UN NUEVO ENFOQUE PARA INTRODUCIR LOS CÓDIGOS DE HAMMING EN LA INGENIERÍA ELECTRÓNICA

E. ALDABAS[1], M. CORBALÁN[2], A. ARIAS[2], I. PLAZA[3], C. MEDRANO[3] y F. ARCEGA[4]
[1]Dto. Ing. Eléctrica, [2]Dto. Ing. Electrónica, EUETIT Terrassa, UPC
[3]Dto. Ing. Electrónica y Comunicaciones, EUPT Teruel, Unizar
[4]Dto. Ing. Eléctrica, EUITIZ Zaragoza, Unizar
www.unizar.es/eduqtech/

En el presente trabajo se presenta una introducción al algoritmo publicado por el profesor Richard Wesley Hamming en 1950 para detectar y corregir un bit erróneo dentro de una palabra binaria de datos. En primer lugar se definen los conceptos previos necesarios para comprender el alcance del problema, a continuación se describe paso a paso la forma en que trabajan los ya denominados códigos de Hamming. Para finalizar se comentan unos ejemplos prácticos y se propone un circuito de Hamming (11,7) desarrollado en GNU Octave.

1. Introducción

En el año 1950, el profesor Richard W. Hamming [1] publicó un artículo sobre detección y corrección de errores [2]. Este trabajo supuso el comienzo de una nueva área de investigación dentro de la teoría de la información. Actualmente, los códigos de Hamming son fundamentales en la teoría de la codificación y tienen una gran cantidad de aplicaciones prácticas. En concreto, los códigos correctores de errores tienen un papel esencial en la vida cotidiana y son usados en modems, memorias, TDT e incluso en comunicaciones vía satélite.

La teoría de los códigos de Hamming es madura, difícil y con una orientación matemática. Sin embargo, hay multitud de artículos y libros que tratan este tema [3-8]. En cualquier caso, en los cursos de electrónica digital básica es difícil encontrar los códigos de Hamming [5] por falta de tiempo y porque es necesario introducir al alumno en conceptos más elementales que son ineludibles. Con el uso generalizado de internet, los alumnos tienen una importante fuente de información, y los códigos de Hamming no son una excepción. En cualquier caso, el presente trabajo pretende animar a los estudiantes de electrónica digital a que se introduzcan en este fascinante tema, y si ya lo conocen, a que vean en él alguna nueva faceta que les pasó desapercibida cuando lo estudiaron por primera vez.

Antes de comenzar el razonamiento para describir los códigos de Hamming es necesario tener presente las siguientes definiciones:

Código binario: Es una representación unívoca de las cantidades, de tal forma que a cada una de éstas se le asigna una combinación de símbolos binarios.

Distancia entre dos combinaciones binarias: Viene dada por el número de bits que hay que cambiar en una de ellas para obtener la otra.

Distancia mínima de un código: Es la menor de las distancias entre dos combinaciones binarias cualesquiera pertenecientes a dicho código.

En base a estas definiciones se concluye que un código binario debe tener al menos la distancia mínima igual a 1 para garantizar que una combinación no represente a varias cantidades o valores.

Los códigos son continuos cuando las combinaciones correspondientes a números decimales consecutivos difieren solamente en un bit. En el caso de que también se cumpla que la última combinación sea adyacente a al primera se está ante los códigos cíclicos. Los códigos ponderados asignan a cada bit un valor o peso en función del lugar que ocupan.

A	B	BPP
0	0	0
0	1	1
1	0	1
1	1	0

Control	Dato	Salida
0	0	0 (dato)
0	1	1 (dato)
1	0	1 (no dato)
1	1	0 (no dato)

dpp = 0 -> no hay error
dpp = 1 -> hay error

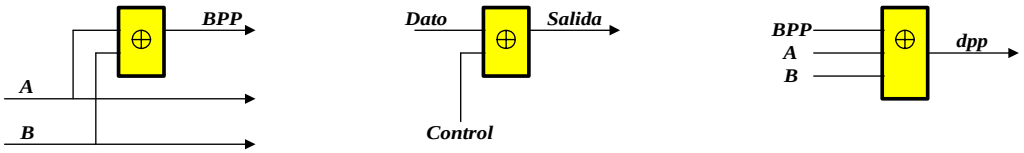

Figura 1. Utilización de la función XOR en los códigos de Hamming.

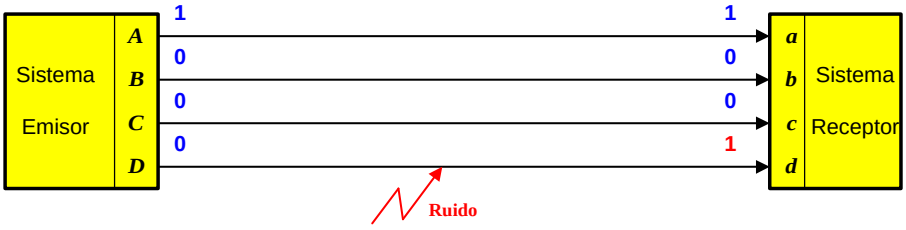

Figura 2. Planteamiento del problema de la transmisión de una palabra binaria.

A	B	C	D	E	Cantidad	BPP	A	B	C	D
0	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1
1	0	1	0	0	2	1	0	0	1	0
1	0	0	1	0	3	0	0	0	1	1
0	1	0	1	0	4	1	0	1	0	0
0	0	1	1	0	5	0	0	1	0	1
1	0	0	0	1	6	0	0	1	1	0
0	1	0	0	1	7	1	0	1	1	1
0	0	1	0	1	8	1	1	0	0	0
0	0	0	1	1	9	0	1	0	0	1

Tabla 1. Código de peso constante 2 entre 5 y código BCDnat con paridad constantemente par.

