❸ Códigos Binarios

No toda la información que maneja un sistema digital es numérica, e inclusive, para la información numérica a veces no es conveniente utilizar el sistema binario descrito en los capítulos anteriores. Por ello es conveniente idear formas diferentes de representar (codificar) información diversa usando solamente ceros y unos. en este capítulo se describen algunos códigos tanto para información numérica como alfanumérica, cuyo uso se ha generalizado por diversas razones, la mayoría de las veces de conveniencia, aunque no siempre.

3.1.- CÓDIGOS NUMÉRICOS

Existen varias situaciones en las que no es conveniente utilizar el binario natural para manejar información numérica, entre ellas se pueden mencionar las siguientes:

Cuando se busca una conversión más sencilla decimal - binario (códigos BCD)
Cuando además de lo anterior se van a manejar números negativos (Códigos BCD autocomplementarios)
Cuando se busca minimizar errores de sensado en "encoders" de posición de una cantidad a otra (código gray)
Cuando se quiere detectar errores en transmisión de datos (código de paridad)
Cuando se quiere detectar y corregir errores en transmisión de datos (código Hamming)

A continuación se tratan algunos de estos códigos.

3.1.1.- DECIMAL CODIFICADO EN BINARIO (BCD)

Los códigos BCD nos permiten representar cada uno de los dígitos decimales (0,...,9) mediante 4 bits.

El más sencillo de los códigos BCD es el **BCD_{8421}** o BCD "natural", que consiste simplemente en representar cada dígito decimal por su binario equivalente. Así tenemos

Dígito Decimal	BCD_{8421}	Dígito Decimal	BCD_{8421}
0	0 0 0 0	5	0 1 0 1
1	0 0 0 1	6	0 1 1 0
2	0 0 1 0	7	0 1 1 1
3	0 0 1 1	8	1 0 0 0
4	0 1 0 0	9	1 0 0 1

Ejemplo: Expresar e 937.25_{10} en BCD.
937.25_{10} = 1001 0011 0111 0010 0101_{BCD}

Ejemplo: Expresar el número N = $(10010110010111)_{BCD}$ escrito en código BCD_{8421}, en decimal.

separando de LSB a MSB en grupos de 4: $N = (10,0101,1001,0111)_{BCD} = 2597_{10}$

3.1.2.- CÓDIGO BCD EXCESO-3

El código BCD exceso-3 se obtiene a partir del código BCD natural, simplemente sumando 3_{10} (0011_2) a cada código BCD de cada dígito decimal. Esto se resume en la siguiente tabla

Dígito Decimal	BCD EXCESO-3	Dígito Decimal	BCD EXCESO-3
0	0 0 1 1	5	1 0 0 0
1	0 1 0 0	6	1 0 0 1
2	0 1 0 1	7	1 0 1 0
3	0 1 1 0	8	1 0 1 1
4	0 1 1 1	9	1 1 0 0

Este código resulta de utilidad en aplicaciones donde se requiere realizar operaciones aritméticas usando complementos. Este código es llamado *autocomplementario* porque el complemento a 9 de un numero decimal puede ser obtenido complementando cada bit individualmente y el resultado sigue siendo un código válido en BCD exceso 3.

Ejemplo: Representar el numero 907_{10} en BCD exceso-3 y usar el complemento a 1 para encontrar el complemento a 9 del número:

$907_{10} = 1100\ 0011\ 1010_{exc\text{-}3}$
$0011\ 1100\ 0101_{exc\text{-}3}$ complemento a 1
$= 092_{10}$ complemento a 9

3.1.3.- CÓDIGO BCD 2421

Este es otro código BCD autocomplementario, y su nombre (2421) indica la ponderación de sus bits para obtener su equivalente en decimal y biceversa. en la siguiente tabla se ilustra este código

Dígito Decimal	BCD 2421	Dígito Decimal	BCD 2421
0	0 0 0 0	5	1 0 1 1
1	0 0 0 1	6	1 1 0 0
2	0 0 1 0	7	1 1 0 1
3	0 0 1 1	8	1 1 1 0
4	0 1 0 0	9	1 1 1 1

Ejemplo: Representar el numero 907_{10} en BCD exceso-3 y usar el complemento a 1 para encontrar el complemento a 9 del número:

907_{10} = 1111 0000 1101_{2421}
0000 1111 0010_{2421} complemento a 1
= 092_{10} complemento a 9

3.1.4.- CÓDIGO 2 DE 5 (BIQUINARIO)

El código 2 de 5 es un código multibit no ponderado, es decir, los códigos no pueden obtenerse usando una expresión polinomial; este código está diseñado para la detección de errores en diferentes tipos de cálculos y operaciones con registros de corrimiento. Se usan cinco bits para representar los dígitos decimales (0-9). Como el nombre lo implica sólo dos de los cinco bits son 1.

