# 6 CODIFICACION EN PALABRAS BINARIAS DETECCIÓN DE ERROR



***Los sistemas digitales procesan la información codificada en palabras binarias, constituidas por conjuntos ordenados de ceros y unos. Esta información puede ser cuantitativa (es decir, una cantidad expresada por un número) o cualitativa (referida a la distinción entre diversas posibilidades o cualidades).***

***La longitud de palabra determina su capacidad para representar informaciones diferentes: una palabra de p dígitos es capaz de expresar $2^p$ posibilidades, las cuales estarán numeradas desde la 00...000 hasta la 11...111.***

***Ahora bien, el manejo de palabras binarias largas presenta una cierta complejidad (su lectura y escritura es dificultosa); tal complejidad puede reducirse «compactando» los dígitos de 4 en 4: sistema hexadecimal (base $16 = 2^4$).***

***La forma más inmediata de codificar la información en binario consiste en numerar en base 2 (código binario directo), pero existen otras posibilidades de interés. Una de ellas respeta la numeración decimal (base 10), que es la habitual para el hombre, y traslada a binario cada una de las cifras por separado: código BCD. También es útil el código Gray, en el cual dos palabras consecutivas difieren en un solo dígito; en ello se basa la simplificación mediante mapas de Karnaugh y la alta seguridad funcional que la codificación Gray aporta en las transiciones entre estados o en las secuencias de vectores.***

***Una cuestión importante respecto a la transferencia de información es la fiabilidad o ausencia de error en la misma. En la transmisión o almacenamiento de la información pueden producirse errores que afecten a uno o varios bits de algunas palabras binarias; interesa disponer de mecanismos que faciliten la detección de la existencia de errores y, si fuera posible, permitan corregirlos: tales mecanismos se basan en añadir dígitos adicionales (redundancia) para efectuar comprobaciones sobre la información recibida.***

***La paridad es el más sencillo de los detectores de error, basado en indicar si el número de «unos» de cada palabra es par o impar. La paridad es un código de distancia 2: la distancia (número de dígitos diferentes) entre dos palabras de este código es siempre par y, por ello, detecta errores en número impar.***

***El concepto de distancia mínima (número de bits en que difieren dos palabras de un código) es básico en el desarrollo de códigos detectores de error. Hamming introdujo un método simple para construir códigos de distancia mínima 4 (basados en la utilización de paridades parciales) que permiten detectar hasta tres errores y efectuar corrección cuando hay uno solo de ellos.***


Tomás Pollán Santamaría. Tecnología Electrónica. E.U.I.T.I.Z. Universidad de Zaragoza. tpollan@posta.unizar.es




## *6.1. La información codificada en palabras binarias*

Cada dígito de una palabra binaria recibe el nombre de bit (*bit = binary digit*) y puede tener dos valores: **0** y **1**. El número de dígitos o bits que forman una palabra binaria determina su longitud **p**.

La información contenida en una palabra digital puede ser de tipo cuantitativo (numérico) o de tipo cualitativo (distinción entre varias situaciones, posibilidades o cualidades). Por ejemplo, el resultado de la medida de una magnitud física con valor 185 conducirá a la palabra binaria **10111001**, mientras que el estado civil de una persona puede ser codificado con palabras de 2 bits (**00** soltera, **01** casada, **10** divorciada, **11** viuda) y los siete colores del arco iris necesitan para su codificación palabras binarias de 3 dígitos.

Los vectores de entrada y de salida de un sistema digital son palabras binarias que expresan, respectivamente, a través de la correspondiente codificación, la información que recibe el sistema y la información resultante del procesado que el sistema efectúa sobre ella. Muchas veces tales vectores se subdividen en varias palabras digitales con significado propio cada una de ellas.

Internamente los sistemas digitales generan condiciones o informaciones intermedias expresadas también en palabras binarias de uno o varios bits. En particular, los sistemas secuenciales incorporan memoria de su evolución anterior en forma de «estado del sistema», expresado en palabras binarias cuyos **n** bits corresponden a las variables de estado.

A la hora de codificar diversas informaciones, la posibilidad más inmediata consiste en numerarlas en sistema binario: *codificación binaria directa*. Pero también existen otras posibilidades de interés, como pueden ser los códigos BCD y Gray.

La codificación BCD tiene sentido, principalmente, para representar números (información cuantitativa) y con tal finalidad su explicación se encuentra detallada en el capítulo 3 (apartado 3.4.). El código Gray (que ya ha sido utilizado en la numeración de los mapas de Karnaugh, apartado 2.2.) será considerado, con mayor detalle, un poco más adelante en este mismo apartado.

### *6.1.1. Longitud de palabra y capacidad de información*

En principio, las palabras digitales son de longitud variable según la información que vayan a representar. Una palabra binaria de longitud **p** está formada por **p** dígitos $\mathbf{b_{p-1}\ b_{p-2}\ b_{p-3}\ ...\ b_3\ b_2\ b_1\ b_0}$ y es capaz de representar o expresar $\mathbf{2^p}$ posibilidades.

Al representar los dígitos de una palabra con subíndices $\mathbf{b_i}$, se comienza numerando el dígito menos significativo con 0 ya que dicho dígito (caso de que la palabra sea un número binario) corresponde a las unidades y su valor relativo es $2^0 = 1$; de esta forma el valor relativo del dígito i-ésimo $\mathbf{b_i}$ es $\mathbf{2^i}$.

Muchos bloques y sistemas digitales adoptan una longitud fija para las palabras que procesan; en tal caso, si la palabra que se desea procesar es de menor número de bits se añade el correspondiente número de ceros para completar su longitud, mientras que cuando el número o información a procesar desborda la longitud de palabra fijada se utilizan varias palabras sucesivas.

Las longitudes de palabra más comunes son las siguientes:

- 4 bits: 16 posibilidades
- 1 *byte* = 8 bits: 256 posibilidades
- 16 bits: 65.536 posibilidades (64K)
- 32 bits: 4.294.967.296 posibilidades (aprox. 4.300 millones)
- 64 bits: aprox. 16 x $10^{18}$ posibilidades (dieciséis trillones)

destacando entre ellas la longitud de 8 bits, que se conoce con el nombre de *byte* y es utilizada como longitud de referencia cuando no se indica otra cosa.

