x_2	x_1	1	Single error (in a position $x_3x_2x_1$) is detected and can be corrected
y_3	y_2	y_1	0	Double error is detected but cannot be corrected
0	0	0	1	Error in parity bit p_0

Figure 9: Error Detection and Correction Using SEC-DED Code

4.5 Cyclic Redundancy Checks

Cyclic redundancy checks (CRCs) are used to detect errors in communication channels, tapes, and disks. Cyclic codes are parity check codes with the additional property that the cyclic shift of a codeword is also a codeword. The wide use of CRCs is due mainly to the following factors: (1) CRC calculators are simple to implement (the needed hardware includes linear feedback shift registers and XOR gates); in particular, CRC's have the advantage that the data can be streamed through the CRC calculator in both directions (send and receive) and when all the data have passed, depending on the direction, the CRC check bits or error detection syndromes are generated, (2) CRC has the ability to detect single-bit errors, multiple adjacent bit errors affecting fewer than *n-k* (for an (*n, k*) code) bits, all odd-bit errors (with proper choice of the generator polynomial), and burst transient errors (typical of communication applications) and (3) CRC codes and implementations are independent or message length, with the caveat that if we place a limit on the message length CRC can detect any two-bit error.