Los códigos de Hamming usan las puertas XOR para tres tareas diferentes: generador de paridad par, inversor programable y detector de paridad par (figura 1).

2. Códigos detectores de errores

Cuando se transmite una información binaria desde un emisor hacia un receptor a través de un medio susceptible a perturbaciones o ruidos externos, aparece el problema de que alguno de los bits de la palabra original puedan modificar su valor y den lugar a una nueva combinación que evidentemente será errónea (figura 2). En un código con distancia mínima 1, cuando se produce un error en un bit resulta una nueva combinación que también puede pertenecer al código. Ante esta situación el receptor no tiene ningún criterio para descartar la combinación como errónea. La condición necesaria para detectar una combinación errónea es que la distancia mínima del código sea 2, es decir, cuando se produzca un error en un bit de una palabra perteneciente al código, la nueva palabra resultante seguro que no pertenecerá al código y por este motivo se podrá descartar como errónea. Esto implica que no se pueden utilizar las 2^n combinaciones formadas con los n bits del código.

Los códigos de distancia mínima 2 pueden ser de peso constante (por ejemplo el código de 2 unos entre 5 bits) o bien de paridad constante, por ejemplo códigos de paridad par [3].

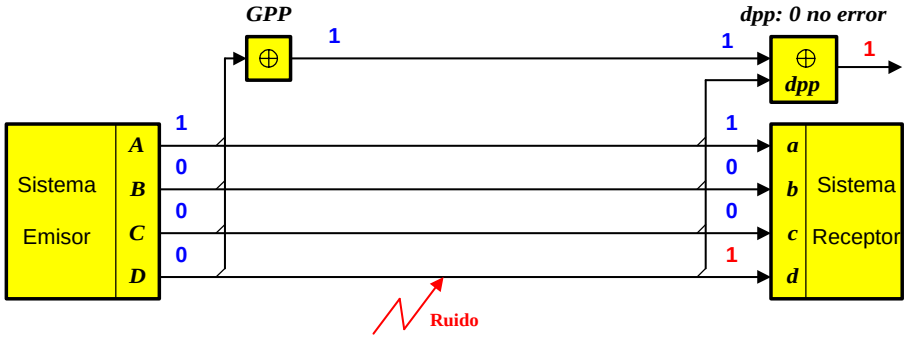

Figura 3a. Generador de bit de paridad par (GPP) y detector de paridad par (dpp).

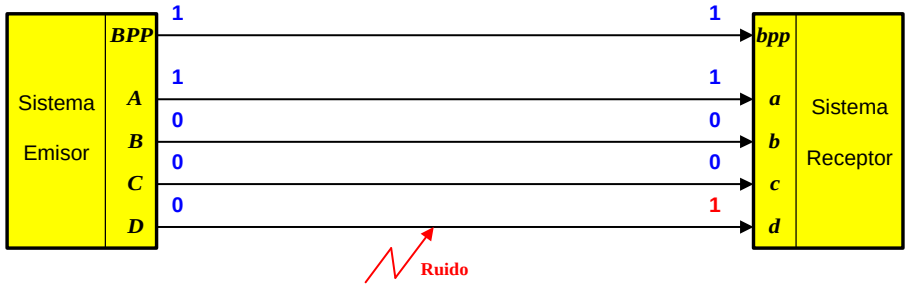

Figura 3b. Transmisión de datos con un código de paridad constantemente par.

bpp	a	b	c	d	Combinación
1	1	0	0	1	?
BPP	A	B	C	D	
1	1	0	0	**0**	8
1	1	0	**1**	1	11
1	1	**1**	0	1	13
1	**0**	0	0	1	1
0	1	0	0	1	9

Tabla 2. Combinaciones pertenecientes al código que son adyacentes a la combinación errónea recibida.

2.1. Limitaciones de los códigos detectores de errores

En las figuras 3a y 3b se observa la transmisión de una combinación binaria a la que se le ha añadido un bit de paridad par (*BPP*). Si el código original es el binario natural de cuatro bits, el nuevo código formado en el sistema emisor será de 5 bits, y la recepción de una palabra con un número de bits impar indica al receptor que es errónea.

Para que el sistema receptor detecte el bit erróneo hay que analizar la palabra recibida y ver cuál es la palabra que pertenece al código original que se ha transmitido. El problema es que hay tantas posibles combinaciones correctas como bits tiene la palabra recibida. En la tabla 2 se pone de manifiesto esta situación, al comprobar que es imposible detectar el bit erróneo en códigos de distancia mínima 2 que cojan todas las combinaciones posibles (2^{n-1}).

Los códigos de paridad constante son capaces de detectar una combinación errónea cuando se ha producido un cambio en un número impar de líneas, sin embargo, cuando cambian su estado un número par de líneas, la combinación recibida es considerada correcta sin serlo.

3. Códigos correctores de errores

La idea básica de los códigos correctores de errores es enviar dos veces la información de cada bit y comparar en la recepción que los bits recibidos por cada uno de los dos caminos es la misma. En caso de ser diferente, se puede afirmar que se ha producido un error en esa línea de datos.

Para corregir un error binario es suficiente con la inversión lógica del valor recibido por la línea de datos errónea. En la figura 4 (b) no hay suficiente información para saber cual de los dos hilos ha sufrido el cambio de valor, por este motivo, el segundo envío de la información se realizará codificado a través de bits de paridad para que la línea de datos sea única. En general, la distancia mínima de un código para que permita corregir errores en X líneas de datos ha de ser:

$$d_m = 2 \cdot X + 1 \tag{1}$$

En particular, para corregir un error la distancia mínima necesaria ha de ser 3. Si la distancia mínima fuera de 2, las circunferencias de la figura 5 serían tangentes y una combinación errónea sería adyacente a dos combinaciones válidas. Al disponer de distancia mínima 3, se garantiza que cualquier combinación errónea sea adyacente solamente a una combinación válida [4].