Así, por ejemplo, la capacidad global de una memoria suele expresarse en número de registros de 8 bits (en *bytes*, aun en los casos en que el procesador del sistema utiliza palabras de 16 o de 32 bits): al indicar una memoria de 1 Mega nos referimos a una capacidad de almacenamiento de $2^{20} \approx 10^6$ *bytes*.

**$2^{20}$** bytes = **1.048.576** bytes = 1.048.576 x 8 bits = 8.388.608 bits $\approx$ 8 x $10^6$ bits.

### *6.1.2. Compactación hexadecimal*

El manejo de palabras binarias por parte del hombre, es decir, su lectura o escritura presenta cierta complejidad, habida cuenta del amplio número de dígitos que las palabras binarias tienen por lo general y de la dificultad que supone citar o visualizar sin errores el correspondiente número de ceros y unos. Por ejemplo, al referirnos a una cantidad no muy grande como puede ser la de 234 unidades tenemos que indicar **11101010** (*uno, uno, uno, cero, uno, cero, uno, cero*; o bien, si leemos este número binario como si fuera decimal, *once millones ciento un mil diez*).

El sistema de numeración hexadecimal, cuya base 16 es la cuarta potencia de 2, permite reducir en gran medida dicha complejidad al compactar de 4 en 4 los dígitos de las palabras binarias. El sistema hexadecimal utiliza 16 signos, del **0** al **F** (15), cuya equivalencia binaria y decimal es la siguiente:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0000** | 0 | **4** | **0100** | 4 | **8** | **1000** | 8 | **C** | **1100** | 12 |
| **1** | **0001** | 1 | **5** | **0101** | 5 | **9** | **1001** | 9 | **D** | **1101** | 13 |
| **2** | **0010** | 2 | **6** | **0110** | 6 | **A** | **1010** | 10 | **E** | **1110** | 14 |
| **3** | **0011** | 3 | **7** | **0111** | 7 | **B** | **1011** | 11 | **F** | **1111** | 15 |

*Ejemplos de palabras en binario y en **hexadecimal**:*

*10010101 = **95*** *10001100 = **8C*** *11110000 = **F0*** *101011 = **2B***

***72** = 1110010* ***D1** = 11010001* ***AB** = 10101011* ***8E** = 10001110*



Una memoria de 16 líneas de direccionamiento tiene una capacidad de 64K registros (65.536) y utilizará para numerarlos los códigos hexadecimales que van del **0000** al **FFFF**:

- el registro ***2A3B*** será el que hace el número de orden decimal *10.811* (comenzando por el número 0) y se seleccionará con los valores binarios **0010101000111011** en el bus de direcciones;
- el registro que hace el número *50.000* de ellos tendrá por referencia hexadecimal ***C350*** y se seleccionará con los valores **1100001101010000** en el bus de direcciones;
- al activar dicho bus con los valores **0100111110011100** el registro seleccionado será el ***4F9C***, que hace el número *20.380* de ellos.

Un bus de direcciones de 24 líneas permite manejar $2^{24}$ = 16.777.216 $\approx$ 16 x $10^6$ registros (16 Megas), cuya numeración requiere 6 dígitos hexadecimales.

Una palabra de 1 *byte* da lugar a 2 dígitos hexadecimales, si es de 16 bits se expresa en 4 dígitos hexadecimales y si es de 32 bits necesita 8 dígitos hexadecimales.

Existen otras formas de compactar las palabras binarias, por ejemplo el sistema de numeración octal, con base 8, que permite agrupar los bits de 3 en 3, pero son mucho menos utilizadas que el sistema hexadecimal.

### *6.1.3. Codificación Gray*

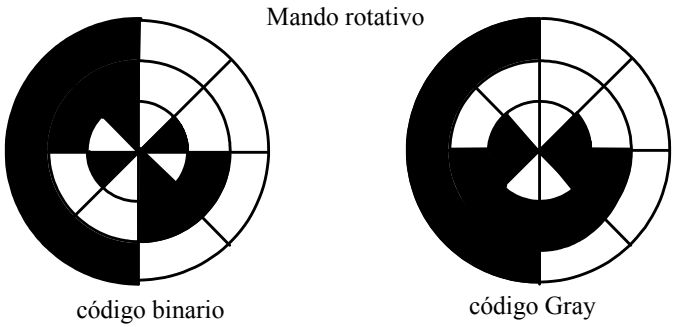
El código Gray se caracteriza por la propiedad de que dos palabras consecutivas del mismo difieren en un solo bit.

Esta propiedad implica que los términos mínimos de vectores de entrada sucesivos son simplificables entre sí (ya que difieren en una sola variable); por ello, la numeración Gray es la base de la simplificación por mapas de Karnaugh. [Con tal finalidad el código Gray fue presentado al tratar dichos mapas en 2.2.]

Pero, además, la codificación Gray proporciona alta seguridad funcional en las transiciones entre palabras binarias: al diferenciarse en un solo bit, el paso de una palabra a la siguiente no puede producir errores o espurios debidos a las diferencias de tiempo en la transición o propagación de cada bit. En tal sentido, el código Gray resulta muy útil en la codificación de estados (es ideal para los grafos de estado), o en la codificación de condiciones o vectores cuando éstos evolucionan siguiendo una secuencia fija.

Por ejemplo, en un mando rotativo con 8 posiciones o niveles el código Gray evita los errores que pueden producirse en las fronteras entre dos posiciones:

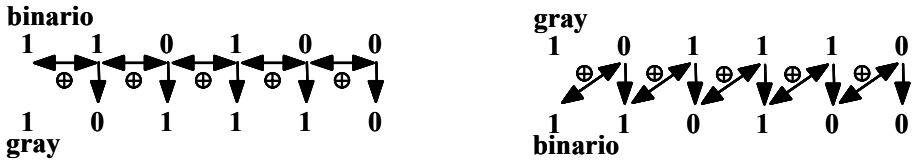
- si la codificación es binaria directa, al pasar del nivel 3 **011** al nivel 4 **100** podría suceder que en la frontera entre ambos apareciese el nivel 0 **000**, por anularse los dos bits menos significativos antes de activarse el otro bit, o bien que se formase el nivel 7 **111** si dicho bit se activa antes que se anulen los dos primeros;
- en código Gray el paso del nivel 3 **010** al nivel 4 **110** no puede producir ningún otro nivel, distinto de ambos, ya que únicamente se modifica el bit más significativo.




