Error Detection and Correction

Sparsh Mittal

Error detection

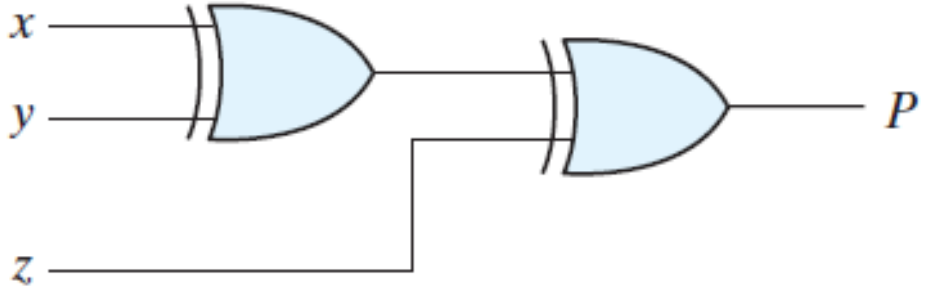
Parity bit: the most common error detection scheme.

A parity bit is generated and stored along with the data word in memory.

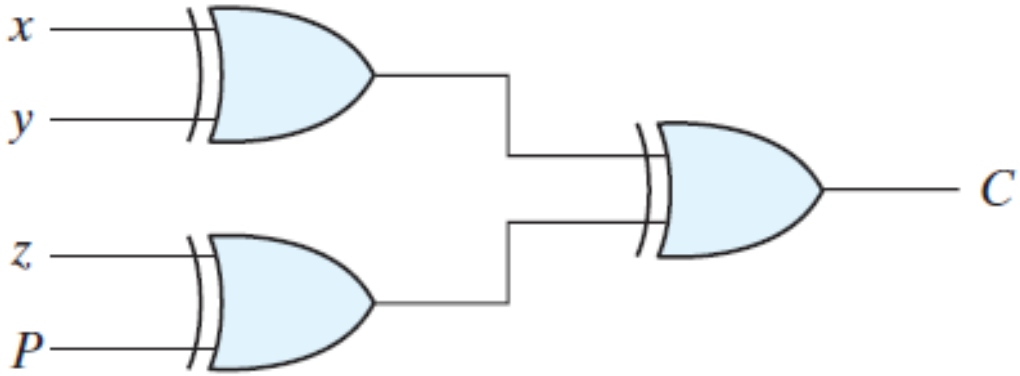
The parity of the word is checked after reading it from memory.

The data word is accepted if the parity of the bits read out is correct.

If the parity checked results in an inversion, an error is detected, but it cannot be corrected.



(a) 3-bit even parity generator



(b) 4-bit even parity checker

Error correction

An error-correcting code generates multiple parity 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 back from memory, the associated parity bits are also read from memory and compared with a new set of check bits generated from the data that have been read.

If the check bits are correct, no error has occurred.

Error correction

If the check bits do not match the stored parity, they generate a unique pattern, called a syndrome, that can be used to identify the bit that is 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 distance

The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different

100→011 has distance 3

010→111 has distance 2

Hamming Code

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.

Example for n=8

Consider, for example, the 8-bit data word 11000100.

We include 4 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 4 parity bits, $P1$, $P2$, $P4$, and $P8$, are in positions 1, 2, 4, and 8, respectively.

The 8 bits of the data word are in the remaining positions.

Calculation of parity bits

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

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

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

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

XOR 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.

Final word

The 8-bit data word is stored in memory together with the 4 parity bits as a 12-bit composite word.

Substituting the 4 *P* bits in their proper positions, we obtain the 12-bit composite word stored in memory:

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

At the time of reading the word

When the 12 bits are read from memory, they are checked again for errors.

The parity is checked over the same combination of bits, including the parity bit.

The 4 check bits are evaluated as follows:

$$C_1 = \text{XOR of bits } (1, 3, 5, 7, 9, 11)$$

$$C_2 = \text{XOR of bits } (2, 3, 6, 7, 10, 11)$$

$$C_4 = \text{XOR of bits } (4, 5, 6, 7, 12)$$

$$C_8 = \text{XOR of bits } (8, 9, 10, 11, 12)$$

Scenarios

A 0 check bit designates even parity over the checked bits and a 1 designates odd parity.

Since the bits were stored with even parity, the result, $C = C8C4C2C1 = 0000$, indicates that no error has occurred. However, if C is not zero 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

Lets compute parity checks

	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 $C = 0101$.

When the binary number C is not equal to 0000, it 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.

Generalizing the ideas

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 $n + k$,

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

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

Table 7.2
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

Grouping of bits

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.

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.

Interesting observation

Each group of bits starts with a number that is a power of 2: 1, 2, 4, 8, 16, etc.

These numbers are also the position numbers for the parity bits.

Single-Error Correction, Double-Error Detection

The Hamming code can detect and correct only a single error.

By adding another parity bit to the coded word, 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, as is 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 arise:

Condition	Conclusion
$C_8C_4C_2C_1 = 0$ and $C_{parity} = 0$	No error has occurred
$C_8C_4C_2C_1 \neq 0$ and $C_{parity} = 1$	A single error has occurred that can be corrected
$C_8C_4C_2C_1 \neq 0$ and $C_{parity} = 0$	A double error occurred that is detected, but cannot be corrected.
$C_8C_4C_2C_1 = 0$ and $C_{parity} = 1$	An error occurred in the parity bit (P13 for n+k=12)

This scheme may detect more than two errors, but is not guaranteed to detect all such errors.

Procedure for computing parity/check bits for SECDED

Sender side

- Place n data-bits. Compute k parity bits P1, P2, P4, P8, etc.
- Now, we have n+k bits.
- Take XOR of n+k bits to compute the parity bit.
- Now, we have n+k+1 bits

Receiver side

- Consider first n+k bits.
- Compute k check bits C1, C2, C4, C8, etc. Compute $C=C_8C_4C_2C_1$.
- Compute parity check bit (C_{parity}) as XOR of all n+k+1 bits.
- From $C_8C_4C_2C_1$ and C_{parity}, decide about errors, using 4 cases shown earlier