UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Introducción a los Códigos de Corrección de Errores

Apellidos, nombre	Gracia Morán, Joaquín[1] (jgracia @ disca.upv.es) Saiz Adalid, Luis José[1] (ljsaiz @ disca.upv.es)
Departamento	[1]Departamento de Informática de Sistemas y Computadores (DISCA)
Centro	Escuela Técnica Superior de Ingeniería Industrial Universitat Politècnica de València

1 Resumen de las ideas clave

En este trabajo vamos a presentar los conceptos básicos y el funcionamiento de los Códigos de Detección y Corrección de Errores. Esta técnica de Tolerancia a Fallos permite detectar y/o corregir errores en diferentes ámbitos de aplicación, como pueden ser en la transmisión de información (buses, unidades de Entrada/Salida y transmisión de datos), o en el almacenamiento de datos (RAM, discos duros, etc.).

La utilidad de este artículo consiste en concretar la obtención de las fórmulas de codificación y decodificación de los datos a partir de la representación de un código mediante su matriz de paridad, así como su uso a la hora de detectar y/o corregir fallos en la información. En cualquier caso, este trabajo se centra en códigos de Detección y Corrección de Errores para datos no muy grandes (desde 8 hasta 128 bits), dejando fuera otros códigos como CRC, Reed-Solomon, etc., diseñados para palabras de datos mayores.

La estructura de este artículo es la siguiente:

1. Objetivos de este trabajo.
2. Una introducción en la que describiremos brevemente la problemática a la hora de transmitir y/o almacenar información en los sistemas informáticos actuales y una posible solución.
3. Descripción de diferentes tipos de Códigos de Detección y Corrección de Errores.
4. Cierre de este trabajo

2 Objetivos

Una vez que os leáis este documento, seréis capaces de:

- Obtener, a partir de la matriz de paridad **H**, las ecuaciones para codificar y decodificar datos en función de un determinado Código de Corrección de Errores.
- Comprobar si una palabra de datos es correcta según un Código de Corrección de Errores.

3 Introducción

Los Códigos de Detección y Corrección de Errores (Códigos de Corrección de Errores para abreviar) permiten mitigar los efectos de diferentes tipos de fallos que se pueden producir tanto en la transmisión de información como en su almacenamiento. En la actualidad, se ha propuesto su uso incluso en la estructura interna de los procesadores (en memorias cache, registros, buses, etc.).

3.1 Redundancia en la información

Durante una transmisión de datos, estos pueden sufrir errores. Una transmisión puede ser temporal (como almacenar la información en un determinado soporte para recuperarla

posteriormente) o espacial (el envío de información entre dos puntos distantes en el espacio).

Esta posibilidad de que se produzcan errores provoca que en determinados entornos se deban proteger los datos. Para conseguir esta protección, es necesario añadir información a los datos para poder detectar, enmascarar e incluso corregir errores en los mismos. Esta acción se denomina **Redundancia en la Información**. La información extra que se añade a los datos, denominada bits de código, bits de paridad o bits redundantes, debe seguir unas reglas claras para poder ser utilizada en la detección y/o corrección de errores. Así pues, un *código* es un medio para representar datos usando un conjunto de reglas bien definidas, mientras que una *palabra codificada* es un elemento del código que satisface las reglas de codificación establecidas.

La codificación de un dato produce una palabra codificada, y la decodificación de una palabra codificada genera un dato, tal y como se puede ver en la Figura 1. Cuando el transmisor genera un dato, este se codifica antes de enviarlo por el canal de transmisión. En este canal, puede haber ruido, lo que provoca que la palabra codificada pueda ser modificada, por lo que, al decodificarla, los datos que reciba el receptor no serán los mismos que los datos enviados por el transmisor.

