Universidad Zaragoza

ACTAS DE LAS JORNADAS SARTECO

12-14 SEPT

AVANCES EN ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Editado por:

Francisco J. Martínez

Julio A. Sangüesa

Piedad Garrido

Arturo González-Escribano

Diego R. Llanos

Sergio Cuenca Asensi

Jesús González Peñalver

Avances en arquitectura y tecnología de computadores
Actas de las Jornadas SARTECO 2018

Editores: Francisco J. Martínez, Julio A. Sangüesa, Piedad Garrido, Arturo Gonzalez-Escribano,
Diego R. Llanos, Sergio Cuenca Asensi, Jesús González Peñalver

(c) 2018, Jornadas SARTECO

ISBN-13: 978-84-09-04334-7

Teruel, 2018

Un nuevo Código de Corrección de Errores matricial con baja redundancia

J. Gracia-Morán, L.J. Saiz-Adalid, D. Gil-Tomás, P.J. Gil-Vicente
Instituto ITACA, Universitat Politècnica de València
E-mail: { jgracia, ljsaiz, dgil, pgil }@itaca.upv.es

***Resumen*—Actualmente, y debido al continuo aumento en la escala de integración, la tasa de fallos en los sistemas de memoria de los computadores ha aumentado. Así, la probabilidad de que se produzcan *Single Cell Upsets* (SCUs) o *Multiple Cell Upsets* (MCUs) aumenta. Una solución común es el uso de Códigos de Corrección de Errores (ECCs). Sin embargo, cuando se utilizan ECCs en aplicaciones empotradas, se debe lograr un buen equilibrio entre la cobertura de errores, la redundancia introducida y la eficiencia en términos de área de silicio ocupada, potencia consumida y retardo de los circuitos de codificación y decodificación.**

En este sentido, existen diferentes propuestas para tolerar MCUs. Por ejemplo, los códigos matriciales utilizan códigos de Hamming y controles de paridad en un formato bidimensional para detectar y/o corregir MCUs. Sin embargo, estos códigos introducen una gran redundancia, lo que conlleva una sobrecarga excesiva con respecto al área, potencia consumida y retardo.

En este trabajo presentamos un nuevo código matricial con una baja redundancia, que permite corregir diferentes patrones de MCUs y que no introduce una gran sobrecarga en los circuitos de codificación y decodificación.

Palabras Clave—Códigos de Corrección de Errores, Multiple Cell Upsets, Tolerancia a Fallos, Confiabilidad

I. Introducción

En la actualidad, la continua reducción de tamaño de la tecnología CMOS proporciona una gran capacidad de almacenamiento de los sistemas de memoria. Sin embargo, esta disminución de tamaño también provoca un aumento en la tasa de errores de la memoria [1][2]. En este sentido, el impacto de una partícula de radiación cósmica puede provocar el cambio en una única celda de memoria (evento conocido como *Single Cell Upset* o SCU) o, como se muestra en diferentes experimentos, en varias celdas de memoria (*Multiple Cell Upset* o MCUs), es decir, errores simultáneos en más de una celda de memoria inducida por una sola partícula [3][4][5][6][7].

Tradicionalmente, se han utilizado Códigos de Corrección de Errores (ECCs) para proteger los sistemas de memoria. Los códigos más comunes han sido los códigos SEC o SEC-DED [8][9][10]. Los códigos SEC pueden corregir un error en una única celda de memoria, mientras que los códigos SEC-DED pueden corregir un error en una celda de memoria, así como pueden detectar también dos errores en dos celdas independientes.

En aplicaciones críticas, se utilizan códigos más complejos y sofisticados [11][12][13][14][15][16][17][18]. Por ejemplo, los códigos matriciales [14][15] son unos códigos bien conocidos que combinan los códigos de Hamming con una verificación de paridad en un formato bidimensional, lo que permite la corrección y/o detección de dos bits erróneos.

Sin embargo, al añadir un ECC a un sistema de memoria, se deben tener en cuenta una serie de factores. El primero de ellos es la redundancia requerida, es decir, los bits adicionales que se usan para detectar y/o corregir los posibles errores ocurridos, y que se añaden a cada palabra de datos almacenada en la memoria. De esta forma, la cantidad de almacenamiento ocupada por los bits redundantes se escala con la capacidad de memoria. Por ejemplo, si se emplea un ECC con un 100% de redundancia en una memoria de 2GB, solo 1GB estará disponible para almacenar la carga (los datos "limpios"); el 1GB restante es requerido para los bits de código.