La formación sucesiva de las palabras del código Gray, por el método especular, ha sido descrita en el capítulo 2 (apartado 2.2.).

El cambio de código de binario normal a código Gray viene dado por una simple operación "**o-exclusiva**" (sea **$b_i$** la cifra i-ésima en código binario y **$g_i$** la cifra i-ésima en código Gray) : $g_i$ (gray) = $b_{i+1} \oplus b_i$ (binario); es decir, basta hacer la operación "**o-exclusiva**" entre el bit correspondiente ( i ) del código binario y el anterior ( i+1 ) del mismo código.

El cambio de código inverso, de código Gray a binario, es análogo: $b_i$ (binario) = $b_{i+1} \oplus g_i$ ; pero en este caso, es preciso hacer la operación "**o-exclusiva**" entre el bit anterior ( i+1 ) del mismo código binario y el bit correspondiente ( i ) del código Gray.




*Ejemplos:* ***10101101***$_{(2}$ = $11111011_{Gray}$ ***11100100***$_{(2}$ = $10010110_{Gray}$

$10011011_{Gray}$ = ***11101101***$_{(2}$ $11101011_{Gray}$ = ***10110010***$_{(2}$



### *6.1.4. Codificación de texto*

El texto que configura un libro, folleto, carta, documento o cualquier otro tipo de escrito está compuesto por una sucesión ordenada de caracteres alfabéticos y numéricos (alfanuméricos), junto con algunos signos de puntuación, espacios en blanco, separación entre párrafos, tabulaciones, etc.

Nuestro alfabeto utiliza 26 caracteres alfabéticos (25 letras simples más la w) en dos formas, mayúsculas y minúsculas, a los cuales hay que añadir 10 cifras decimales, varios signos de puntuación, algunos signos matemáticos (+, -, =, >, < , ...) y otros caracteres especiales. Un teclado mecanográfico de tipo normal presenta cerca de medio centenar de teclas, con dos posibilidades cada una; en total, unos 100 caracteres. Para su codificación bastarán palabras de 7 bits y sobrará una veintena larga de palabras que se utilizarán para caracteres de control (fin de párrafo, fin de página, salto de línea, tabulaciones, ...).

El código más utilizado para la codificación de textos es el **ASCII** (*American Standard Code for Information Interchange*) que utiliza palabras de 7 bits conforme a la tabla siguiente (existe, también, un código ASCII ampliado que utiliza palabras de 8 bits):

| **Hexadecimal** $b_6b_5b_4$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $b_3b_2b_1b_0$ | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0** | NUL | DLE | SP | 0 | @ | P | ` | p |
| **1** | SOH | DC1 | ! | 1 | A | Q | a | q |
| **2** | STX | DC2 | " | 2 | B | R | b | r |
| **3** | ETX | DC3 | # | 3 | C | S | c | s |
| **4** | EOT | DC4 | $ | 4 | D | T | d | t |
| **5** | ENQ | NAK | % | 5 | E | U | e | u |
| **6** | ACK | SYN | & | 6 | F | V | f | v |
| **7** | BEL | ETB | ´ | 7 | G | W | g | w |
| **8** | BS | CAN | ( | 8 | H | X | h | x |
| **9** | HT | EM | ) | 9 | I | Y | i | y |
| **A** | LF | SUB | * | : | J | Z | j | z |
| **B** | VT | ESC | + | ; | K | [ | k | { |
| **C** | FF | FS | ' | < | L | \ | l | \| |
| **D** | CR | GS | - | = | M | ] | m | } |
| **E** | SO | RS | . | > | N | ^ | n | ~ |
| **F** | SI | US | / | ? | O | _ | o | DEL |

Las 10 cifras decimales ocupan los códigos del **30** al **39** mientras que las letras van del **41** (A) al **5A** (Z), las mayúsculas, y del **61** (a) al **7A** (z), las minúsculas. El resto de los códigos se refiere a signos de puntuación y otros caracteres diversos.

Los 32 códigos iniciales, del **00** al **1F**, se destinan a caracteres de control:

| | | | | | |
|---|---|---|---|---|---|
| **00** | NUL | Nulo | **10** | DLE | Anulación de lo transmitido |
| **01** | SOH | Inicio de encabezamiento | **11** | DC1 | Control dispositivo 1 |
| **02** | STX | Inicio de texto | **12** | DC2 | Control dispositivo 2 |
| **03** | ETX | Final de texto | **13** | DC3 | Control dispositivo 3 |
| **04** | EOT | Final de transmisión | **14** | DC4 | Control dispositivo 4 |
| **05** | ENQ | Pregunta | **15** | NAK | Acuse de recibo negativo |
| **06** | ACK | Acuse de recibo | **16** | SYN | Sincronizador |
| **07** | BEL | Aviso acústico | **17** | ETB | Final del bloque transmitido |
| **08** | BS | Retroceder un espacio | **18** | CAN | Anulación |
| **09** | HT | Tabulador horizontal | **19** | EM | Fin del medio o soporte |
| **0A** | LF | Nueva línea | **1A** | SUB | Substituir |
| **0B** | VT | Tabulador vertical | **1B** | ESC | Escape (anulación de orden) |
| **0C** | FF | Nueva página | **1C** | FS | Separador de archivos |
| **0D** | CR | Retorno del carro | **1D** | GS | Separador de grupo |
| **0E** | SO | Fuera de código | **1E** | RS | Separador de registros |
| **0F** | SI | Retorno al código | **1F** | US | Separador de unidad |
| **10** | SP | Espacio (código **10**) | **7F** | DEL | Borrado (código **7F**) |

Dado que el código **ASCII** es de 7 bits y, en cambio, suelen utilizarse palabras normalizadas a 8 bits (1 *byte*), el octavo dígito se aprovecha para la detección de errores como bit de paridad (véase el apartado siguiente).