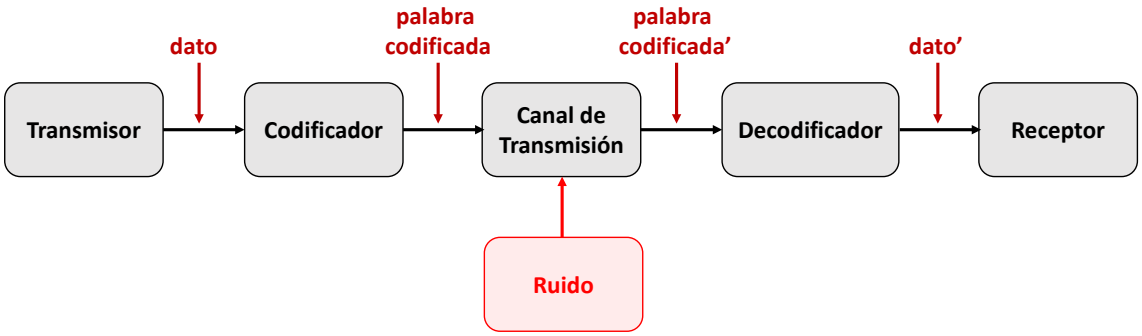

Figura 1. Proceso de codificación y decodificación de la información.

Dentro del conjunto de las posibles palabras del código, las palabras codificadas conforman un subconjunto, de tal forma que la distinción entre palabras codificadas y palabras no válidas posibilita la detección y/o la corrección de errores en los datos. Así pues, si lo que queremos es detectar errores, el código deberá definirse de tal forma que permita revelar los cambios producidos en las palabras codificadas y que las transforman en palabras posibles no válidas. Por otra parte, si lo que queremos es corregir errores, primero hay que detectarlos, y después, hay que identificar qué bit (o bits) son los que han sido modificados. Así pues, el código deberá definirse de tal forma que, a partir de una palabra posible no válida producida por un error, se pueda deducir la palabra codificada original (palabra sin fallo).

3.2 Tipos comunes de errores en la información

Antes de continuar, conviene conocer los tipos de errores comunes que se pueden producir en la información. Los errores pueden ser *simples* o *múltiples*. Un *error simple* es aquel que afecta a un único bit. Puede ser de dos tipos: i) *bit-flip* cuando el problema es transitorio. Su efecto es cambiar el valor de un determinado bit (de '0' a '1', o de '1' a '0'); o ii) *Stuck-at* cuando el problema es permanente. En este caso el error fija el valor de un bit a '0' o a '1'.

Por otra parte, un *error múltiple* es aquel que afecta a varios bits. Pueden ser *adyacentes* o *de ráfaga*. Un error múltiple adyacente es aquel que se produce en bits contiguos, y están causados, por ejemplo, por un cortocircuito entre líneas o por perturbaciones externas. Un error de ráfaga es un error múltiple que afecta a varios bits contiguos donde, al menos, el primer y el último bit son erróneos. En la Figura 2 se pueden ver varios ejemplos de errores simples y múltiples adyacentes y de ráfaga (los errores están marcados en rojo).

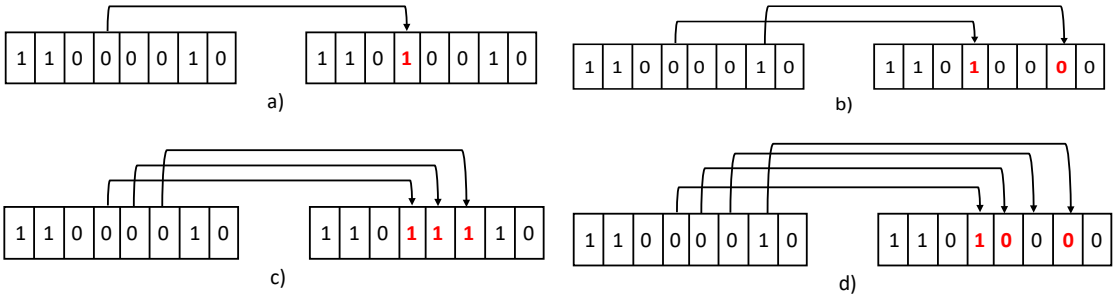

Figura 2. Ejemplos de errores en la información: a) error simple; b) error doble no adyacente; c) error triple adyacente; d) error de ráfaga de 4 bits.