Otro factor a tener en cuenta es la complejidad de los circuitos de codificación y decodificación, pues esta complejidad afectará a la sobrecarga introducida con respecto al área de silicio ocupada, potencia consumida y retardo de dichos circuitos.

En este trabajo presentamos un nuevo ECC matricial que reduce en gran medida la redundancia introducida, al tiempo que mantiene, o incluso mejora, la cobertura de errores, reduciendo de este modo la sobrecarga introducida.

Este nuevo código ha sido diseñado utilizando la metodología FUEC, desarrollada por los autores en [19], donde se introduce un algoritmo (y una herramienta) para diseñar códigos FUEC. Los códigos FUEC son una mejora de los *Códigos de Control de Errores Desiguales* (UEC) [8].

Este trabajo se organiza de la siguiente manera. La Sección II presenta una breve introducción al diseño de ECCs, así como presenta los tipos de error más comunes. La Sección III resume el funcionamiento de los códigos matriciales, y presenta nuestra propuesta. La Sección IV describe los diferentes resultados obtenidos durante la evaluación de los ECCs introducidos en la Sección III. Finalmente, la Sección V concluye este trabajo.

II. Códigos de Corrección de Errores

A. *Introducción al diseño de Códigos de Corrección de Errores*

Un ECC binario (n, k) codifica una palabra de entrada de k bits en una palabra de salida de n bits [20]. La palabra de entrada $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ es un vector de k bits que representa los datos originales. La palabra de código $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ es un vector de n bits, donde los $(n - k)$ bits redundantes añadidos se llaman bits de paridad o bits de código. $\mathbf{b}$ se transmite a través de un canal no confiable que entrega la palabra recibida $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$. El vector de error $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ modela el error inducido por el

ISBN 978-84-09-04334-7

canal. Si no ha ocurrido ningún error en el bit *i*-ésimo, $e_i = 0$; de lo contrario, $e_i = 1$. De esta forma, **r** se puede interpretar como $\mathbf{r} = \mathbf{b} \oplus \mathbf{e}$. La Fig. 1 sintetiza esta codificación, transmisión y proceso de decodificación.

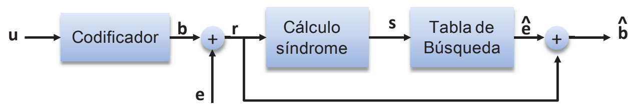

Fig. 1. Proceso de codificación, transmisión y decodificación.

La matriz de paridad $\mathbf{H}_{(n-k)\times n}$ de un código lineal define el código [8]. Para el proceso de codificación, **b** debe cumplir el requisito $\mathbf{H}\cdot\mathbf{b}^T=\mathbf{0}$. Para la decodificación del síndrome, éste se define como $\mathbf{s}^T=\mathbf{H}\cdot\mathbf{r}^T$, y depende exclusivamente de **e**:

$$\mathbf{s}^T = \mathbf{H}\cdot\mathbf{r}^T = \mathbf{H}\cdot(\mathbf{b} \oplus \mathbf{e})^T = \mathbf{H}\cdot\mathbf{b}^T \oplus \mathbf{H}\cdot\mathbf{e}^T = \mathbf{H}\cdot\mathbf{e}^T \quad (1)$$

Debe existir un síndrome **s** diferente para cada **e** corregible. Si **s=0**, podemos suponer que **e=0**. Por lo tanto, **r** es correcto. De lo contrario, ha ocurrido un error. La decodificación del síndrome se realiza mediante una tabla de búsqueda que relaciona cada **s** con el vector de error decodificado $\hat{\mathbf{e}}$. La palabra de código decodificada $\hat{\mathbf{b}}$ se calcula como $\hat{\mathbf{b}} = \mathbf{r} \oplus \hat{\mathbf{e}}$. A partir de $\hat{\mathbf{b}}$, es fácil obtener $\hat{\mathbf{u}}$ descartando los bits de paridad. Si las hipótesis de fallos empleadas para diseñar el ECC son consistentes con el comportamiento del canal, $\hat{\mathbf{u}}$ y **u** deben ser iguales con una probabilidad muy alta.

