Parity Generation and Checking

Exclusive-OR functions are very useful in systems requiring error-detection and correction codes. As discussed in Section 1-7, a parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a *parity generator*. The circuit that checks the parity in the receiver is called a *parity checker*.

As an example, consider a 3-bit message to be transmitted together with an even parity bit. Table 4-4 shows the truth table for the parity generator. The three bits, x, y, and z, constitute the message and are the inputs to the circuit. The parity bit P is the output. For even parity, the bit P must be generated to make the total number of 1's even (including P). From the truth table, we see that P constitutes an odd function because it is equal to 1 for those minterms whose numerical values have an odd number of 1's. Therefore, P can be expressed as a three-variable exclusive-OR function:

$$P = x \oplus y \oplus z$$

The logic diagram for the parity generator is shown in Fig. 4-25(a).

The three bits in the message together with the parity bit are transmitted to their destination, where they are applied to a parity-checker circuit to check for possible errors in the transmission. Since the information was transmitted with even parity, the four bits received must have an even number of 1's. An error occurs during the transmission if the four bits received have an odd number of 1's, indicating that one bit has changed in value during transmission. The output of the parity checker, denoted by C, will be equal to 1 if an error occurs, that is, if the four bits received have an odd number of 1's. Table 4-5 is the truth table for the even-parity checker. From it we see that the function C consists of the eight minterms with binary numerical values having an odd number of 1's. This corresponds to the map of Fig. 4-24(a), which represents an odd function. The parity checker can be implemented with exclusive-OR gates:

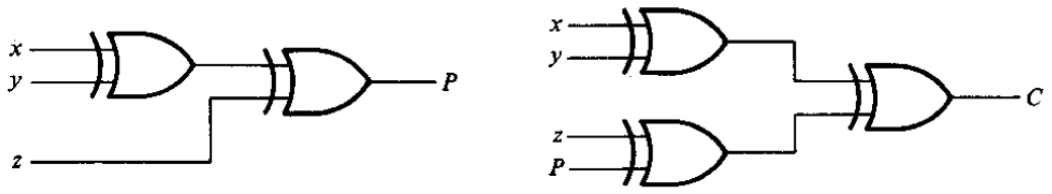
$$C = x \oplus y \oplus z \oplus P$$

TABLE 4-4
Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The logic diagram of the parity checker is shown in Fig. 4-25(b).

It is worth noting that the parity generator can be implemented with the circuit of Fig. 4-25(b) if the input P is connected to logic-0 and the output is marked with P. This is because $z \oplus 0 = z$, causing the value of z to pass through the gate unchanged. The advantage of this is that the same circuit can be used for both parity generation and checking.



(a) 3-bit even parity generator (b) 4-bit even parity checker

FIGURE 4-25
Logic diagram of a parity generator and checker

TABLE 4-5
Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

It is obvious from the foregoing example that parity-generation and checking circuits always have an output function that includes half of the minterms whose numerical values have either an odd or even number of 1's. As a consequence, they can be implemented with exclusive-OR gates. A function with an even number of 1's is the complement of an odd function. It is implemented with exclusive-OR gates except that the gate associated with the output must be an exclusive-NOR to provide the required complementation.

7-9 ERROR-CORRECTING CODE

The complexity level of a memory array may cause occasional errors in storing and retrieving the binary information. The reliability of a memory unit may be improved by employing error-detecting and correcting codes. The most common error-detection scheme is the parity bit. (See Section 4-9.) A parity bit is generated and stored along with the data word in memory. The parity of the word is checked after reading it from memory. The data word is accepted if the parity sense is correct. If the parity checked results in an inversion, an error is detected, but it cannot be corrected.

An error-correcting code generates multiple check bits that are stored with the data word in memory. Each check bit is a parity over a group of bits in the data word. When the word is read from memory, the associated parity bits are also read from memory and compared with a new set of check bits generated from the read data. If the check bits compare, it signifies that no errror has occurred. If the check bits do not compare with the stored parity, they generate a unique pattern, called a *syndrome,* that can be used to identify the bit in error. A single error occurs when a bit changes in value from 1 to 0 or from 0 to 1 during the write or read operation. If the specific bit in error is identified, then the error can be corrected by complementing the erroneous bit.

Hamming Code

One of the most common error-correcting codes used in random-access memories was devised by R. W. Hamming. In the Hamming code, k parity bits are added to an n-bit data word, forming a new word of $n + k$ bits. The bit positions are numbered in sequence from 1 to $n + k$. Those positions numbered as a power of 2 are reserved for the parity bits. The remaining bits are the data bits. The code can be used with words of any length. Before giving the general characteristics of the code, we will illustrate its operation with a data word of eight bits.

Consider, for example, the 8-bit data word 11000100. We include four parity bits with the 8-bit word and arrange the 12 bits as follows:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

The four parity bits, P_1, P_2, P_4, and P_8, are in positions 1, 2, 4, and 8, respectively. The eight bits of the data word are in the remaining positions. Each parity bit is calculated as follows:

P_1 = XOR of bits (3, 5, 7, 9, 11) = $1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$

P_2 = XOR of bits (3, 6, 7, 10, 11) = $1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

P_4 = XOR of bits (5, 6, 7, 12) = $1 \oplus 0 \oplus 0 \oplus 0 = 1$

P_8 = XOR of bits (9, 10, 11, 12) = $0 \oplus 1 \oplus 0 \oplus 0 = 1$

Remember that the exclusive-OR operation performs the odd function. It is equal to 1 for an odd number of 1's in the variables and to 0 for an even number of 1's. Thus, each parity bit is set so that the total number of 1's in the checked positions, including the parity bit, is always even.