4 Códigos Detectores y Correctores de Errores

Los Códigos Detectores de Errores tienen la capacidad de detectar cuando un dato no es correcto (no representa una palabra codificada válida), mientras que los Códigos Correctores de Errores pueden reconstruir palabras codificadas válidas a partir de palabras con errores. Como se ha comentado antes, la codificación de la información con fines de detección y corrección de errores comporta siempre un aumento en el número de bits de la información (Redundancia de la Información).

4.1 Distancia de Hamming

Esta métrica proporciona una medida de las capacidades de detección y corrección de un código. Hace referencia al número de bits en el que difieren dos palabras (posición a posición). En la Figura 3 podemos ver unos ejemplos. En la parte izquierda, la distancia de Hamming entre las dos palabras mostradas es uno, mientras que, en la derecha, la distancia de Hamming es dos.

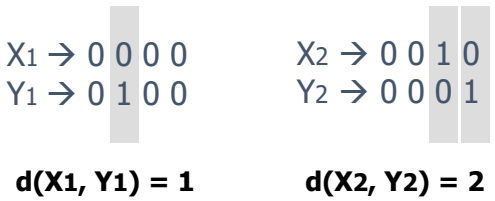

Figura 3. Ejemplos de cálculos de la distancia de Hamming.

La **distancia de Hamming de un código** es la distancia de Hamming mínima entre dos palabras codificadas válidas distintas. De esta forma, en un código con distancia de Hamming igual a dos, cualquier alteración de un bit produce una palabra inválida en el código, por lo que el error se puede detectar. Es decir, los códigos con distancia de Hamming igual a dos

permiten detectar errores de un bit en los datos. Para corregir el error, hay que transformar la palabra inválida en la palabra válida más próxima. Esto sólo es posible si no hay dos o más palabras a la misma distancia de Hamming.

Veamos un ejemplo más detallado. Sea el código de paridad par para palabras de dos bits. Al tener 3 bits (dos bits de datos más un bit de paridad), tendremos ocho posibles palabras, de las cuales sólo cuatro serán válidas, tal y como se muestra en la Tabla 1. Como se puede ver, la distancia mínima de Hamming entre palabras válidas es de dos, por lo que, si falla un bit, la palabra resultante no es válida, es decir, podemos detectar el error.

Tabla 1. Palabras válidas y no válidas del código de bit de paridad par.

PALABRAS VÁLIDAS		PALABRAS NO VÁLIDAS	
Bits de Datos	**Bit de Paridad**	**Bits de Datos**	**Bit de Paridad**
0 0	0	0 0	1
0 1	1	0 1	0
1 0	1	1 0	0
1 1	0	1 1	1

Veamos otro ejemplo. Sea ahora un código en el que las únicas palabras válidas son {000 y 111}, tal y como se muestra en la Tabla 2. En este caso, la distancia de Hamming del código es igual a tres. Esto significa que, si falla un bit, se puede corregir, sustituyendo la palabra errónea por la palabra codificada más cercana (con menor distancia de Hamming, que además, resulta ser la palabra correcta). Si fallan dos bits, la palabra más cercana según la distancia de Hamming no es válida, es decir, podemos detectar el error, pero no corregirlo.

En este ejemplo, es importante remarcar que no se pueden hacer ambas cosas al mismo tiempo. Es decir, o bien se corrigen errores simples, o bien se detectan errores simples y dobles. La decisión de hacer funcionar el código de una forma u otra depende de la hipótesis de fallo con la que se diseñe nuestro sistema. Si no se esperan que se produzcan errores dobles, optaremos por la corrección de errores simples. Pero si se produce un error doble, con ese funcionamiento se decodificaría erróneamente, por lo que habría que optar por la detección.

Tabla 2. Palabras válidas y no válidas del código {000, 111}.