Dígito Decimal	Código 2 de 5	Dígito Decimal	Código 2 de 5
0	0 0 0 1 1	5	0 1 1 0 0
1	0 0 1 0 1	6	1 0 0 0 1
2	0 0 1 1 0	7	1 0 0 1 0
3	0 1 0 0 1	8	1 0 1 0 0
4	0 1 0 1 0	9	1 1 0 0 0

Ejemplo: Representar el numero decimal 237_{10} en código 2 de 5.

237_{10} = 00110 01001 $10010_{2\text{ de }5}$

3.2.- CÓDIGO GRAY

Este es un código binario no ponderado y tiene la propiedad de que los códigos para dígitos decimales sucesivos difiere en un sólo bit. al código Gray también se le llama autorreflejado, o cíclico. En la siguiente tabla se muestra dicho código para los números del 0 al 16

Dígito Decimal	Código Gray	Dígito Decimal	Código Gray
0	0 0 0 0	8	1 1 0 0
1	0 0 0 1	9	1 1 0 1
2	0 0 1 1	10	1 1 1 1
3	0 0 1 0	11	1 1 1 0
4	0 1 1 0	12	1 0 1 0
5	0 1 1 1	13	1 0 1 1
6	0 1 0 1	14	1 0 0 1
7	0 1 0 0	15	1 0 0 0

3.2.1.- CONVERSIÓN GRAY - BINARIO

Para convertir de Binario a Gray puede seguirse el siguiente procedimiento

Algoritmo

1.- El MSB se deja igual
2.- Avanzando de MSB a LSB se suma cada bit con el siguiente despreciando el acarreo para obtener el siguiente bit del código Gray

Ejemplo Escribir en Código Gray el número 45_{10}

Como $45_{10} = 101101_2$ Al aplicar el algoritmo a este número binario, tenemos:

$$\begin{array}{cccccc} 1 \overset{\oplus}{\rightarrow} & 0 \overset{\oplus}{\rightarrow} & 1 \overset{\oplus}{\rightarrow} & 1 \overset{\oplus}{\rightarrow} & 0 \overset{\oplus}{\rightarrow} & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1 & 0 & 1 & 1 \end{array}$$

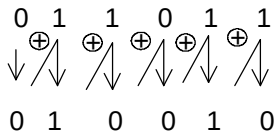
Es decir, $45_{10} = 1\ 1\ 1\ 0\ 1\ 1_{gray}$

Para convertir de Gray a Binario puede seguirse el siguiente procedimiento

Algoritmo

1.- El MSB se deja igual
2.- Avanzando de MSB a LSB a cada bit obtenido en binario se le suma sin acarreo el siguiente bit de código Gray.

Ejemplo Obtener el equivalente decimal del siguiente código gray: $N = 011011_{gray}$



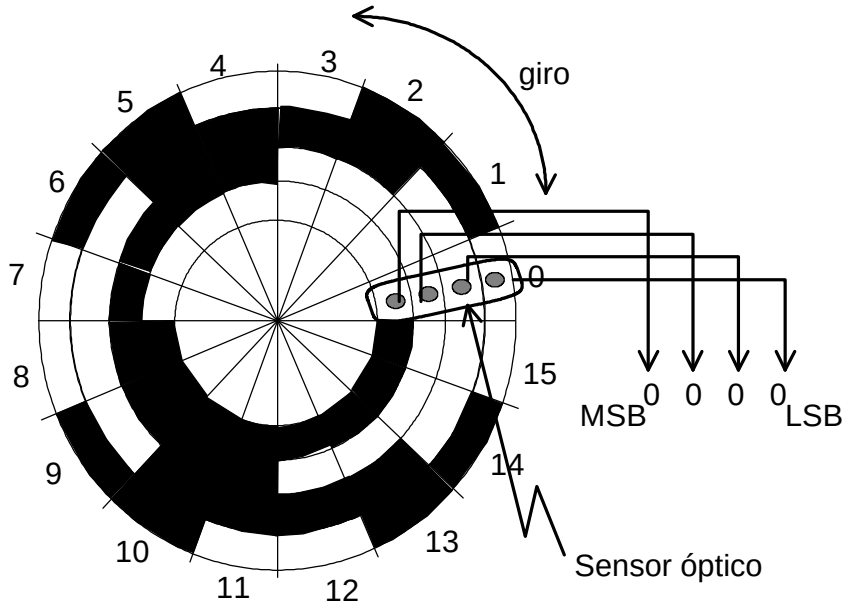
Al aplicar el algoritmo a este número binario, tenemos:
Es decir, $N = 010010_2 = 18_{10}$