El código **ASCII** con paridad utiliza palabras binarias de 8 dígitos, de los cuales el más significativo (el que inicia la palabra) es el bit de paridad; la paridad de la palabra completa es siempre 0. Este código goza de amplia aceptación siendo el utilizado habitualmente en el procesado de textos, en la transmisión de la información, en la comunicación con periféricos (impresoras, monitores,...), etc.

### *6.2. La paridad para detectar error*

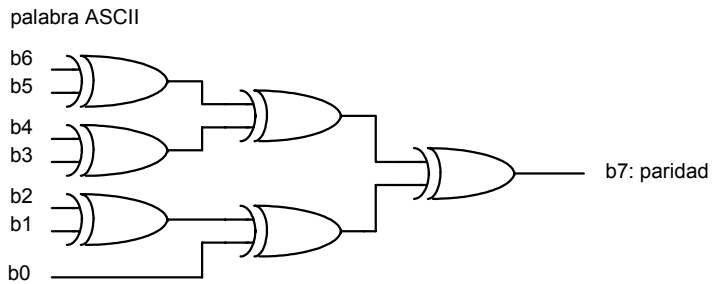
La paridad constituye una forma muy simple de detectar errores basada en contabilizar el número de «unos» que cada palabra binaria contiene; se dice que una palabra es par (paridad = **0**) cuando el número de «unos» que contiene es par y será impar (paridad = **1**) cuando lo sea el número de «unos».

Al añadir a una palabra binaria su paridad se forma una palabra «ampliada» que es siempre par: si la palabra era impar, al añadirle un **1** resulta par; si era par, se le añade un **0** y su paridad no cambia. Caso de recibir una palabra «ampliada» que sea impar, ello supone que, al menos, uno de los dígitos de dicha palabra es erróneo

De esta manera, la paridad detecta error en una palabra binaria cuando el número de bits erróneos de la misma es impar; en cambio, si en una palabra se modifican (se invierten) un número par de bits la paridad de la palabra no se altera y el error no será detectado.



La paridad de una palabra se calcula circuitalmente mediante puertas "**o-exclusiva**" en estructura arborescente:




Cada puerta "**o-exclusiva**" calcula la paridad de sus dos entradas, es decir, genera un **1** al recibir un número impar de «unos» en las mismas. De esta forma, las primeras puertas calculan la paridad de un par de dígitos de la palabra y las puertas siguientes van agrupando las paridades de dos en dos, hasta completar el cálculo de la paridad de la palabra completa.

La paridad es un código de *distancia mínima* 2: al ampliar las palabras binarias añadiendo a las mismas su paridad, las palabras del código resultante son todas ellas de paridad par y se diferencian, al menos, en 2 bits, pues la modificación de un solo bit produce una palabra impar que no pertenece a dicho código.

Un código de distancia mínima 2 permite detectar la modificación o error de un dígito: al invertirse un bit la palabra resultante tiene una distancia 1 respecto a la palabra correcta y no pertenecerá al código (será una palabra errónea), lo cual indica la existencia de error.

En el caso de la paridad, la distancia entre cualesquiera dos palabras «ampliadas» (con su bit de paridad) es siempre múltiplo de 2, pues la paridad de una palabra «ampliada» es siempre par; lo cual permite detectar la existencia de error cuando afecta a un número impar de bits (1, 3, 5, ..) pero, en cambio, la paridad no detecta error cuando el número de bits erróneos es par (2, 4, …).

Otra forma de aplicar la paridad a conjuntos de **n** palabras binarias consiste en añadir una palabra más, cuyos dígitos correspondan a la paridad global de los bits que ocupan la misma posición en las **n** palabras anteriores; es decir, el bit i-ésimo de la nueva palabra es la paridad de los **n** bits i-ésimos de las palabras anteriores.

Esta forma de detección de errores es complementaria de la paridad de las palabras individuales, ya que realiza una detección «en vertical» sobre las columnas de bits (supuestas las **n** palabras colocadas en columna, una debajo de otra), mientras que la paridad de cada palabra realiza una detección «en horizontal» sobre la fila de bits que forma la palabra.

En relación con estos dos tipos de paridad (*vertical* y *horizontal*) las **n** palabras de longitud **p** se organizan matricialmente, formando una matriz de dimensión **n** x **p**, y se calculan las paridades de las filas (*paridad horizontal*) y de las columnas (*paridad vertical*) de la matriz.

Las dos palabras resultantes (*vertical* y *horizontal*) tienen idéntica paridad, ya que corresponde a la paridad global de las **n** palabras en bloque; si al conjunto de las **n** palabras se añaden las citadas paridades (la horizontal como columna y la vertical como fila) y se agrega dicho bit de paridad global, se configura una nueva matriz ampliada de dimensiones **n+1** x **p+1**, cuyas paridades en ambos sentidos (horizontal y vertical) son todas nulas (matriz de paridad **0**).

*Ejemplo: Conjunto de 12 palabras, con su paridad vertical y horizontal.*

| | paridad horizontal ↓ | | *Presencia de un error único:* | | |
|---|---|---|---|---|---|
| | 1 1 0 1 1 0 1 1 | **0** | 1 1 0 1 1 0 1 1 | **0** | ***0*** |
| | 1 0 0 0 1 0 1 1 | **0** | 1 0 0 0 1 0 1 1 | **0** | ***0*** |
| | 0 1 1 0 0 0 0 1 | **1** | 0 1 1 0 0 0 0 1 | **1** | ***0*** |
| | 1 1 1 0 0 1 1 1 | **0** | 1 1 1 0 **1** 1 1 1 | **0** | ***1*** ← |
| | 1 0 1 1 0 1 0 1 | **1** | 1 0 1 1 0 1 0 1 | **1** | ***0*** |
| | 1 0 0 0 0 1 0 1 | **1** | 1 0 0 0 0 1 0 1 | **1** | ***0*** |
| | 0 0 1 1 0 1 1 0 | **0** | 0 0 1 1 0 1 1 0 | **0** | ***0*** |
| | 0 1 1 1 1 0 1 0 | **1** | 0 1 1 1 1 0 1 0 | **1** | ***0*** |
| | 1 0 1 1 0 1 1 0 | **1** | 1 0 1 1 0 1 1 0 | **1** | ***0*** |
| | 0 1 0 0 0 1 0 1 | **1** | 0 1 0 0 0 1 0 1 | **1** | ***0*** |
| | 1 1 1 0 0 0 1 1 | **1** | 1 1 1 0 0 0 1 1 | **1** | ***0*** |
| paridad | 0 1 1 1 1 0 0 0 | **0** | 0 1 1 1 1 0 0 0 | **0** | ***0*** |
| vertical → | **1 1 0 0 0 0 1 0** | **1** | **1 1 0 0 0 0 1 0** | **1** | ***0*** |
| | | | ↑ | | |
| nuevas paridades del conjunto «ampliado»: → | | | ***0 0 0 0 1 0 0 0*** | | ***0*** |

Esta combinación de paridades presenta una probabilidad muy alta de detectar la existencia de error, pues solamente falla en los casos en que la situación de todos los bits erróneos corresponda a filas y a columnas que contengan ambas un número par de errores.