PALABRAS VÁLIDAS	PALABRAS NO VÁLIDAS	
000	001	100
111	010	101
	011	110

En resumen, para que un código detecte errores de b bits se debe cumplir que $d \geq b+1$, siendo d su distancia Hamming. Por otra parte, para que un código corrija errores en b bits, se debe cumplir que $d \geq 2b+1$ (también siendo d su distancia Hamming). Y si queremos que un código tenga propiedades de corrección y detección, entonces $d \geq 2b_c + b_d + 1$, siendo d su distancia de Hamming, b_c los bits erróneos a corregir y b_d el número de bits erróneos adicionales a detectar.

4.2 Propiedades de los códigos

Veamos a continuación una serie de propiedades de los Códigos de Corrección de Errores:

- **Códigos separables.** Se dice que un código es separable cuando los bits de datos originales y los bits de código están separados y son fácilmente identificables.
- **Redundancia de un código.** Es el número de bits añadidos respecto al número de bits originales de datos: $Redundancia = \frac{Bits\ de\ código}{Bits\ de\ datos}$
- **Cobertura de un código.** Hace referencia a los errores cubiertos respecto a los errores totales considerados en la hipótesis de diseño. Ejemplo: un código de 3 bits de paridad par cubre el 100% de los errores de 1 bit (hipótesis 1) y el 0 % de los errores de 2 bits (hipótesis 2).

4.3 Códigos SEC de Hamming

Estos códigos de corrección de errores han sido (y son) ampliamente utilizados en los sistemas informáticos. Se usan principalmente en las memorias de los computadores, donde se concentra entre el 60% y el 70% de los fallos transitorios. Los códigos SEC de Hamming presentan una baja redundancia, una rápida detección y corrección de errores, y sus circuitos suelen ser sencillos.

En concreto, el código SEC (del inglés *Single Error Correction*) de Hamming es un código separable capaz de corregir errores de un bit (tiene una distancia de Hamming igual o superior a tres). Se basa en la idea de las paridades solapadas. Cada bit de código se genera calculando la paridad de un subconjunto de bits de datos (distinto para cada bit de código) de tal forma que todos los bits de datos quedan cubiertos por varios bits de código.

Veamos a continuación un ejemplo del proceso de generación, diseño de codificadores y decodificadores, y funcionamiento de un código SEC de Hamming. En concreto, del código SEC H(7, 4), siendo 7 el número de bits de la palabra codificada, y 4 el número de bits de datos. De esta forma, se puede ver fácilmente que el número de bits de código es 3.

Para obtener las fórmulas para la codificación de los datos, en primer lugar, se numeran los bits de la palabra codificada del 1 al 7 y se representan en binario. En este ejemplo concreto, solo son necesarios tres bits (del 001 al 111). Los bits cuya posición representada en binario tengan un solo '1' corresponden a bits de código, y los demás son los bits de datos. Cada bit de código se obtiene calculando la paridad del resto de bits cuya posición representada en binario tiene a '1' el mismo bit que el bit de código. Por ejemplo, el bit de código en la posición 001 se obtiene calculando la paridad de los bits de datos situados en las posiciones 011, 101 y 111, o el bit de código en la posición 010 se obtiene con los bits 011, 110 y 111. Puedes ver este ejemplo de obtención de las fórmulas del codificador en la Tabla 3, donde C_i son los bits de código, y D_i los bits de datos. Un aspecto a tener en cuenta es que C_0 es el bit menos significativo (LSB), mientras que C_2 es el bit más significativo (MSB).

Podemos expresar este código de forma matricial, representando en columnas cada uno de los valores, de forma que en cada fila quedan los bits del mismo peso, obteniendo la matriz de paridad **H**:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Tabla 3. Diseño codificador SEC Hamming (7, 4).

Numeración bits palabra codificada en binario							Fórmulas codificación
001	**010**	**011**	**100**	**101**	**110**	**111**	
C_0		**D_0**		**D_1**		**D_3**	$C_0 = D_0 \oplus D_1 \oplus D_3$
	C_1	**D_0**			**D_2**	**D_3**	$C_1 = D_0 \oplus D_2 \oplus D_3$
			C_2	**D_1**	**D_2**	**D_3**	$C_2 = D_1 \oplus D_2 \oplus D_3$

En cuanto al proceso de decodificación, el primer paso es calcular la paridad de cada fila para obtener el síndrome, el cual indicará si la información recibida es correcta (síndrome igual a cero) o la información contiene errores (síndrome distinto de cero). El síndrome se calcula fácilmente a partir de la matriz de paridad **H**, tal y como se puede ver en la Tabla 4. Cada síndrome se asocia con un error distinto. En el caso de los códigos de Hamming, si el valor del síndrome coincide con alguna de las columnas de la matriz, indica que el error se ha producido en el bit correspondiente a esa columna. Obviamente, si el síndrome es cero, se entiende que no hay errores.