3.2.2.- APLICACIÓN A SENSORES ÓPTICOS

La principal característica del código gray (que sólo cambia un bit entre dos códigos consecutivos) es muy utilizada en sensores ópticos para codificar la posición (angular o lineal) mediante discos o cintas codificadas en gray, dependiendo del caso.

Por ejemplo, en la siguiente figura se ilustra la codificación en dos colores (blanco y negro) de un disco que será montado sobre la flecha de un motor. Se usa el código gray de 4 bits para codificar la posición angular de la flecha. Cada sector en el disco tiene un código de posición de 0,a 15. Se usarán sensores opto electrónicos reflectivos para determinar la posición de la flecha en un sector o

sea, con una resolución de 1/16 de vuelta, o de 22.5°. si se desea una mejor resolución se deberán usar más bits, y por lo tanto mayor número de sectores.



Aunque el disco pudiera se codificado en binario natural, el hacerlo en gray tiene la ventaja de que si el sensor queda ubicado entre dos sectores, la lectura producida producirá un error de cuando mucho media posición. En cambio, si es en binario este error puede ser tan grande como 180°.

3.5.- CÓDIGOS ALFANUMÉRICOS

Muchas aplicaciones de sistemas digitales (especialmente las computadoras o la transmisión de textos) requieren del procesamiento de datos los como números, letras y símbolos especiales. Para manejar estos datos usando dispositivos digitales, cada símbolo debe estar representado por un código binario. El código alfanumérico más generalizado en la actualidad es el denominado ASCII (American Standard Code for Information Interchange). Este es un código de 7 bit. La siguiente tabla muestra una parte del código ASCII:

Ejemplo: la palabra "Start" se representa en código ASCII como sigue

1010011	1110100	1100001	1110010	1110100
S	t	a	r	t

Caracter	hex	dec	ASCII $A_6A_5A_4A_3A_2A_1A_0$	Caracter	hex	dec	ASCII $A_6A_5A_4A_3A_2A_1A_0$	Caracter	hex	dec	ASCII $A_6A_5A_4A_3A_2A_1A_0$	Caracter	hex	dec	ASCII $A_6A_5A_4A_3A_2A_1A_0$
nulo	0	0	0 0 0 0 0 0 0	espacio	20	32	0 1 0 0 0 0 0	**@**	40	0	1 0 0 0 0 0 0	**`**	60	0	1 1 0 0 0 0 0
SOH	1	1	0 0 0 0 0 0 1	**!**	21	33	0 1 0 0 0 0 1	**A**	41	1	1 0 0 0 0 0 1	**a**	61	1	1 1 0 0 0 0 1
STX	2	2	0 0 0 0 0 1 0	**"**	22	34	0 1 0 0 0 1 0	**B**	42	2	1 0 0 0 0 1 0	**b**	62	2	1 1 0 0 0 1 0
ETX	3	3	0 0 0 0 0 1 1	**#**	23	35	0 1 0 0 0 1 1	**C**	43	3	1 0 0 0 0 1 1	**c**	63	3	1 1 0 0 0 1 1
EOT	4	4	0 0 0 0 1 0 0	**$**	24	36	0 1 0 0 1 0 0	**D**	44	4	1 0 0 0 1 0 0	**d**	64	4	1 1 0 0 1 0 0
ENQ	5	5	0 0 0 0 1 0 1	**%**	25	37	0 1 0 0 1 0 1	**E**	45	5	1 0 0 0 1 0 1	**e**	65	5	1 1 0 0 1 0 1
ACK	6	6	0 0 0 0 1 1 0	**&**	26	38	0 1 0 0 1 1 0	**F**	46	6	1 0 0 0 1 1 0	**f**	66	6	1 1 0 0 1 1 0
BELL	7	7	0 0 0 0 1 1 1	**'**	27	39	0 1 0 0 1 1 1	**G**	47	7	1 0 0 0 1 1 1	**g**	67	7	1 1 0 0 1 1 1
BS	8	8	0 0 0 1 0 0 0	**(**	28	40	0 1 0 1 0 0 0	**H**	48	8	1 0 0 1 0 0 0	**h**	68	8	1 1 0 1 0 0 0
HT	9	9	0 0 0 1 0 0 1	**)**	29	41	0 1 0 1 0 0 1	**I**	49	9	1 0 0 1 0 0 1	**i**	69	9	1 1 0 1 0 0 1
LF	A	10	0 0 0 1 0 1 0	*****	2A	42	0 1 0 1 0 1 0	**J**	4A	10	1 0 0 1 0 1 0	**j**	6A	10	1 1 0 1 0 1 0
VT	B	11	0 0 0 1 0 1 1	**+**	2B	43	0 1 0 1 0 1 1	**K**	4B	11	1 0 0 1 0 1 1	**k**	6B	11	1 1 0 1 0 1 1
FF	C	12	0 0 0 1 1 0 0	**,**	2C	44	0 1 0 1 1 0 0	**L**	4C	12	1 0 0 1 1 0 0	**l**	6C	12	1 1 0 1 1 0 0
return	D	13	0 0 0 1 1 0 1	**-**	2D	45	0 1 0 1 1 0 1	**M**	4D	13	1 0 0 1 1 0 1	**m**	6D	13	1 1 0 1 1 0 1
SO	E	14	0 0 0 1 1 1 0	**.**	2E	46	0 1 0 1 1 1 0	**N**	4E	14	1 0 0 1 1 1 0	**n**	6E	14	1 1 0 1 1 1 0
SI	F	15	0 0 0 1 1 1 1	**/**	2F	47	0 1 0 1 1 1 1	**O**	4F	15	1 0 0 1 1 1 1	**o**	6F	15	1 1 0 1 1 1 1
DLE	10	16	0 0 1 0 0 0 0	**0**	30	48	0 1 1 0 0 0 0	**P**	50	16	1 0 1 0 0 0 0	**p**	70	16	1 1 1 0 0 0
DC1	11	17	0 0 1 0 0 0 1	**1**	31	49	0 1 1 0 0 0 1	**Q**	51	17	1 0 1 0 0 0 1	**q**	71	17	1 1 1 0 0 0 1
DC2	12	18	0 0 1 0 0 1 0	**2**	32	50	0 1 1 0 0 1 0	**R**	52	18	1 0 1 0 0 1 0	**r**	72	18	1 1 1 0 0 1 0
DC3	13	19	0 0 1 0 0 1 1	**3**	33	51	0 1 1 0 0 1 1	**S**	53	19	1 0 1 0 0 1 1	**s**	73	19	1 1 1 0 0 1 1
DC4	14	20	0 0 1 0 1 0 0	**4**	34	52	0 1 1 0 1 0 0	**T**	54	20	1 0 1 0 1 0 0	**t**	74	20	1 1 1 0 1 0 0
NAK	15	21	0 0 1 0 1 0 1	**5**	35	53	0 1 1 0 1 0 1	**U**	55	21	1 0 1 0 1 0 1	**u**	75	21	1 1 1 0 1 0 1
SYN	16	22	0 0 1 0 1 1 0	**6**	36	54	0 1 1 0 1 1 0	**V**	56	22	1 0 1 0 1 1 0	**v**	76	22	1 1 1 0 1 1 0
ETB	17	23	0 0 1 0 1 1 1	**7**	37	55	0 1 1 0 1 1 1	**W**	57	23	1 0 1 0 1 1 1	**w**	77	23	1 1 1 0 1 1 1
CAN	18	24	0 0 1 1 0 0 0	**8**	38	56	0 1 1 1 0 0 0	**X**	58	24	1 0 1 1 0 0 0	**x**	78	24	1 1 1 1 0 0 0
EM	19	25	0 0 1 1 0 0 1	**9**	39	57	0 1 1 1 0 0 1	**Y**	59	25	1 0 1 1 0 0 1	**y**	79	25	1 1 1 1 0 0 1
SUB	1A	26	0 0 1 1 0 1 0	**:**	3A	58	0 1 1 1 0 1 0	**Z**	5A	26	1 0 1 1 0 1 0	**z**	7A	26	1 1 1 1 0 1 0
ESC	1B	27	0 0 1 1 0 1 1	**;**	3B	59	0 1 1 1 0 1 1	**[**	5B	27	1 0 1 1 0 1 1	**{**	7B	27	1 1 1 1 0 1 1
FS	1C	28	0 0 1 1 1 0 0	**<**	3C	60	0 1 1 1 1 0 0	****	5C	28	1 0 1 1 1 0 0	**|**	7C	28	1 1 1 1 1 0 0
GS	1D	29	0 0 1 1 1 0 1	**=**	3D	61	0 1 1 1 1 0 1	**]**	5D	29	1 0 1 1 1 0 1	**}**	7D	29	1 1 1 1 1 0 1
RS	1E	30	0 0 1 1 1 1 0	**>**	3E	62	0 1 1 1 1 1 0	**^**	5E	30	1 0 1 1 1 1 0	**~**	7E	30	1 1 1 1 1 1 0
US	1F	31	0 0 1 1 1 1 1	**?**	3F	63	0 1 1 1 1 1 1	**_**	5F	31	1 0 1 1 1 1 1	**del**	7F	31	1 1 1 1 1 1 1