Se define:

- n → número de bits del código original que se pretende transmitir.
- p → número de bits de paridad par generados en el transmisor, o sea, número de líneas que añadimos al código inicial.
- c → número de bits detectores-correctores de paridad par generados por el receptor.

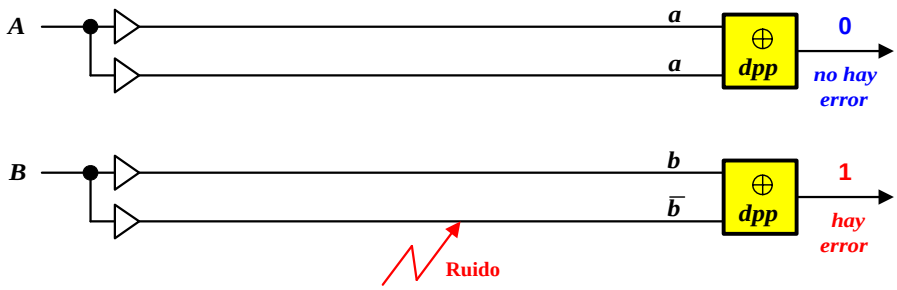

Figura 4. Idea básica para detectar un error en un bit concreto.

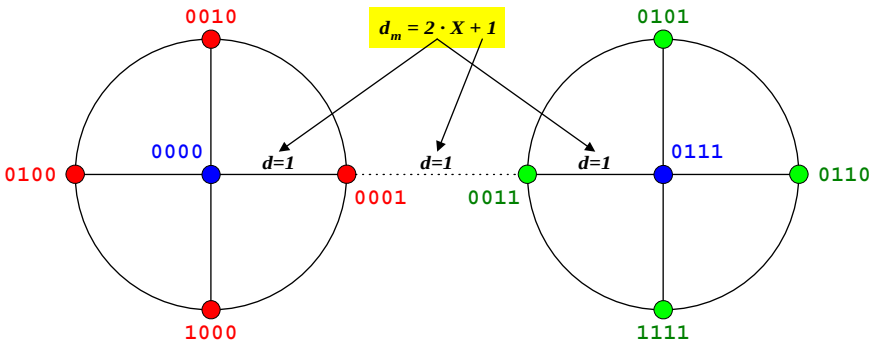

Figura 5. Justificación gráfica de la necesidad de códigos con distancia mínima 3 para corregir un error.

Numéricamente c debe ser igual a p para que cada uno de los bits detectores de paridad par estén vinculados a una sola línea de bit de paridad par. Así, también se deberá cumplir que cada p_i solo esté contenido una vez en alguno de los detectores de paridad, y más concretamente lo estará en su correspondiente c_i .

$$c \equiv p \tag{2}$$

El propósito del algoritmo de Hamming es realizar una tabla detectora-correctora del bit erróneo a partir de los bits detectores de paridad par generados por el receptor, es decir, poder identificar la línea donde se ha producido el error y así proceder a su corrección (tabla 3).

El número de combinaciones que se pueden formar con los c bits tiene que ser mayor o igual que el número de líneas del código original (n) más el número de líneas de paridad añadidas (p) más uno, este último para contemplar el caso de no error.

$$2^c \geq n + p + 1 \tag{3}$$

La asignación de una combinación de las 2^c posibles a una línea física o a un bit en concreto no es aleatoria. Las combinaciones se clasifican en tres grupos bien diferenciados:

1. Combinación asignada a la situación en que no haya error en la transmisión.
2. Combinaciones asignadas a los bits de paridad generados en el transmisor.
3. Combinaciones asignadas a los bits de datos del código original.

A continuación se propone un ejemplo concreto: desarrollar el algoritmo de Hamming para corregir un error de un bit en la transmisión de palabras de 7 bits en código ASCII. Atendiendo a la fórmula anteriór:

Con $c = 3$ se pueden tratar códigos de hasta $n = 4$ bits.
Con $c = 4$ se pueden tratar códigos de hasta $n = 11$ bits.

Por lo tanto, en el ejemplo propuesto se añadirán 4 bits de paridad par en el origen de la transmisión (p =4).

Se pretende que el minterm $c_4\, c_3\, c_2\, c_1$ indique la línea física o el bit donde se ha producido el error. Para ello hay que responder a la siguiente pregunta: ¿Cuándo será uno el bit c_i?. En general, un bit c_i será 1 siempre que la combinación donde se ha producido el error tenga un uno en dicho c_i. El bit c_4 será uno cuando haya un error en las combinaciones comprendidas entre b_8 a b_{15}, ambas incluidas. Para construir la ecuación booleana bastará con introducir en la función detectora de paridad par (XOR) todos los bits que tengan un uno en la columna de c_i. En el caso de que haya error en una combinación que tenga un cero en la columna c_i, este cero se conseguirá precisamente no incluyendo la combinación b_i en la función XOR.

$$\begin{aligned} c_1 &= b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \oplus b_{13} \oplus b_{15}; \\ c_2 &= b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} \oplus b_{14} \oplus b_{15}; \\ c_3 &= b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15}; \\ c_4 &= b_8 \oplus b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15}; \end{aligned} \tag{4}$$

Como ya se ha mencionado al principio de este punto, la idea básica de los códigos correctores de errores es enviar dos veces la información de cada línea b_i. Para conseguir este objetivo hay que codificar la información a través de los bits de paridad par, y estos bits se deben elegir para que no se produzcan interacciones entre los distintos detectores de pararidad par c_i. La única forma de cumplir con todos estos requisitos es utilizar las combinaciones b_1, b_2, b_4 y b_8 como bits de paridad puesto que solamente aparecen una vez en sus respectivas c_i, es decir, solamente tienen un uno en su minterm.