Además, en caso de que se detecte error en una sola fila y una sola columna, dicho bit puede corregirse (bajo el supuesto, muy razonable, de que no existen en la misma fila y columna otros errores que, a su vez, estén «compensados» en el resto de filas y columnas). [Véase en el ejemplo anterior la presencia de un error y cómo queda indicado al calcular las nuevas paridades vertical y horizontal del conjunto.] También pueden corregirse varios errores en caso de que todos ellos se encuentren en la misma fila o en la misma columna y que el número de errores sea impar.



La *verificación de suma* (*checksum*) es un tipo de detección análogo a la paridad vertical: utiliza como palabra adicional el resultado de la suma de las **n** palabras anteriores ejecutada sobre **p** bits (sin tener en cuenta los arrastres superiores), siendo **p** la longitud de palabra de todas ellas; es decir, la suma de las **n** palabras en módulo $2^p$ (lo cual asegura que la longitud de palabra del resultado es también **p**). De esta forma se detecta la presencia de errores siempre que no se encuentren «compensados» en la misma columna, en el sentido de que el número de bits que han cambiado de **0** a **1** es el mismo que los bits que han efectuado el cambio opuesto de **1** a **0**.

### *6.3. Códigos detectores y correctores de error*

En la transmisión y en el almacenamiento (conservación en memoria, sea ésta de tipo RAM o sea en disco, cinta o cualquier otro soporte físico de la misma) de las palabras binarias pueden producirse errores que modifiquen el valor booleano de uno o de varios bits. La palabra o palabras correspondientes expresarán una información errónea.

Un código es capaz de detectar errores, es decir, de discriminar palabras con información errónea, cuando la modificación de uno o varios bits de una palabra del código da lugar a una palabra binaria que no pertenece al mismo. El concepto de distancia permite analizar y generalizar la forma de operar de los códigos detectores de error.

La *distancia de Hamming* entre dos palabras binarias de la misma longitud es el número de dígitos en que dichas palabras se diferencian. Se dice que un código es de *distinta mínima* **D** cuando dos palabras del mismo difieren, al menos, en el valor de **D** de sus bits. Tal código es capaz de detectar cualquier error que afecte a **D-1** dígitos o menos, ya que la modificación en una palabra del código de un número de bits inferior a **D** da lugar a una palabra que no pertenece al código. Para detectar la modificación o error que afecte a **n** bits se requiere un código cuya distancia mínima sea **n+1** o superior.

La paridad es un código de distancia par (múltiplo de 2): las palabras permisibles distan entre sí un número par de dígitos. Cualquier modificación que afecte a un número impar de bits da lugar a una palabra no permitida y, por tanto, reconocida como errónea; en cambio, la paridad no permite detectar error cuando el número de bits afectados es par.

Hamming desarrolló métodos sistemáticos, basados en la misma idea que la paridad, para construir códigos de distancia mínima 3 y 4, con la particularidad de que dichos códigos permiten corregir el error cuando éste afecta a un solo bit.

El código Hamming de distancia mínima 3 utiliza paridades «parciales» referidas a subconjuntos de dígitos de la palabra inicial. El código Hamming de distancia mínima 4 es análogo, añadiendo simplemente un bit de paridad global; ese bit suplementario aporta mayor fiabilidad respecto a la corrección de error.

El código Hamming de distancia mínima 3 permite detectar y corregir errores relativos a un solo bit, pero no es capaz de diferenciar los que afectan a un bit de los que afectan a un número par de ellos; de manera que, en el caso de que exista error en dos bits y se efectúe la corrección como si fuese en uno solo de ellos, la palabra corregida tiene más errores que la original.

El código Hamming de distancia mínima 4 (que supone simplemente añadir un bit de paridad global al de distancia mínima 3) informa si el error es en número par o impar de bits, evitando correcciones erróneas; es cierto que no permite diferenciar si el error afecta a 1 o a 3 (o más) bits, pero la probabilidad de que afecte a 1 ó 2 dígitos es muy superior a la de tener 3, 5, 7,… errores.

El método Hamming para formar un código de distancia mínima 4 se describe en las dos páginas siguientes: en la página de la izquierda (pág. 150) se explica la manera de actuar y en la de la derecha (pág. 151) se detalla su aplicación a un caso concreto: una palabra inicial de 12 dígitos.

De igual manera en las siguientes dos páginas se describe la forma de detectar y corregir errores en código Hamming de distancia mínima 4: a la izquierda (pág. 152) se explica la manera de proceder y a la derecha (pág. 153) se aplica al caso de un bit erróneo en la palabra obtenida anteriormente (en las páginas 150 y 151).

La generalización de este método para aplicarlo a palabras de diferente longitud es directa.



### *6.3.1. Construcción del código Hamming*

*El procedimiento de construcción del código se explica en relación con las sucesivas columnas de la tabla representada en la página siguiente.*

• Sea una palabra inicial de 12 bits: **$b_{11}$ $b_{10}$ $b_9$ $b_8$ $b_7$ $b_6$ $b_5$ $b_4$ $b_3$ $b_2$ $b_1$ $b_0$**.

• Se trata de construir una palabra ampliada por un conjunto de paridades parciales, que se entremezclan con los bits de la palabra inicial; para ello, es necesario numerar en binario los bits de la palabra ampliada [columna (1)]. Para facilitar la descripción se utiliza la denominación |número| para indicar el que numera los bits de la palabra ampliada.