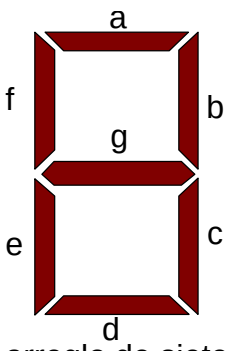
En resumen, el código ASCII consta entre otros, de los siguientes grupos de caracteres:

Rango (hexadecimal)	Caracteres
0 al 19	códigos de control y comunicaciones
30 al 39	dígitos del 0 al 9
41 al 5A	letras mayúsculas de la A a la Z
61 al 7A	letras minúsculas de la “a” a la z

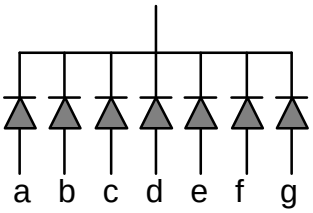
3.5.1.- CODIGO DE SIETE SEGMENTOS.

Un dispositivo muy generalizado por su sencillez y bajo costo en dispositivos digitales de visualización es el exhibidor o display de siete segmentos, el cual consiste en un arreglo de siete indicadores luminosos (LED's) u opacos (cristal líquido) arreglado como se muestra en la siguiente figura. Existen dos tipos de exhibidores de siete segmentos construidos con LED's, estos son los de ánodo común y los de cátodo común, los cuales se muestran también en la figura.

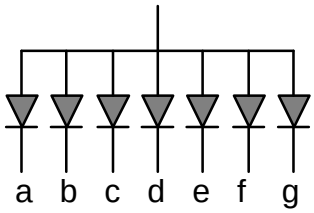
Este tipo de displays permite la representación de información de tipo numérico principalmente, sin



arreglo de siete segmentos



cátodo común



ánodo común

embargo, también permite algunos caracteres alfabéticos, tales como: a, A, b, c, C, d, E, F, G, H, y, j, L, o, O, p, q, r, s, u, z.

En la siguiente tabla se muestra el **código de 7 segmentos para un display de ánodo común** para los dígitos decimales y el equivalente en BCD:

Decimal	BCD	a	b	c	d	e	f	g	display
0	0 0 0 0	0	0	0	0	0	0	1	
1	0 0 0 1	1	0	0	1	1	1	1	
2	0 0 1 0	0	0	1	0	0	1	0	
3	0 0 1 1	0	0	0	0	1	1	0	
4	0 1 0 0	1	0	0	1	1	0	0	
5	0 1 0 1	0	1	0	0	1	0	0	
6	0 1 1 0	0	1	1	1	1	1	1	
7	0 1 1 1	0	0	0	1	1	1	1	
8	1 0 0 0	0	0	0	0	0	0	0	
9	1 0 0 1	0	0	0	0	1	0	0	