Combinación	N°unos	C4	C3	C2	C1	Asignación
b0	0	0	0	0	0	situación de "no error"
b1	1	0	0	0	1	bit de paridad par "P1"
b2	1	0	0	1	0	bit de paridad par "P2"
b3	2	0	0	1	1	bit de datos G "D0"
b4	1	0	1	0	0	bit de paridad par "P3"
b5	2	0	1	0	1	bit de datos F "D1"
b6	2	0	1	1	0	bit de datos E "D2"
b7	3	0	1	1	1	bit de datos D "D3"
b8	1	1	0	0	0	bit de paridad par "P4"
b9	2	1	0	0	1	bit de datos C "D4"
b10	2	1	0	1	0	bit de datos B "D5"
b11	3	1	0	1	1	- NO USADA -
b12	2	1	1	0	0	bit de datos A "D6"
b13	3	1	1	0	1	- NO USADA -
b14	3	1	1	1	0	- NO USADA -
b15	4	1	1	1	1	- NO USADA -

Tabla 3. Tabla detectora-correctora del bit erróneo.

$$
\begin{aligned}
b_1 &= b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \oplus b_{13} \oplus b_{15}; \\
b_2 &= b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} \oplus b_{14} \oplus b_{15}; \\
b_4 &= b_5 \oplus b_6 \oplus b_7 \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15}; \\
b_8 &= b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15};
\end{aligned}
\tag{5}
$$

Se observa que si se sustituyen las ecuaciones de los bits de paridad b_1, b_2, b_4 y b_8 en sus respectivas funciones c_i, queda duplicada al información de todos los b_i. Como la función XOR de una variable repetida dos veces es cero ($A \oplus A = 0$), cuando no se produzca ningún error aparecerá el minterm cero en las variables $c_4\, c_3\, c_2\, c_1$, por este motivo al mintem cero se le asigna a la situación de "no error" en la tabla 3.

Para finalizar, las combinaciones asignadas a datos serán aquellas que tengan más de un uno, pero, preferiblemente el menor número de unos para que las funciones c_i sean más simples. Las combinaciones b_3 , b_5 , b_6 , b_9 , b_{11} y b_{12} tienen 2 unos en sus minterms. Por lo tanto, hay que coger una combinación de 3 unos para completar los 7 bits de datos que utiliza el código ASCII.

Siguiendo con el ejemplo concreto, las ecuaciones resultantes quedarían simplificadas en relación con las ecuaciones generales.

$$
\begin{aligned}
c_1 &= b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9; \\
c_2 &= b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10}; \\
c_3 &= b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{12}; \\
c_4 &= b_8 \oplus b_9 \oplus b_{10} \oplus b_{12};
\end{aligned}
\tag{6}
$$

Y los bits generados en el origen serían:

$$
\begin{aligned}
b_1 &= P_1 = b_3 \oplus b_5 \oplus b_7 \oplus b_9; \\
b_2 &= P_2 = b_3 \oplus b_6 \oplus b_7 \oplus b_{10}; \\
b_4 &= P_3 = b_5 \oplus b_6 \oplus b_7 \oplus b_{12}; \\
b_8 &= P_4 = b_9 \oplus b_{10} \oplus b_{12};
\end{aligned}
\tag{7}
$$

La asignación de los bits de paridad P_i dentro del grupo de combinaciones que tiene un solo uno puede ser aleatoria. De la misma manera, la asignación de los datos D_i dentro del grupo de las 7 combinaciones b_3 , b_5 , b_6 , b_9 , b_{11} , b_{12} y b_7 puede ser también aleatoria. En este ejemplo se ha seguido un criterio de subíndices crecientes en ambos casos.

Se puede comprobar fácilmente que si hay un error en la linea de datos D_5, el conjunto de las cuatro variables c_i indicarían el minterm 10. A c_2 y a c_4 llegaría b_{10} a través de los bits de paridad b_2 y b_8 respectivamente y simultáneamente a ambos llegaría $/b_{10}$ a través del hilo físico que ha sufrido el error.

$$c_1 = b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 = 0 \quad \text{porque no interviene } b_{10}.$$
$$c_2 = b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus /b_{10} = 1 \quad \text{porque } b_{10} \oplus /b_{10} = 1.$$
$$c_3 = b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{11} = 0 \quad \text{porque no interviene } b_{10}.$$
$$c_4 = b_8 \oplus b_9 \oplus /b_{10} \oplus b_{12} = 1 \quad \text{porque } b_{10} \oplus /b_{10} = 1.$$

Una vez decodificada esta información ($Y_{10} = 1$) se actuaría con un inversor programable sobre la línea de datos que ha sufrido el error, es decir, la b_{10} y el receptor leería el dato correcto que le había enviado en transmisor, $D_{6\text{-}0}$. En la figura 6 se muestra el esquema completo que permite la corrección de un error en palabras de 7 bits.