*Habida cuenta de que las paridades parciales se refieren a las posiciones de los «unos» en el |número| de cada bit, en la columna (2) se han sustituido los «ceros» por – y los «unos» por símbolos diversos según su posición ( @ , * , Δ , # , & ) a los que denominaremos grafismos. [Esta representación gráfica es absolutamente superflua una vez comprendido el método.]*

• El primer bit de la palabra ampliada (numerado con 00…) se reserva para la paridad global y es el último que se calcula. Los bits de la palabra ampliada que solamente tienen un «uno» en su |número| (es decir, los que tienen un solo grafismo) se reservan para las paridades parciales [columna (3)].

• En el resto de los bits se colocan ordenadamente los dígitos de la palabra inicial (de menor a mayor valor significativo, tal como están ordenados sus |números|) [columnas (4) y (5)]. Si es preciso se continúa la numeración de bits, hasta que quepan todos los de la palabra inicial, reservando siempre los bits cuyo |número| contiene un solo «uno» para paridades parciales.

• Cada paridad parcial corresponde a un |número| con un solo «uno» y se calcula sobre los bits cuyo |número| contiene un «uno» en la misma posición; es decir, estarán reservados para paridad parcial los bits con un solo grafismo y para calcular una de ellas tomaremos los bits señalados con el mismo grafismo y hallaremos la paridad del conjunto de ellos.

La paridad parcial **P1** es la de los bits cuyo |número| acaba por «uno», es decir, aquellos cuyo grafismo es **&**: **$b_{11}$ $b_{10}$ $b_8$ $b_6$ $b_4$ $b_3$ $b_1$ $b_0$**. La paridad parcial **P2** es la de los bits cuyo |número| tiene un «uno» en penúltima posición, o sea cuyo grafismo es **#**: **$b_{10}$ $b_9$ $b_6$ $b_5$ $b_3$ $b_2$ $b_0$**. La paridad **P3** corresponde a un «uno» en antepenúltima posición y su grafismo es Δ: **$b_{10}$ $b_9$ $b_8$ $b_7$ $b_3$ $b_2$ $b_1$**; **P4** se refiere al grafismo *: **$b_{10}$ $b_9$ $b_8$ $b_7$ $b_6$ $b_5$ $b_4$** y **P5** a @: **$b_{11}$**.

• Una vez calculadas y puestas en su lugar las paridades parciales, se calcula la paridad global de la palabra y se coloca en el bit menos significativo (|número| = 0). De esta forma, se tiene la palabra ampliada completa en código Hamming de distancia mínima 4; bien entendido que, conforme a la numeración de los bits, las columnas están ordenadas del bit menos significativo (el primero de arriba) al más significativo (el último de abajo).

Sea la palabra inicial **1 0 0 1 0 1 1 1 0 1 0 0** que deseamos pasar a código Hamming de distancia 4:

| (1) número de orden | (2) *grafismo* | (3) paridades parciales | (4) Posiciones de los bits | (5) palabra inicial | (6) cálculo de paridades | (7) paridad global |
|---|---|---|---|---|---|---|
| 00000 | ———— | **P0** | **P0** | | | **0** |
| 00001 | ———— & | **P1** [&] | **P1** [&] | | **0** | **0** |
| 00010 | ——— # — | **P2** [#] | **P2** [#] | | **1** | **1** |
| 00011 | —— — # & | | **bit 0** | **0** | **0** | **0** |
| 00100 | —— Δ —— | **P3** [Δ] | **P3** [Δ] | | **0** | **0** |
| 00101 | —— Δ — & | | **bit 1** | **0** | **0** | **0** |
| 00110 | —— Δ # — | | **bit 2** | **1** | **1** | **1** |
| 00111 | —— Δ # & | | **bit 3** | **0** | **0** | **0** |
| 01000 | — * ——— | **P4** [*] | **P4** [*] | | **0** | **0** |
| 01001 | — * —— & | | **bit 4** | **1** | **1** | **1** |
| 01010 | — * — # — | | **bit 5** | **1** | **1** | **1** |
| 01011 | — * — # & | | **bit 6** | **1** | **1** | **1** |
| 01100 | — * Δ —— | | **bit 7** | **0** | **0** | **0** |
| 01101 | — * Δ — & | | **bit 8** | **1** | **1** | **1** |
| 01110 | — * Δ # — | | **bit 9** | **0** | **0** | **0** |
| 01111 | — * Δ # & | | **bit 10** | **0** | **0** | **0** |
| 10000 | @ ———— | **P5** [@] | **P3** [@] | | **1** | **1** |
| 10001 | @ ——— & | | **bit 11** | **1** | **1** | **1** |

Palabra inicial: **1 0 0 1 0 1 1 1 0 1 0 0**

Código Hamming: **1 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 0**

(Se han destacado y subrayado los bits que corresponden a las paridades).

Es fácil comprobar la distancia mínima entre dos palabras de este código; consideremos dos palabras iniciales diferentes (cuya distancia sea menor de 3):

- si se diferencian en un solo bit, diferirán también en, al menos, dos bits de paridad parcial (ya que el número de orden del bit modificado tendrá, por lo menos, dos unos),
- si tienen dos bits diferentes, lo será también, al menos, uno de los bits de paridad (ya que los números de orden de los bits modificados diferirán, cuando menos, en un uno),

o sea que la distancia entre dos palabras ampliadas no puede ser inferior a 3; además, como el código incluye la paridad global, su distancia será siempre múltiplo de 2, es decir, entre dos palabras ampliadas habrá una distancia mínima de 4.



### *6.3.2. Detección y corrección de errores*

• La verificación respecto a si una palabra ampliada pertenece o no al código se realiza comprobando las paridades parciales y la paridad global de la palabra completa; si la palabra es correcta (si no hay errores detectables), las nuevas paridades deben ser, todas ellas, nulas pues corresponden a conjuntos de bits ampliados con su propia paridad.