3.6.- CÓDIGOS PARA DETECCIÓN Y CORRECCIÓN DE ERRORES

Los sistemas digitales pueden cometer errores de vez en cuando. Aunque los dispositivos en circuito integrado tales como microprocesadores, puertas lógicas o circuitos de memoria carecen de partes móviles y por lo tanto tienen alta confiabilidad. pero los dispositivos que tienen interacción con partes móviles son menos confiables. Se pueden producir errores por polvo en las cabezas lectoras de una unidad de disco. También es muy común la ocurrencia de errores en la *transmisión de datos a distancia*. Los datos que se transmiten por modem (a través de línea telefónica) pueden recibirse incorrectamente si la línea tiene ruidos. También la perturbaciones en el suministro de energía eléctrica pueden producir errores. En resumen, cuando se leen, escriben o transmiten

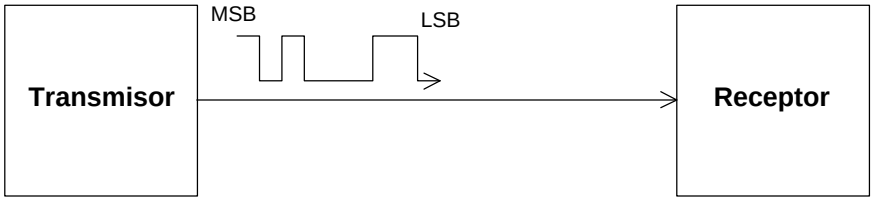
caracteres de un sitio a otro, pueden producirse errores. En esta sección se ilustran dos códigos que permiten detectar errores y, en algunos casos, incluso corregirlos.

3.6.1 TRANSMISIÓN SERIE Y PARALELO

Existen dos métodos básicos para transmitir información entre dos dispositivos digitales, como se ilustra en la siguiente figura, la transmisión de datos digitales en paralelo ocupa una línea física por cada bit de los datos a enviar, esto hace que este tipo de transmisión sea usada en distancias relativamente cortas (unos cuantos metros), ya que de otra manera se vuelve muy costosa.

Transmisión paralela del caracter "F"

Por otro lado, la transmisión de datos a distancias muy grandes (del orden de cientos de metros a miles de kilómetros) hace necesario usar menos líneas físicas. en este caso puede usarse el esquema de la figura siguiente o transmisión en serie, la cual utiliza una sola línea para enviar el número de bits que se desee.



Transmisión serie del caracter "F"

Aunque la transmisión en serie resulta económica por el número reducido de líneas que requiere, e inclusive, por el hecho de que permite usar en lugar de líneas conductoras, el aire o el vacío cuando en lugar de pulsos eléctricos se usa radiación electromagnética (luz, ondas de radio, ultrasonido, etc.). A cambio de esta economía, el transmisor y el receptor se vuelven un poco más complejos, además de que el trayecto que recorre la información (por ser largo) se vuelve más susceptible a interferencias.

3.6.2.- CÓDIGO DE PARIDAD

Un método muy simple, pero ampliamente utilizado por su sencillez para detectar errores en transmisión de datos consiste en añadir un **bit de paridad (p)** a cada carácter, normalmente en la posición más significativa.

En el código de **paridad par**, el bit de paridad (p) se elige de manera que el número de bits 1 del dato sea un número par incluyendo el bit de paridad. En el código de **paridad impar**, el bit de paridad se elige de modo que el número de bits 1 (incluyendo el de paridad) del dato sea impar.

De esta manera, cuando cambia un bit durante la transmisión, el número de unos en el carácter recibido tendrá la paridad equivocada y el receptor sabrá que se ha producido un error.

Ejemplo: La siguiente tabla muestra un código de paridad par de 8 bits para los caracteres ASCII "FIE." .

Bit de paridad P	**Código ASCII de 7 bits** A_6	A_5	A_4	A_3	A_2	A_1	A_0	**Caracter**
1	1	0	0	0	1	1	0	F
1	1	0	0	1	0	1	0	I
1	1	0	0	0	1	0	1	E
0	0	1	0	1	1	1	0	.