The 8-bit data word is stored in memory together with the 4 parity bits as a 12-bit composite word. Substituting the four P bits in their proper positions, we obtain the 12-bit composite word stored in memory:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	1	1	1	0	0	1	0	1	0	0

When the 12 bits are read from memory, they are checked again for possible errors. The parity is checked over the same combination of bits including the parity bit. The four check bits are evaluated as follows:

C_1 = XOR of bits (1, 3, 5, 7, 9, 11)

C_2 = XOR of bits (2, 3, 6, 7, 10, 11)

C_4 = XOR of bits (4, 5, 6, 7, 12)

C_8 = XOR of bits (8, 9, 10, 11, 12)

A 0 check bit designates an even parity over the checked bits and a 1 designates an odd parity. Since the bits were stored with even parity, the result, $C = C_8C_4C_2C_1 = 0000$, indicates that no error has occurred. However, if $C \neq 0$, then the 4-bit binary number formed by the check bits gives the position of the erroneous bit. For example, consider the following three cases:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

In the first case, there is no error in the 12-bit word. In the second case, there is an error in bit position number 1 because it changed from 0 to 1. The third case shows an error in bit position 5 with a change from 1 to 0. Evaluating the XOR of the corresponding bits, we determine the four check bits to be as follows:

	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

Thus, for no error, we have $C = 0000$; with an error in bit 1, we obtain $C = 0001$; and with an error in bit 5, we get $C = 0101$. The binary number of C, when it is not equal to 0000, gives the position of the bit in error. The error can be corrected by complementing the corresponding bit. Note that an error can occur in the data word or in one of the parity bits.

The Hamming code can be used for data words of any length. In general, the Hamming code consists of k check bits and n data bits for a total of $n + k$ bits. The syndrome value C consists of k bits and has a range of 2^k values between 0 and $2^k - 1$. One of these values, usually zero, is used to indicate that no error was detected, leaving $2^k - 1$ values to indicate which of the $n + k$ bits was in error. Each of these $2^k - 1$ values can be used to uniquely describe a bit in error. Therefore, the range of k must be equal to or greater than $n + k$, giving the relationship

$$2^k - 1 \geq n + k$$

Solving for n in terms of k, we obtain

$$2^k - 1 - k \geq n$$

This relationship gives a formula for evaluating the number of data bits that can be used in conjunction with k check bits. For example, when $k = 3$, the number of data bits that can be used is $n \leq (2^3 - 1 - 3) = 4$. For $k = 4$, we have $2^4 - 1 - 4 = 11$, giving $n \leq 11$. The data word may be less than 11 bits, but must have at least 5 bits, otherwise, only 3 check bits will be needed. This justifies the use of 4 check bits for the 8 data bits in the previous example. Ranges of n for various values of k are listed in Table 7-8.

The grouping of bits for parity generation and checking can be determined from a list of the binary numbers from 0 through $2^k - 1$. (Table 1-1 gives such a list.) The least significant bit is a 1 in the binary numbers 1, 3, 5, 7, and so on. The second significant bit is a 1 in the binary numbers 2, 3, 6, 7, and so on. Comparing these numbers with the bit positions used in generating and checking parity bits in the Hamming code, we note the relationship between the bit groupings in the code and the position of the 1 bits in the binary count sequence. Note that each group of bits starts with a number that is a power of 2 such as 1, 2, 4, 8, 16, etc. These numbers are also the position numbers for the parity bits.

TABLE 7-8
Range of Data Bits for *k* Check Bits

Number of Check Bits, *k*	Range of Data Bits, *n*
3	2–4
4	5–11
5	12–26
6	27–57
7	58–120

Single-Error Correction, Double-Error Detection

The Hamming code can detect and correct only a single error. Multiple errors are not detected. By adding another parity bit to the coded word, the Hamming code can be used to correct a single error and detect double errors. If we include this additional parity bit, then the previous 12-bit coded word becomes $001110010100P_{13}$, where P_{13} is evaluated from the exclusive-OR of the other 12 bits. This produces the 13-bit word 0011100101001 (even parity). When the 13-bit word is read from memory, the check bits are evaluated and also the parity P over the entire 13 bits. If $P = 0$, the parity is correct (even parity), but if $P = 1$, then the parity over the 13 bits is incorrect (odd parity). The following four cases can occur:

If $C = 0$ and $P = 0$	No error occurred
If $C \neq 0$ and $P = 1$	A single error occurred, which can be corrected
If $C \neq 0$ and $P = 0$	A double error occurred, which is detected but cannot be corrected
If $C = 0$ and $P = 1$	An error occurred in the P_{13} bit

Note that this scheme cannot detect more than two errors.

Integrated circuits that use a modified Hamming code to generate and check parity bits for a single-error correction, double-error detection scheme are available commercially. One that uses an 8-bit data word and a 5-bit check word is IC type 74637. Other integrated circuits are available for data words of 16 and 32 bits. These circuits can be used in conjunction with a memory unit to correct a single error or detect double errors during the write and read operations.