• La comprobación de la paridad global se calcula sobre todos los bits de la palabra ampliada y su resultado puede ser:

- comprobación de la paridad global = **0** y, en tal caso, o no existe error, o éste afecta a un número par de dígitos y no se puede hacer corrección de error sobre la palabra recibida
- comprobación de la paridad global = **1** y, en tal caso, existe error y, además, afecta a un número impar de dígitos; en principio, puede suponerse que afecta a un solo bit (pues, salvo sistemas de transmisión o almacenamiento muy defectuosos, es mucho más probable que haya error en un bit que en tres o más de ellos) y, consiguientemente, es viable realizar la corrección de dicho error.

• Cada comprobación de paridad parcial se calcula sobre los bits cuyo |`número`| contiene un «uno» en la misma posición; es decir, se toman todos los bits señalados con el mismo `grafismo` y se halla la paridad del conjunto de ellos.

La nueva paridad parcial **CP1** es la de todos los bits cuyo |`número`| acaba por «uno», es decir, aquellos cuyo `grafismo` es &: **$b_{11}$ $b_{10}$ $b_8$ $b_6$ $b_4$ $b_3$ $b_1$ $b_0$ P1**. La comprobación de paridad parcial **CP2** es la de los bits cuyo |`número`| tiene un «uno» en penúltima posición, es decir su grafismo es #: **$b_{10}$ $b_9$ $b_6$ $b_5$ $b_3$ $b_2$ $b_0$P2**. La comprobación **CP3** corresponde a un «uno» en antepenúltima posición, o sea al `grafismo` Δ: **$b_{10}$ $b_9$ $b_8$ $b_7$ $b_3$ $b_2$ $b_1$ P3**; **CP4** se refiere al `grafismo` *: **$b_{10}$ $b_9$ $b_8$ $b_7$ $b_6$ $b_5$ $b_4$ P4** y **CP5** a @: **$b_{11}$ P5**.

• En caso de que la palabra pertenezca al código Hamming todas las comprobaciones de paridad darán resultado **0**, tanto las parciales como la global. Tal cosa sucederá cuando no haya habido error en la transferencia de la palabra; nunca podremos estar absolutamente seguros de la ausencia de error, pero si todas las comprobaciones de paridad son nulas, sabemos que, de haber error hay cuatro o más errores (y siempre en número par), lo cual es altamente improbable.

• La comprobación de paridades parciales da lugar a un número binario …**CP5 CP4 CP3 CP2 CP1**, con las siguientes posibilidades:

- si dicho número es nulo y la paridad global también lo es, estamos en el caso anterior y, en principio, aceptaremos (con muy alta probabilidad) la ausencia de error;
- si este número no es nulo y la paridad global es **0**, hay error y afecta a un número par de dígitos, por lo cual no podemos corregirlo;
- si tal número no es nulo y la paridad global es **1**, es razonable suponer que el error afecta a un solo bit y es posible corregirlo: el número …**CP5 CP4 CP3 CP2 CP1** señala al dígito erróneo (ver la justificación en la página 154).

En la palabra ampliada **1 1 0 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0**, obtenida en el subapartado anterior (6.3.1.) que corresponde a la inicial **1 0 0 1 0 1 1 1 0 1 0 0**, introducimos un error (invertimos el bit nº 9) **1 1 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0**. La manera de aplicar el procedimiento de detección de error es la siguiente:

| **(1)** | **(2)** | **(3)** | **(4)** | **(5)** | **(6)** | **(7)** |
|---|---|---|---|---|---|---|
| **número** | | Posiciones | **comprobación** | **palabra** | **cálculo de** | **error** |
| **de orden** | ***grafismo*** | de los bits | **de paridades** | **ampliada** | **CP** | |
| 00000 | ———— — | **P0** | **CP0** | **0** | **1** | **← Si** |
| 00001 | ———— & | **P1** | **CP1** [&] | **0** | **1** | ***posición*** |
| 00010 | ——— # — | **P2** | **CP2** [#] | **1** | **0** | ***del bit*** |
| 00011 | —— — # & | **bit 0** | | **0** | | ***erróneo:*** |
| 00100 | —— Δ— — | **P3** [Δ] | **CP3** [Δ] | **0** | **0** | ***01001*** |
| 00101 | —— Δ— & | **bit 1** | | **0** | | |
| 00110 | —— Δ # — | **bit 2** | | **1** | | |
| 00111 | —— Δ # & | **bit 3** | | **0** | | |
| 01000 | — * ——— | **P4** | **CP4** [*] | **0** | **1** | |
| 01001 | — * —— & | **bit 4** | | **0** | | **←** |
| 01010 | — * — # — | **bit 5** | | **1** | | |
| 01011 | — * — # & | **bit 6** | | **1** | | |
| 01100 | — * Δ— — | **bit 7** | | **0** | | |
| 01101 | — * Δ— & | **bit 8** | | **1** | | |
| 01110 | — * Δ # — | **bit 9** | | **0** | | |
| 01111 | — * Δ # & | **bit 10** | | **0** | | |
| 10000 | @——— | **P3** | **CP5** [@] | **1** | **0** | |
| 10001 | @——— & | **bit 11** | | **1** | | |

Se debe corregir el bit cuyo |número| es **0 1 0 0 1**, según indica el número que forman las paridades parciales.

Código Hamming correcto: **1 1 0 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0**

Palabra inicial: **1 0 0 1 0 1 1 1 0 1 0 0**

La palabra inicial se obtiene a partir de la palabra ampliada, eliminando en ella las paridades, tanto las parciales como la global. [Compruébese que es correcta, es decir, que coincide con la palabra inicial del apartado 6.3.1.]



Cuando el número …**CP5 CP4 CP3 CP2 CP1** no es nulo y la paridad global es **1**, sabemos que hay error y que afecta a un número impar de dígitos; podemos suponer que el error afecta a un solo bit y, en tal caso, el número …**CP5 CP4 CP3 CP2 CP1** señala al dígito erróneo:

- **CP1 = 1** significa que el error se encuentra entre los bits que corresponden a la paridad **P1** (es decir, entre aquellos cuyo |número| acaba por «uno», grafismo &),
- **CP2 = 1** significa que el error se encuentra entre los bits que corresponden a la paridad **P2** (es decir, su |número| tiene un «uno» en penúltima posición, grafismo #)
- y así sucesivamente …;

de manera que el número binario que forman …**CP5 CP4 CP3 CP2 CP1** corresponde precisamente al |número| del bit erróneo.