Ejemplo Si un transmisor envía la información anterior y hay errores en la transmisión, suponiendo que el receptor recibe la siguiente información, en la siguiente tabla se anota los datos que llegaron erróneos y si se detectó o no el error, agrega en la columna vacía cuantos bits cambiaron en la transmisión

Dato enviado	**Dato recibido** (supuesto)	error	paridad	bits erróneos
1100,0110 (F)	1100,1110 (+)	SI	mal	1
1100,1010 (I)	0110,1011 (k)	SI	mal	3
1100,0101 (E)	1100,0101 (E)	NO	bien	0
0010,1110 (.)	0010,1101 (-)	SI	bien	2

Como puede verse, el código de paridad No siempre resulta efectivo para detectar errores. ¿Qué tipo de errores si detecta y cuales no?

3.6.3.- CÓDIGO DE HAMMING

Richard Hamming (1950) ideó un método no sólo para detectar errores sino también para corregirlos, y se conoce como **código Hamming**. En él se añaden **k** bits de paridad a un carácter de **n** bits, formando un nuevo carácter de **n + k** bits. Los bits se enumeran empezando por 1, no por 0, siendo el bit 1, el de la izquierda, el más significativo. Todo bit cuyo número sea potencia de 2 es un bit de paridad y todos los demás se utilizan para datos.
Para un carácter ASCII de 7 bits, se añaden 4 bits de paridad. Los bits 1, 2, 4 y 8 son bits de paridad; 3, 5, 6, 7, 9, 10 y 11 son los 7 bits de datos. Cada bit de paridad comprueba determinadas posiciones de bit y se ajusta de modo que el número total de unos en las posiciones comprobadas sea par, si se trata de paridad par.
Las posiciones de los bits comprobados por los de paridad son:

El bit 1 comprueba los bits 1, 3, 5, 7, 9 y 11.
El bit 2 comprueba los bits 2, 3, 6, 7, 10 y 11.
El bit 4 comprueba los bits 4, 5, 6 y 7.
El bit 8 comprueba los bits 8, 9, 10 y 11.

En general, el bit **n** es comprobado por los bits b_1, b_2,....,b_j, tales que $b_1 + b_2 + ... + b_j = \mathbf{n}$. Por ejemplo, el bit 5 es comprobado por los bits 1 y 4 porque 1 + 4 = 5. El bit 6 es comprobado por los bits 2 y 4 porque 2 + 4 = 6.

Ejemplo: Usando paridad par, construir el código de Hamming para el carácter "b".

Código ASCII para "b"

1	1	0	0	0	1	0

Código de Hamming para "b"

número de bit	1	2	3	4	5	6	7	8	9	10	11
	0	0	1	1	1	0	0	1	0	1	0

El carácter ASCII "b" se representa por el número binario 1100010. El código de Hamming para "b" en binario es 00111001010.
Considérese que pasaría si el bit 1 se modificara durante la transmisión. El carácter recibido sería 10111001010 en lugar de 00111001010. El receptor comprobaría los 4 bits de paridad con los resultados siguientes:

Bit de paridad 1 incorrecto (los bits 1, 3, 5, 7, 9 y 11 contienen tres unos).
Bit de paridad 2 correcto (los bits 2, 3, 6, 7, 10 y 11 contienen dos unos).
Bit de paridad 4 correcto (los bits 4, 5, 6 y 7 contienen dos unos).
Bit de paridad 8 correcto (los bits 8, 9, 10 y 11 contienen dos unos).

El número total de unos en los bits 1, 3, 5, 7, 9 y 11 debería de ser par, ya que se está usando paridad par. El bit incorrecto debe ser uno de los bits comprobados por el bit de paridad 1, es decir, uno de los bits 1, 3, 5, 7, 9 u 11. Como el bit de paridad 2 es correcto, sabemos que los bits 2, 3, 6, 7, 10 y 11 son correctos, de forma que el error no estaba en los bits 3, 7 u 11. Esto deja los bits 1, 5 y 9. El bit de paridad 4 es correcto, lo cual significa que los bits 4, 5, 6 y 7 no contienen errores.. Esto reduce la elección al 1 ó 9. El bit de paridad 8 también es correcto y, por lo tanto, el bit 9 es correcto. Por consiguiente, el bit incorrecto debe ser el 1. Dado que se recibió como un 1, debería haberse transmitido como un 0. En esta forma se pueden corregir los errores.