B. *Modelos de Error*

En teoría de códigos [8], el término *error aleatorio* (*random error*) se refiere comúnmente a uno o más bits erróneos, distribuidos aleatoriamente en la palabra codificada (bits de datos más bits de código generados por el ECC). Los *errores aleatorios* pueden ser *simples* o *múltiples*. Los *errores simples* afectan a una única celda de memoria. Se producen comúnmente por lo que en inglés se conoce como *Single Event Upsets* (SEU, en memorias RAM) o *Single Event Transients* (SET, en lógica combinacional) [21].

Como se ha comentado previamente, con el continuo aumento de la escala de integración, los *errores múltiples* son cada vez más frecuentes [3][4][5][6][7]. En la actualidad, se ha observado que cuando una partícula cósmica impacta en una celda de memoria, se produce una radiación de pares electrón-agujero a lo largo de la vía de transporte [22]. De esta forma, se pueden generar *errores adyacentes*, es decir, errores múltiples donde todos los bits erróneos son contiguos. Este es el tipo de error múltiple más frecuente [4][23].

III. CÓDIGOS MATRICIALES

A. *Introducción a los códigos matriciales*

Los códigos matriciales son ECCs que combinan dos o más tipos de códigos para detectar y/o corregir distintos tipos de errores [14][15][16][24][25][26]. Normalmente, se suelen combinar diferentes tipos de códigos de Hamming y verificaciones de paridad. De esta manera, los códigos matriciales forman un esquema bidimensional para detectar y/o corregir diferentes tipos de errores adyacentes, considerando la adyacencia en las dos dimensiones: horizontal y vertical.

Un ejemplo de este esquema se puede ver en la Fig. 2 (extraído de [15]), donde X_i representa los bits de datos, C_i son los bits de control horizontales (calculados mediante un código de Hamming), y P_i son los bits de paridad vertical (calculados mediante paridad par).

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			

Fig. 2. Esquema de código matricial [15].

El funcionamiento básico de este código matricial es el siguiente. Los bits de datos (X_i) se dividen en grupos de 4 bits. Cada grupo está codificado por un código de Hamming (7, 4), que sirve para generar los diferentes C_i. Por último, un conjunto de bits de paridad (P_i) completa la matriz. Un error simple se corrige mediante los bits C_i (código de Hamming). Por otro lado, para corregir errores adyacentes, se utilizan los bits C_i junto con los bits P_i.

El principal problema de este ECC es que introduce una redundancia muy elevada. De esta forma, se aumenta la memoria requerida para los bits de código, así como la sobrecarga introducida con respecto al área, potencia consumida y retardo.

Sin embargo, este código matricial presenta una mejor tasa de detección y corrección que otros códigos, ya que es capaz de corregir todos los errores simples, así como de corregir o detectar todos los errores adyacentes de 2 bits. Además, el hecho de utilizar códigos de Hamming y bits de paridad, provocan que la sobrecarga no sea muy elevada, pues los circuitos para codificar y decodificar estos dos códigos suelen ser muy eficientes.

B. *Nuestra propuesta*

Tal y como se acaba de ver, y a pesar de la redundancia que introducen, los códigos matriciales pueden ser una buena opción para el tratamiento de errores adyacentes. En este sentido, y utilizando la metodología FUEC [19], hemos sido capaces de generar un código matricial con una baja redundancia y que es capaz de corregir errores simples así como diferentes tipos de errores adyacentes.

El esquema de nuestro código se puede ver en la Fig. 3, donde X_i representa los bits de datos y C_i los bits de control (calculados mediante un código generado con la metodología FUEC).

C_0	C_1	C_2	C_3	C_4
C_5	C_6	C_7	C_8	X_0
X_1	X_2	X_3	X_4	X_5
X_6	X_7	X_8	X_9	X_{10}
X_{11}	X_{12}	X_{13}	X_{14}	X_{15}

Fig. 3. Esquema de nuestro código matricial.