Tabla 4. Diseño decodificador SEC Hamming (7, 4).

Información recibida							Fórmulas cálculo síndrome
R_0	**R_1**	**R_2**	**R_3**	**R_4**	**R_5**	**R_6**	
1	**0**	**1**	**0**	**1**	**0**	**1**	$S_0 = R_0 \oplus R_2 \oplus R_4 \oplus R_6$
0	**1**	**1**	**0**	**0**	**1**	**1**	$S_1 = R_1 \oplus R_2 \oplus R_5 \oplus R_6$
0	**0**	**0**	**1**	**1**	**1**	**1**	$S_2 = R_3 \oplus R_4 \oplus R_5 \oplus R_6$

Veamos ahora un sencillo ejemplo de codificación y decodificación. Sea el dato *D* = (0110). Utilizando la Tabla 3, calculamos la palabra codificada (ver Tabla 5):

Tabla 5. Codificación de la palabra de datos (0110).

C_0	**C_1**	**D_0**	**C_2**	**D_1**	**D_2**	**D_3**	
		0		**1**	**1**	**0**	
1	**0**	**1**	**0**	**1**	**0**	**1**	$C_0 = 0 \oplus 1 \oplus 0 = 1$
0	**1**	**1**	**0**	**0**	**1**	**1**	$C_1 = 0 \oplus 1 \oplus 0 = 1$
0	**0**	**0**	**1**	**1**	**1**	**1**	$C_2 = 1 \oplus 1 \oplus 0 = 0$

Al finalizar el proceso de codificación, obtenemos la palabra codificada (1100110). Supongamos ahora que esta palabra codificada sufre un error en un bit (es indiferente que sea de datos o de código), de forma que la palabra recibida sea (110010), donde se ha resaltado el bit erróneo. Si ahora se decodifica esta palabra mediante las fórmulas mostradas en la Tabla 4, obtenemos el síndrome 110 (tal y como se muestra en la Tabla 6). Al ser el síndrome distinto de cero, podemos asumir que el dato no es correcto. La columna de la matriz de paridad cuyo valor coincida con el síndrome obtenido indica la posición del bit erróneo (la columna resaltada en la Tabla 6). La corrección de este error es tan sencilla

como invertir el bit correspondiente. Una vez corregido ese bit y extrayendo los bits de datos, se obtiene la palabra decodificada.

Tabla 6. Decodificación de la palabra de datos (110010).

R_0	R_1	R_2	R_3	R_4	R_5	R_6	
1	**1**	**0**	**0**	**1**	**0**	**0**	
1	**0**	**1**	**0**	**1**	**0**	**1**	$S_0 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$
0	**1**	**1**	**0**	**0**	**1**	**1**	$S_1 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$
0	**0**	**0**	**1**	**1**	**1**	**1**	$S_0 = 0 \oplus 1 \oplus 0 \oplus 0 = 1$

4.4 Otros códigos de detección/corrección de errores

A partir de la idea de las paridades solapadas que utiliza el código SEC de Hamming, se han diseñado diferentes tipos de Códigos de Detección y Corrección de Errores. En este apartado vamos a ver algunos de ellos.