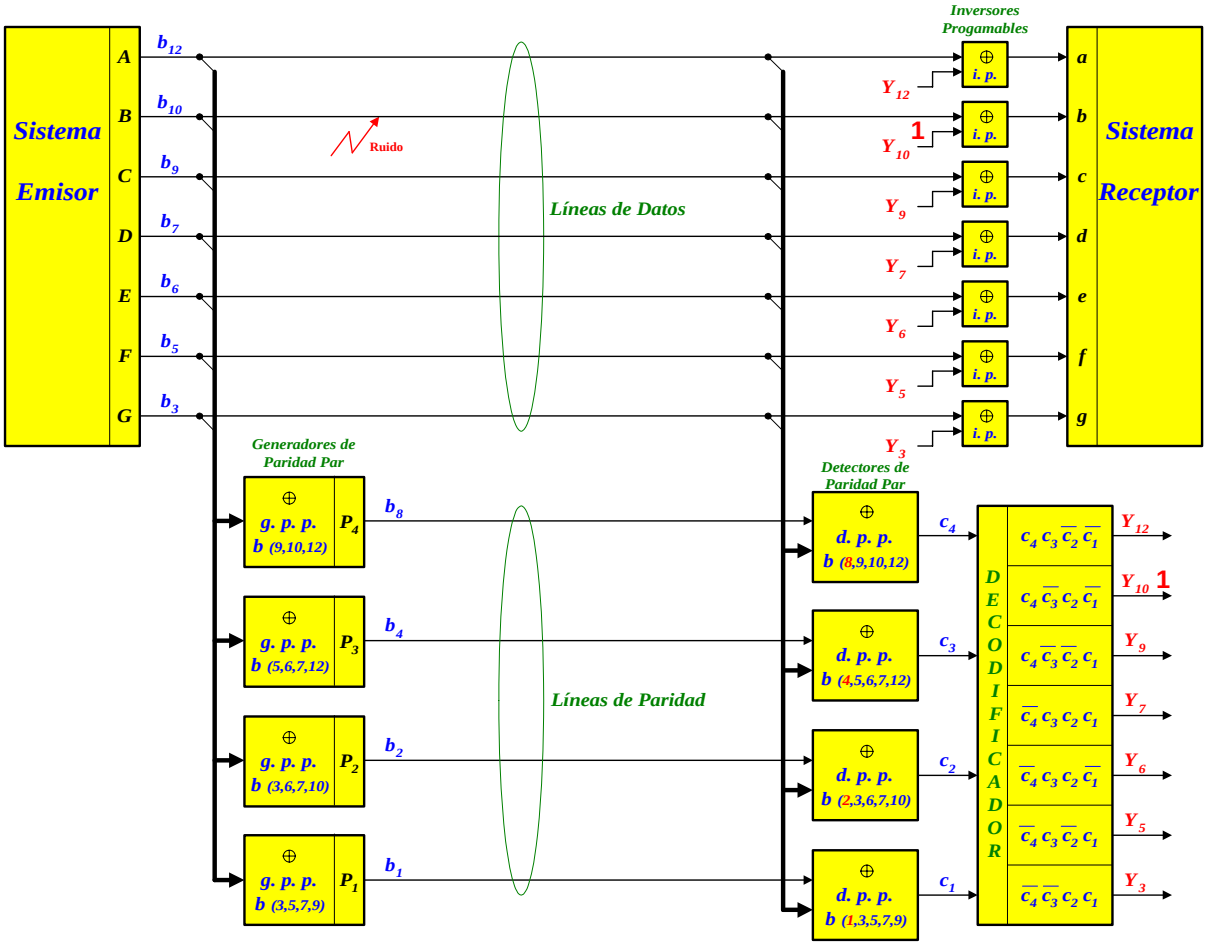

Figura 6 Esquema general del circuito de Hamming (11,7).

El circuito de Hamming también dispone de información suficiente para corregir un error en las líneas de paridad par añadidas en el origen, sin embargo, no se realiza esta acción porque únicamente interesa asegurar que llegan correctamente las n líneas de datos del código inicial.

Supongamos que se ha producido un error en el bit de paridad b_8 , entonces aparecerá el minterm ocho en los bits detectores de paridad par del receptor.

$$c_1 = b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 = 0 \quad \text{porque no interviene } b_8.$$
$$c_2 = b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10} = 0 \quad \text{porque no interviene } b_8.$$
$$c_3 = b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{11} = 0 \quad \text{porque no interviene } b_8.$$
$$c_4 = /b_8 \oplus (b_9 \oplus b_{10} \oplus b_{12}) = 1 \quad \text{porque } /b_8 \oplus b_8 = 1.$$

El método descrito en el ejemplo funciona para cualquier código se 7 bits independientemente de que sea el ASCII, el binario natural o cualquier otro código de 7 bits. La conclusión es que una vez desarrollado el algoritmo de Hamming para corregir un error en palabras de n bits, el sistema funciona independientemente del código que se le introduzca. Por este motivo se puede hablar de "códigos de Hamming" en plural.

Los nuevos códigos generados al añadir al código inicial los bits de paridad, tienen $n+p$ bits y sus palabras se completan aplicando las ecuaciones P_i. Por ejemplo, la palabra $X_{10\text{-}0}$ pertenecerá a un nuevo código que garantiza una distancia mínima de 3.

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
			D6		D5	D4	P4	D3	D2	D1	P3	D0	P2	P1	
			X10		X9	X8	X7	X6	X5	X4	X3	X2	X1	X0	

INPUT DATA

DATA BLOCK

			INPUT DATA								RESULTING FEC			
			7	6	5	4	3	2	1	0	3	2	1	0
loaded first -	Byte	0	0	1	2	3	4	5	6	7	8	9	10	11
	Byte	1	12	13	14	15	16	17	18	19	20	21	22	23
	Byte	2	24	25	26	27	28	29	30	31	32	33	34	35
	Byte	3	36	37	38	39	40	41	42	43	44	45	46	47
	Byte	4	48	49	50	51	52	53	54	55	56	57	58	59
	Byte	5	60	61	62	63	64	65	66	67	68	69	70	71
	Byte	6	72	73	74	75	76	77	78	79	80	81	82	83
	Byte	7	84	85	86	87	88	89	90	91	92	93	94	95
	Byte	8	96	97	98	99	100	101	102	103	104	105	106	107
	Byte	9	108	109	110	111	112	113	114	115	116	117	118	119
	Byte	10	120	121	122	123	124	125	126	127	128	129	130	131
	Byte	11	132	133	134	135	136	137	138	139	140	141	142	143
	Byte	12	144	145	146	147	148	149	150	151	152	153	154	155
	Byte	13	156	157	158	159	160	161	162	163	164	165	166	167
	Byte	14	168	169	170	171	172	173	174	175	176	177	178	179
	Byte	15	180	181	182	183	184	185	186	187	188	189	190	191
	Byte	16	192	193	194	195	196	197	198	199	200	201	202	203
loaded last -	Byte	17	204	205	206	207	208	209	210	211	212	213	214	215
CRC	Byte	18	216	217	218	219	220	221	222	223	224	225	226	227
CRC	Byte	19	228	229	230	231	232	233	234	235	236	237	238	239

Figura 7. Generación de una trama de 20 palabras.