Como hemos comentado en la Sección II.A, un código queda definido por su matriz de paridad **H**. En este sentido, la matriz **H** de nuestro código es la siguiente:

ISBN 978-84-09-04334-7

$$H = \begin{bmatrix} 1000000001100000100100000 \\ 0100000000011100010000000 \\ 0010000001000011000001000 \\ 0001000000100100000010010 \\ 0000100000010010001000001 \\ 0000010000001001000000000 \\ 0000001000000000101010100 \\ 0000000100000000010100010 \\ 0000000010000000000001101 \end{bmatrix} \quad (2)$$

A partir de **H**, es muy sencillo obtener las fórmulas para la codificación y para la obtención del síndrome, tal y como se puede ver en la TABLA I. En concreto, la TABLA I-a muestra las fórmulas para la obtención del código que se almacena con cada una de las palabras de datos, mientras que la TABLA I-b muestra las fórmulas para la obtención del síndrome.

TABLA I
FÓRMULAS OBTENIDAS A PARTIR DE LA MATRIZ DE PARIDAD (2)

a)	b)
$C_0 = X_0 \oplus X_1 \oplus X_7 \oplus X_{10}$	$s_0 = C_0 \oplus X_0 \oplus X_1 \oplus X_7 \oplus X_{10}$
$C_1 = X_2 \oplus X_3 \oplus X_4 \oplus X_8$	$s_1 = C_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_8$
$C_2 = X_0 \oplus X_5 \oplus X_6 \oplus X_{12}$	$s_2 = C_2 \oplus X_0 \oplus X_5 \oplus X_6 \oplus X_{12}$
$C_3 = X_1 \oplus X_4 \oplus X_{11} \oplus X_{14}$	$s_3 = C_3 \oplus X_1 \oplus X_4 \oplus X_{11} \oplus X_{14}$
$C_4 = X_2 \oplus X_5 \oplus X_9 \oplus X_{15}$	$s_4 = C_4 \oplus X_2 \oplus X_5 \oplus X_9 \oplus X_{15}$
$C_5 = X_3 \oplus X_6$	$s_5 = C_5 \oplus X_3 \oplus X_6$
$C_6 = X_7 \oplus X_9 \oplus X_{11} \oplus X_{13}$	$s_6 = C_6 \oplus X_7 \oplus X_9 \oplus X_{11} \oplus X_{13}$
$C_7 = X_8 \oplus X_{10} \oplus X_{14}$	$s_7 = C_7 \oplus X_8 \oplus X_{10} \oplus X_{14}$
$C_8 = X_{12} \oplus X_{13} \oplus X_{15}$	$s_8 = C_8 \oplus X_{12} \oplus X_{13} \oplus X_{15}$

Con respecto a las ventajas de nuestra propuesta, la primera de ellas es que nuestro código solamente introduce 9 bits de redundancia, en vez de los 16 utilizados en el código anterior.

La importancia de una baja redundancia proviene del hecho de que estos bits adicionales también deben almacenarse en la memoria. De esta forma, una mayor redundancia significa una menor disponibilidad para los bits de datos. Por ejemplo, si tenemos un chip de memoria de 1GB, solo 512MB estarán disponibles para almacenar bits de datos en el caso del código matricial de la Sección III.A, pues los 512MB restantes son necesarios para almacenar los bits de código. En el caso de nuestra propuesta, sólo 370MB son necesarios para almacenar los bits de código, pudiéndose dedicar sobre 655MB al almacenamiento de los bits de datos.

Otra diferencia es su cobertura de errores. Nuestra propuesta es capaz de corregir diferentes patrones de errores: errores simples, errores dobles adyacentes tanto en horizontal como en vertical, y errores cuádruples adyacentes en cuadrados de 2x2.

IV. EVALUACIÓN DE LOS CÓDIGOS MATRICIALES

En esta sección vamos a presentar los diferentes resultados obtenidos durante la evaluación de los códigos matriciales presentados en la Sección III. Para ello, hemos realizado dos procesos diferentes. Durante el primero, hemos inyectado fallos en los modelos en C de los códigos matriciales. Con esta inyección, evaluamos la cobertura de errores. En un segundo paso, hemos implementado los dos códigos matriciales en VHDL, y los hemos sintetizado con el fin de estimar la sobrecarga introducida con respecto al área, potencia consumida y retardo.

A. *Evaluación de la cobertura de errores*

Con el fin de estudiar la cobertura de errores de los dos códigos presentados previamente, hemos desarrollado un simulador que permite inyectar diferentes tipos de error. El esquema básico se muestra en la Fig. 4.