Códigos SEC-DED de Hamming

Este código permite, además de corregir un error en un bit, detectar dos errores en dos bits aleatorios (DED: *Double Error Detection*). Se forma a partir del código SEC, pero añadiendo un bit más de paridad, el cual engloba al resto de bits. De esta forma, la distancia de Hamming de este código pasa a ser igual a 4. Como ejemplo, a continuación se muestran la matriz de paridad y las fórmulas para la codificación del código SEC-DED(8, 4) basado en el código de Hamming visto en el punto anterior (ver Figura 4):

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

a)

$$C_0 = D_0 \oplus D_1 \oplus D_3$$
$$C_1 = D_0 \oplus D_2 \oplus D_3$$
$$C_2 = D_1 \oplus D_2 \oplus D_3$$
$$C_3 = C_0 \oplus C_1 \oplus C_2 \oplus D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_4$$

b)

*Figura 4. Código SEC-DED de Hamming(8, 4): a) Matriz de paridad **H**; b) Fórmulas para la codificación de los datos.*

En el caso de este código, a la hora de comprobar el tipo de fallo, se analiza por separado el síndrome formado por los bits $C_0C_1C_2$ y C_3, tal y como se puede ver en la Tabla 7:

Tabla 7. Análisis bits de paridad código SEC-DED de Hamming.

Bits $C_0C_1C_2$	Bit C_3	Situación	Acción
"000"	'0'	No hay error	-
"000"	'1'	Error en el bit C_3	-
Distinto de "000"	'0'	Error de 2 bits	Señalar el error No corregir nada
Distinto de "000"	'1'	Error de 1 bit	Corregir el error

Códigos SEC-DAEC

Este tipo de códigos permiten corregir errores simples y errores dobles adyacentes (DAEC: *Double Adjacent Error Correction*). Este tipo de fallos son muy comunes en los sistemas de memoria actuales. Con el incremento en la escala de integración de la tecnología CMOS, el tamaño de los transistores que forman las celdillas de memoria es cada vez más pequeño. De esta forma, los errores inducidos, por ejemplo, por radiación, afectan cada vez más a dos celdillas contiguas.

A la hora de diseñar un código de corrección de errores interesa que el número de bits de código sea lo menor posible. Por ejemplo, la siguiente matriz de paridad describe un código SEC-DAEC (13, 8). En este caso, los bits de código se sitúan en las primeras 5 columnas, mientras que los bits de datos ocupan el resto de columnas (ver Figura 5).

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

*Figura 5. Matriz de paridad **H** del código SEC-DAEC(13, 8).*

Las ecuaciones lógicas se obtienen de forma similar al código de Hamming. En caso de error simple, el síndrome coincide con la columna correspondiente a la posición del bit erróneo. En caso de error doble adyacente, el síndrome será la OR exclusiva de las columnas correspondientes a las posiciones afectadas.

Códigos Ultrafast

Este tipo de códigos aumenta la redundancia con el fin de obtener codificadores y decodificadores muy rápidos. Su uso está pensado para aquellos elementos en los cuales se necesita una latencia de funcionamiento muy baja, como puede ser el banco de registros. Un ejemplo de un código Ultrafast SEC(8, 4) es el que se muestra en la Figura 6. La Figura 6.a muestra la matriz de paridad, la Figura 6.b las fórmulas para codificar la información, y la Figura 6.c las fórmulas para calcular el síndrome (siendo D_i los bits de datos a enviar/almacenar, C_i los bits de código, S_i los bits de síndrome y R_i los bits recibidos/leídos). Como se puede ver, tanto la codificación como el cálculo del síndrome son operaciones muy rápidas. Eso sí, la redundancia es del 100%.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

a)

$$C_0 = D_0 \oplus D_1$$
$$C_1 = D_0 \oplus D_2$$
$$C_2 = D_1 \oplus D_3$$
$$C_3 = D_2 \oplus D_3$$

b)

$$S_0 = R_0 \oplus R_4 \oplus R_5$$
$$S_1 = R_1 \oplus R_4 \oplus R_6$$
$$S_2 = R_2 \oplus R_5 \oplus R_7$$
$$S_3 = R_3 \oplus R_6 \oplus R_7$$

c)

*Figura 6. Código Ultrafast SEC(8, 4): a) Matriz de paridad **H**; b) Fórmulas para la codificación de los datos; c) Fórmulas para el cálculo del síndrome.*

Códigos Matriciales

En lugar de representar la información en forma de vector, este tipo de Códigos de Corrección de Errores organiza los datos y los bits de código de forma bidimensional, tal y como se puede ver en la Figura 7, donde X_i representan los bits de datos, C_i los bits de código

en horizontal (calculados mediante un código de SEC de Hamming), y P_i los bits de paridad vertical (calculados mediante paridad par). Como se puede ver, los bits de datos (X_i) se juntan en grupos de cuatro bits, protegidos por un código SEC de Hamming (7, 4) (bits C_i). Completan la matriz los bits de paridad P_i, que protegen los bits de las columnas.