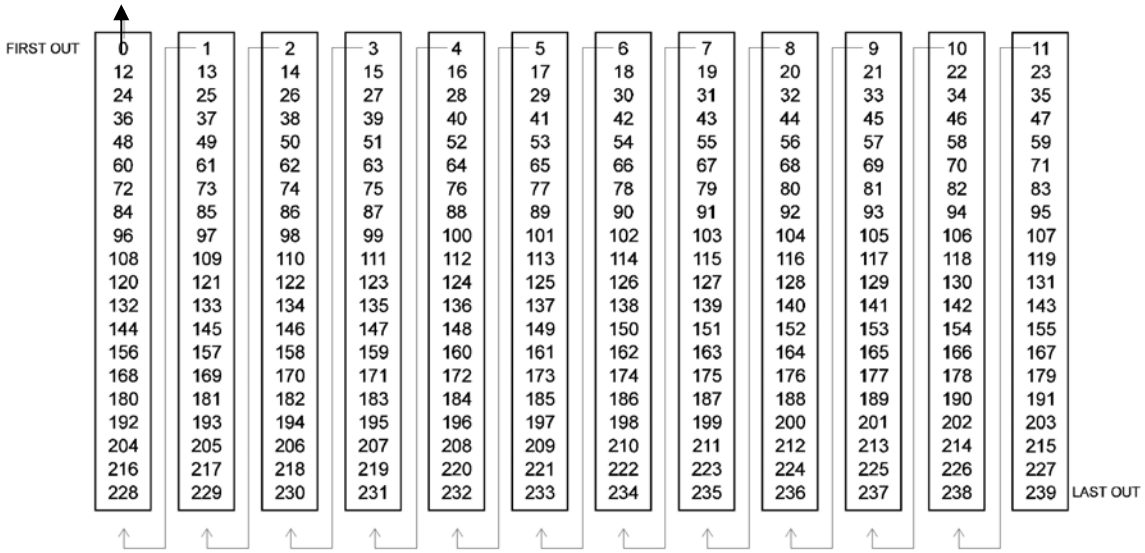

Figura 8. Detalle de la secuencia utilizada para enviar los bits a través de un canal serie.

4. Ejemplos de aplicación de los códigos correctores de errores

En el año 1987 Intel introdujo el circuito integrado 8206 cuya misión era detectar y corregir errores en palabras de 8 o de 16 bits [6]. Inicialmente su campo de aplicación era corregir los posibles errores generados al leer datos desde una memoria. Recientemente, los códigos de Hamming han visto una aplicación similar al corregir errores en bits almacenados en memorias RAM dinámicas insertadas dentro de circuitos integrados de aplicación específica ASICs [7].

En el año 1996, la firma CML introdujo el circuito integrado FX909A [8]. Se trata de un modem para comunicaciones inalámbricas vía radio que incorpora la modulación GMSK. Es importante destacar que para aprovechar al máximo las posibilidades de los códigos correctores de un error, el registro de desplazamiento que incorpora ordena los bits transversalmente (figuras 7 y 8), es decir, si durante la transmisión hay un ruido de unos milisegundos, se estropearán varios bits pero todos de palabras distintas y por lo tanto todas estas palabras se podrán reconstruir. En cambio, si los bits se hubieran enviado palabra tras palabra, la modificación de varios bits consecutivos impedirían la reconstrucción de la información de dicha palabra.

5. Simulación del algoritmo de Hamming

En una primera aproximación se le ofrece al alumno el circuito clásico de Hamming (7,4) con cuatro bits de datos y tres de paridad. Se ha elegido el programa CircuitMaker Student V6.2c por su facilidad de uso e interactividad. La licencia permite al alumno simular circuitos de hasta 50 componentes en su ordenador personal. Como se aprecia en la figura 9 el interfaz es gráfico y permite cambiar el estado de los interruptores de entrada en plena simulación. Las líneas aparecen en color azul cuando su nivel lógico es cero y en color rojo cuando el nivel lógico es uno.

Para circuitos de Hamming mayores esta versión de CircuitMaker resulta insuficiente por su limitación en cuanto a número de componentes. Aprovechando las experiencias de otros trabajos publicados en el TAEE'06 sobre aplicaciones de software libre [9, 10], se ha optado por la actual versión de GNU Octave 3.0.0 [11]. La programación es similar a Matlab y los algoritmos funcionan de manera estable. Sin embargo, la representación gráfica de los resultados no es muy estable dado que la función "*imshow(matriz, mapa_color)*" no responde siempre con el mismo color aún cuando se mantenga el mismo número en un determinado elemento de la matriz.