Esta herramienta permite la inyección de diferentes tipos de errores. Al comparar las palabras de entrada y salida, el simulador puede verificar si el error inyectado conduce a una decodificación correcta o incorrecta. Además, el circuito del decodificador puede activar la señal NRE (*Error No Recuperable*) cuando se detecta un error que no se puede corregir. Repitiendo el proceso para todos los errores de un tamaño y modelo dado, es posible contar el número de errores corregidos y/o detectados con respecto al número total de errores posibles, es decir, es posible calcular la cobertura de cada código.

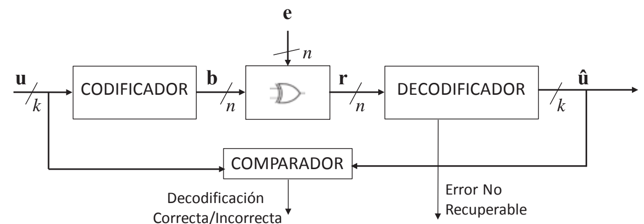

Fig. 4. Diagrama de bloques de la herramienta de inyección de fallos.

Todos los bloques de la herramienta de inyección se han desarrollado en C, utilizando los operadores de lógica de bit para una simulación precisa del comportamiento del hardware. Los circuitos codificadores y decodificadores se pueden obtener fácilmente a partir de la matriz **H**, tal y como se ha visto en la Sección III. Estos circuitos se implementan en C como funciones de codificación y decodificación. Cambiar el simulador para un código diferente es tan simple como ajustar las longitudes de palabra y reemplazar las funciones de codificación y decodificación para el nuevo código.

Hay que remarcar que no hemos inyectado errores según su probabilidad de ocurrencia, ya que nuestro objetivo es medir las coberturas de corrección. Específicamente, hemos inyectado cada tipo de error (errores simples o errores adyacentes de diferentes longitudes) en todos los bits de la palabra de código para verificar las capacidades de corrección de errores de los diferentes códigos.

Con respecto al tipo de error inyectado, en este trabajo hemos inyectado errores simples, así como errores adyacentes con distintos patrones, tal y como se puede ver en la Fig. 5. En concreto, hemos inyectado errores adyacentes con una longitud comprendida entre 2 y 5 utilizado un patrón horizontal (Fig. 5-a) y vertical (Fig. 5-b). Además, también hemos inyectado errores adyacentes con un patrón cuadrado (Fig. 5-c), particularmente errores de un tamaño de 2x2, 2x3 y 3x2.

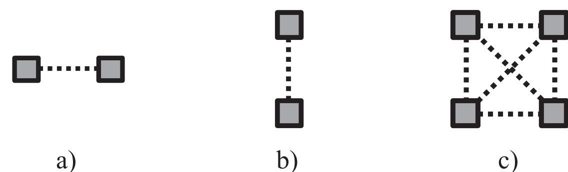

Fig. 5. Patrones utilizados en la inyección de fallos.

ISBN 978-84-09-04334-7
9 788409 043347

En la TABLA II podemos ver las diferentes coberturas de corrección de los dos códigos presentados en las secciones anteriores. En concreto, la cobertura de corrección la hemos calculado como:

$$C_{correc} = \frac{Errores_Corregidos}{Errores_Inyectados} \times 100 \quad (3)$$

donde *Errores_Corregidos* es el número de errores corregidos por cada código, mientras que *Errores_Inyectados* es el número de errores inyectados de un determinado patrón.

Como se esperaba, nuestra propuesta puede corregir todos los errores simples y dobles adyacentes, tanto con los patrones horizontal y vertical como con el patrón cuadrado. Por otro lado, esta cobertura de corrección es nula (o casi) en patrones de error con un tamaño mayor. Esto es debido a que al tener una redundancia tan baja, el número de síndromes disponibles no es muy elevado, lo que hace que casi no queden síndromes disponibles para aquellos errores no contemplados en las hipótesis de fallos (corrección de errores simples y de errores dobles adyacentes en varios patrones).