X_1	X_2	X_3	X_4	C_1	C_2	C_3
X_5	X_6	X_7	X_8	C_4	C_5	C_6
X_9	X_{10}	X_{11}	X_{12}	C_7	C_8	C_9
X_{13}	X_{14}	X_{15}	X_{16}	C_{10}	C_{11}	C_{12}
P_1	P_2	P_3	P_4			

a)

$$C_1 = X_2 \oplus X_3 \oplus X_4$$
$$C_2 = X_1 \oplus X_3 \oplus X_4$$
$$C_3 = X_1 \oplus X_2 \oplus X_4$$
$$C_4 = X_6 \oplus X_7 \oplus X_8$$
$$C_5 = X_5 \oplus X_7 \oplus X_8$$
$$C_6 = X_5 \oplus X_6 \oplus X_8$$
$$C_7 = X_{10} \oplus X_{11} \oplus X_{12}$$
$$C_8 = X_9 \oplus X_{11} \oplus X_{12}$$
$$C_9 = X_9 \oplus X_{10} \oplus X_{12}$$
$$C_{10} = X_{14} \oplus X_{15} \oplus X_{16}$$
$$C_{11} = X_{13} \oplus X_{15} \oplus X_{16}$$
$$C_{12} = X_{13} \oplus X_{14} \oplus X_{16}$$

b)

$$p_1 = X_1 \oplus X_5 \oplus X_9 \oplus X_{13}$$
$$p_2 = X_2 \oplus X_6 \oplus X_{10} \oplus X_{14}$$
$$p_3 = X_3 \oplus X_7 \oplus X_{11} \oplus X_{15}$$
$$p_4 = X_4 \oplus X_8 \oplus X_{12} \oplus X_{16}$$

c)

Figura 7. a) Esquema código matricial; b) Fórmulas para el cálculo de los bits de código; c) Fórmulas para el cálculo de los bits de paridad

5 Cierre

A lo largo de este trabajo hemos presentado los conceptos básicos de los Códigos de Detección y Corrección de Errores. Esta técnica de Tolerancia a Fallos es muy utilizada para proteger la información, principalmente en memorias y en transmisión de datos.

Nos hemos centrado en el estudio detallado del código SEC de Hamming, a partir del cual se han desarrollado multitud de Códigos de Detección y Corrección de Errores. Todos estos códigos se basan en la idea de las paridades solapadas. Algunos ejemplos son los códigos SEC-DED, SEC-DAEC, Ultrafast o matriciales, mostrados en este trabajo.

6 Bibliografía

Dubrova, E.: “Fault-Tolerant Design”, Ed. Springer, 2013.

Fujiwara, E.: “Code Design for Dependable Systems: Theory and Practical Application”, Ed. Wiley-Interscience, 2006.

Hamming, R. W.: “Error detecting and error correcting codes,” Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

Gracia-Morán, J., Saiz-Adalid, L.J., Gil-Tomás, D., Gil-Vicente, P.J.: “Un nuevo Código de Corrección de Errores matricial con baja redundancia”, III Jornadas de Computación Empotrada y Reconfigurable (JCER2018), Jornadas SARTECO, pp. 561-566, Teruel, España, Septiembre 2018.

Saiz-Adalid, L.J., Gracia-Morán, J., Gil-Tomás, D., Baraza-Calvo, J.C., Gil-Vicente, P.J.: “Ultrafast Codes for Multiple Adjacent Error Correction and Double Error Detection”, IEEE Access, Vol.: 7(1), pp. 151131-151143, Diciembre 2019.