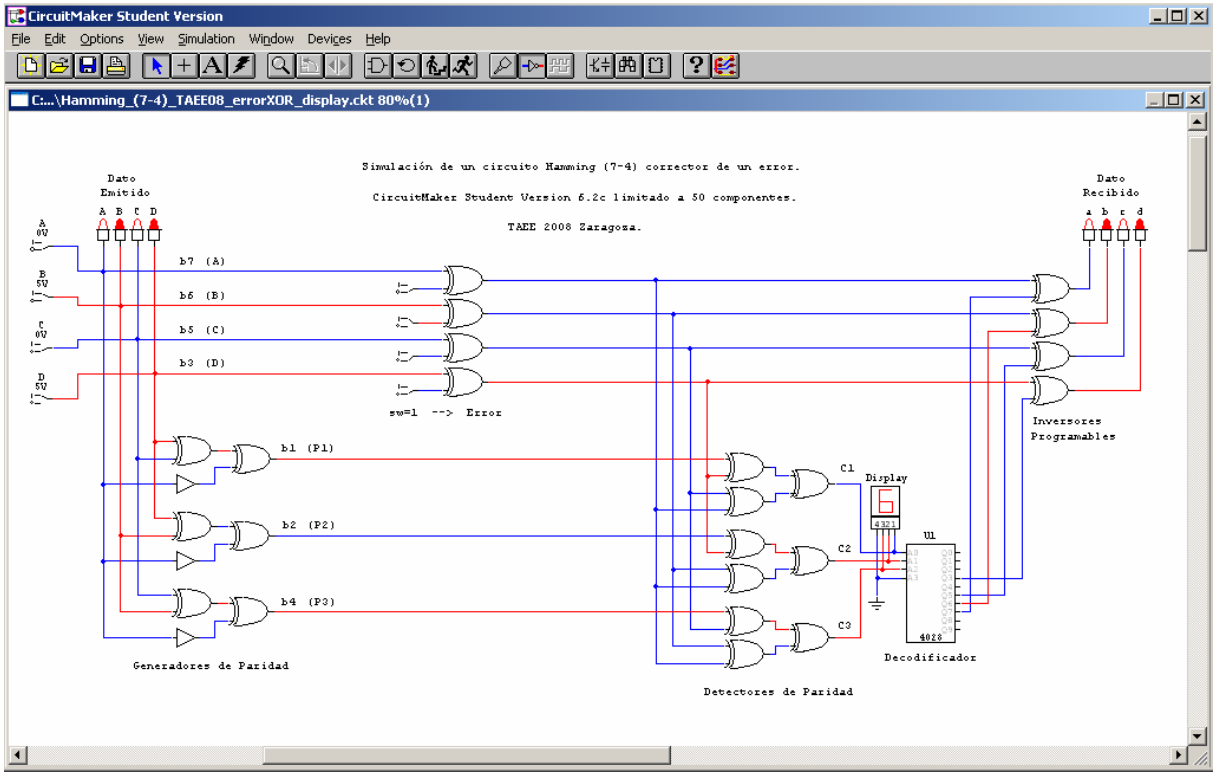

Figura 9. Circuito de Hamming (7,4) en la ventana del programa CircuitMaker Student V6.2c

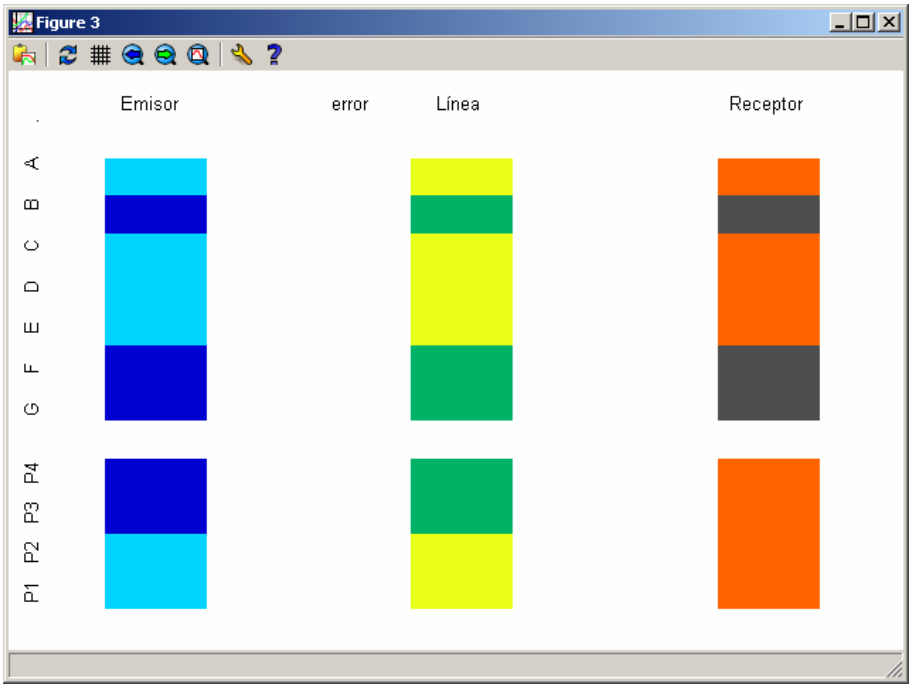

Figura 10. Circuito de Hamming (11,7). Sin ningún bit erróneo. Octave 3.0.0.

En las figuras 10, 11 y 12 se muestra el resultado de la simulación de un código de Hamming (11,7) con 7 bits de datos y 4 bits de paridad. El alumno puede modificar el dato binario de entrada y el bit donde se produce el error durante la transmisión. El programa calcula los bits de paridad el origen, los bits en la línea después del error y los bits definitivos que entran en el receptor.