La siguiente tabla indica el número de bits de paridad necesarios en el código de Hamming de distancia mínima 4 según el número de bits de la palabra inicial:

| | | **número de bits** | |
|---|---|---|---|
| *número de bits de paridad parcial* | *nº total de bits de paridad* | número máximo de bits de la palabra ampliada | **nº máximo de bits de la palabra inicial** |
| 3 | 3 + 1 = **4** | $2^3 = 8$ | 8 – 4 = **4** |
| 4 | 4 + 1 = **5** | $2^4 = 16$ | 16 – 5 = **11** |
| 5 | 5 + 1 = **6** | $2^5 = 32$ | 32 – 6 = **26** |
| 6 | 6 + 1 = **7** | $2^6 = 64$ | 64 – 7 = **57** |
| 7 | 7 + 1 = **8** | $2^7 = 128$ | 128 – 8 = **120** |
| 8 | 8 + 1 = **9** | $2^8 = 256$ | 256 – 9 = **247** |

Una palabra inicial de 4 dígitos duplica su longitud al pasar a código Hamming de distancia mínima 4, si es de 1 byte requiere 5 dígitos adicionales de paridad (pasa a 13 bits, un aumento del 65 %) y para palabras de 16 bits es preciso añadir otros 6 (40%). En cambio, una palabra inicial de 32 bits aumenta solamente en 7 más (22%) y una de 120 bits se amplía a 128 (un 7%).

Con este mismo tipo de idea conceptual (la de introducir adecuadamente paridades parciales) pueden construirse códigos más complejos de distancia mínima superior. La detección y corrección de errores, es decir, la fiabilidad de la información es un tema de interés cada vez mayor y constituye una rama especializada dentro del amplio campo de la codificación de la información.

Consideremos otro ejemplo, relativo a una palabra inicial de 10 dígitos:

*Sea una palabra binaria inicial de 10 bits:* **1 1 0 0 1 1 1 0 0 0**

*que en código de Hamming de distancia mínima 4 será:* **1 1 0 0 1 1 0 1 0 0 1 0 1 1 0**

*(los dígitos subrayados corresponden a los bits de paridad).*

***a)*** *si en la palabra inicial se modifica el bit* $b_5$*:* **1 1 0 0 0 1 1 0 0 0**

*su correspondiente palabra codificada será:* **1 1 0 0 0 1 1 1 0 0 1 0 0 1 1**

*que dista 4 bits de la anterior.*

***b)*** *si en la palabra inicial se modifican los bits* $b_5$ *y* $b_6$*:* **1 1 0 1 0 1 1 0 0 0**

*su correspondiente será:* **1 1 0 1 0 1 0 1 0 0 1 0 1 0 1**

*que dista 4 bits de la primera y otros 4 bits de la anterior.*

***c)*** *si en la palabra inicial se modifican los bits* $b_5$ *,* $b_6$ *y* $b_7$*:* **1 1 1 1 0 1 1 0 0 0**

*su correspondiente palabra codificada será:* **1 1 1 1 0 1 1 1 0 0 0 0 1 0 0**

*que dista 4 bits de la palabras anterior y 6 bits de las dos primeras.*

***d)*** *si en la transmisión de la primera palabra ampliada se invierte un solo bit, el que hace el número de orden sexto:* **1 1 0 0 0 1 1 1 1 0 1 0 0 1 1**

*la palabra recibida corresponde a una palabra errónea, es decir, que no coincide con la primera palabra inicial:* **1 1 0 0 0 1 1 1 0 0**

*pero la comprobación de la paridad global da* ***1*** *(error en número impar de bits) y el número correspondiente a las comprobaciones de paridades parciales valdrá* ***0110****, lo cual indica error en el bit cuyo número de orden es 6.*

***e)*** *invirtiendo dicho bit 6 se recupera la palabra correcta:* **1 1 0 0 0 1 1 1 0 0 1 0 0 1 1**

*y su correspondiente palabra inicial:* **1 1 0 0 0 1 1 0 0 0**

***f)*** *si en la transmisión de la primera palabra se invierten los bits que hacen los números de orden sexto y séptimo:* **1 1 0 0 0 1 1 0 1 0 1 0 0 1 1**

*el número correspondiente a las comprobaciones de paridades parciales valdrá* ***0001****, lo cual indica error en el bit cuyo número de orden es 1*

*corrigiéndolo se generaría la palabra:* **1 1 0 0 0 1 1 0 1 0 1 0 0 0 1**

*que es errónea pues corresponde a la palabra inicial:* **1 1 0 0 0 1 0 1 0 0**

*distinta de la primera. Ello es debido a que el error afectaba a dos bits: la comprobación de la paridad global da* ***0*** *(error en número par de bits).*



Conviene tener presente la relación entre la detección y corrección de errores y la redundancia en la representación de la información. Como hemos visto en este apartado y en el anterior (6.2. y 6.3.) la detección y corrección de errores se basa en utilizar formas de expresar la información (códigos) que no sean mínimas, sino formas expandidas con suficiente redundancia.

De hecho en el lenguaje hablado continuamente reajustamos la información gracias a la redundancia: palabras que se pronuncian o se escuchan a medias, sílabas trastocadas o suprimidas, discordancias gramaticales, pequeñas faltas de coherencia, etc.,… no suponen ningún problema porque el receptor las completa o corrige en relación con el contexto, gracias a que existe redundancia. Sin ella, cualquier conversación sería muy dificultosa y cualquier lectura precisaría de una atención agotadora.

En muchas ocasiones, la pregunta: ¿puedes explicarme eso más despacio?, no es sino una solicitud de mayor redundancia.

De igual forma, los códigos binarios pueden detectar errores cuando no son mínimos, cuando se amplían con bits añadidos que permiten efectuar comprobaciones y reajustes; en definitiva, códigos con redundancia.

Al incorporar bits adicionales, existirán palabras que pertenecen al código y otras que no pertenecen al mismo, de forma que, si al producirse error la palabra resultante no pertenece al código, ello nos permite detectar la existencia de error. La adecuada combinación de los bits añadidos permite también la reparación de la palabra errónea cuando el número de errores es limitado.