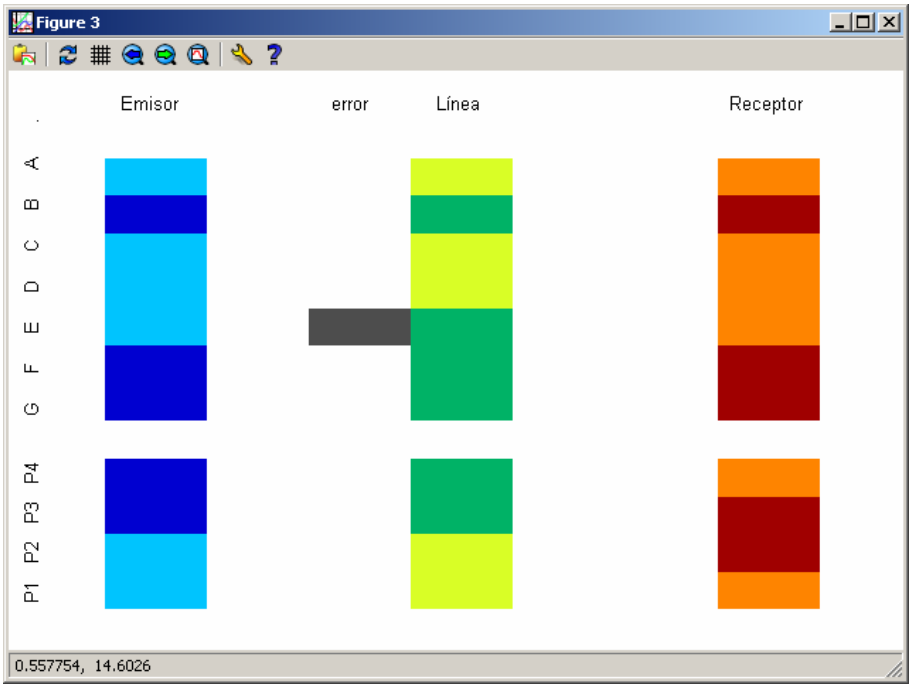

Figura 11. Circuito de Hamming (11,7). Con error en el bit de datos b_6. Octave 3.0.0.

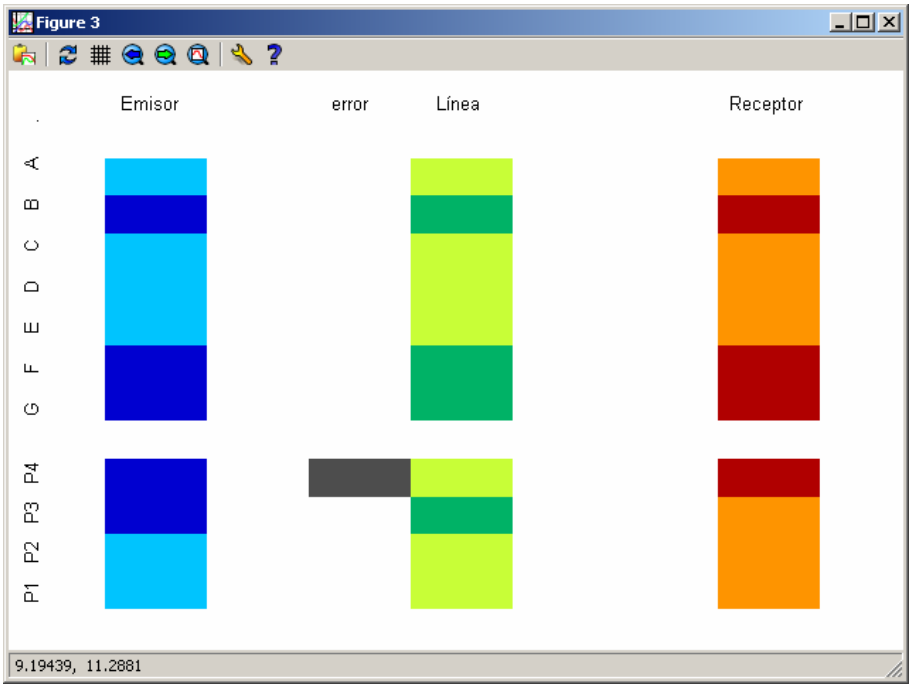

Figura 12. Circuito de Hamming (11,7). Con error en el bit de paridad b_8. Octave 3.0.0.

Se observa que el algoritmo funciona correctamente en los tres casos posibles:

1) Sin error en la transmisión (figura 10).
2) Con un error en un bit de datos (figura 11).
3) Con un error en un bit de paridad (figura 12).

6. Conclusiones

En la mayor parte de los programas docentes de electrónica digital básica, bien sea por falta de tiempo o por la gran extensión de las materias a estudiar, los códigos de Hamming rara vez se abordan. Pese a la gran cantidad de información que hay en internet sobre el tema, este trabajo pretende aportar un grano de arena más para que los alumnos puedan estudiar por sí mismos los conceptos de detección y corrección de errores y amplíen su visión sobre la electrónica digital.

La experiencia docente de los autores durante los últimos 10 años confirma que con el planteamiento de los códigos de Hamming que se hace en este trabajo, los alumnos están muy motivados para abordar este complejo tema y en consecuencia, alcanzan fácilmente un grado de comprensión adecuado.

Referencias

[1] http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Hamming.html.

[2] Richard W. Hamming; "Error detecting and error correcting codes"; The Bell System Technical Journal; Vol. XXVI, No. 2, pp. 147-160, April, 1950.

[3] Enrique Mandado; "Sistemas electrónicos digitales"; Editorial Marcombo; 5ª edición; 1984.

[4] John F. Wakerly; "Diseño digital. Principios y prácticas"; Editorial Prentice Hall, 3ª edición, 2001.

[5] Lisa Anneberg and Ece Yaprak; "Error detection and correction templates for digital courses"; IEEE Transactions on Education, Vol. 42, No. 2, pp.114-117, May, 1999.

[6] 8206 Error detection and correction unit; Intel; September, 1987.

[7] Ken Gray; "Adding error-correcting circuit to ASIC memory"; IEEE Spectrum, pp. 55-60, April, 2000.

[8] FX909A Wireless modem data pump; CML Semiconductor Products; March, 1996.

[9] C. Medrano, J.M. Valiente, I. Plaza y P. Ramos; "Evaluación de herramientas de software libre para cálculo numérico"; TAEE 2006, Madrid, 12-14 de Julio de 2006.

[10] J. Vicente, B. García, I. Ruiz, A. Méndez y O. Lage; "Nueva metodología de enseñanza de procesado digital de la señal utilizando la API "joPAS""; TAEE 2006, Madrid, 12-14 de Julio de 2006.

[11] John W. Eaton; Software GNU Octave; www.octave.org