Con respecto al código matricial presentado en [15], este código sólo es capaz de corregir errores simples, mientras que para errores múltiples adyacentes su comportamiento es más irregular. El principal inconveniente de este método es que el código de Hamming utilizado para calcular los distintos C_i únicamente tiene en cuenta los errores simples. De esta forma, un error adyacente que afecte a un bit de datos y a otro de código (por ejemplo, un error que afecte a los bits X_4 y C_1) es tratado como un errro simple, y corregido de forma errónea. En cuanto a los patrones verticales, aquellos que son impares son tratados correctamente, gracias a los bits de paridad par P_i. Por último, la cobertura de corrección de los patrones de errores cuadrados es bastante baja. Este es un resultado esperado, pues en este caso, se están inyectando una gran cantidad de errores.

TABLA II
PORCENTAJES DE ERRORES CORREGIDOS

	Código matricial [15]	Nuestra Propuesta
Patrón de Errores Horizontal		
Longitud del Error	% Errores Corregidos	% Errores Corregidos
1	100,00	100,00
2	89,15	100,00
3	45,45	0,00
4	5,88	0,00
5	0,00	0,00
Patrón de Errores Vertical		
Longitud del Error	% Errores Corregidos	% Errores Corregidos
1	100,00	100,00
2	36,00	100,00
3	100,00	6,67
4	27,27	0,00
5	100,00	0,00
Patrón de Errores Cuadrado		
Longitud del Error	% Errores Corregidos	% Errores Corregidos
2x2	28,57	100,00
3x2	26,67	0,00
2x3	17,65	0,00

En conclusión, nuestro código es muy eficiente para tolerar errores simples y dobles adyacentes de diferentes patrones. Si el comportamiento esperado de la memoria se ajusta a esta hipótesis de fallo, nuestra propuesta es una alternativa válida. Más allá este tipo de errores, el rendimiento de nuestros códigos disminuye notablemente debido a su baja redundancia. En este caso, habría que recurrir a códigos más complejos.

B. Resultados de la síntesis de los códigos matriciales

En la Sección III.B hemos visto que nuestro código matricial presenta una menor redundancia con respecto al código matricial de la Sección III.A. Esta menor redundancia provoca que se necesite menos memoria para almacenar los bits de código. En este apartado, vamos a comprobar si esa menor redundancia también se traduce en una menor sobrecarga con respecto al área, potencia consumida y retardo de los circuitos codificadores y decodificadores.

Para ello, hemos sintetizado los diferentes circuitos de los dos códigos matriciales presentados anteriormente. En primer lugar, los hemos implementado en VHDL, y utilizando el software CADENCE [27], hemos llevado a cabo una síntesis lógica para la tecnología de 45 nm, mediante el uso de la biblioteca NanGate FreePDK45 [28][29].

Como se puede ver en la Fig. 6, nuestra propuesta presenta una menor sobrecarga espacial tanto en el codificador como en el decodificador. La menor redundancia de nuestra propuesta provoca que los circuitos de nuestro código sean más sencillos, lo que se traduce en una menor sobrecarga con respecto al área ocupada. Además, hay que tener en cuenta que nuestra propuesta también necesita una menor cantidad de memoria donde almacenar los bits de código.

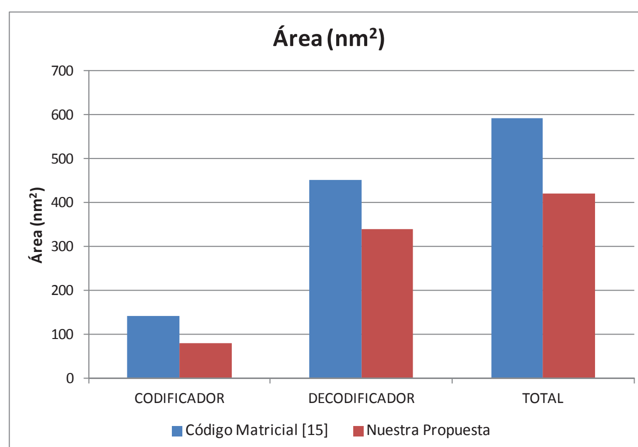

Fig. 6. Área ocupada por los diferentes ECCs matriciales.

Por otra parte, en la Fig. 7 se puede ver la potencia consumida por los diferentes códigos matriciales. Como en el caso del área, nuestra propuesta también presenta un menor consumo. Este es un resultado esperado, ya que el consumo suele ser proporcional al área ocupada. Al emplear nuestros circuitos una menor área, se produce un menor consumo.

ISBN 978-84-09-04334-7
9 788409 043347

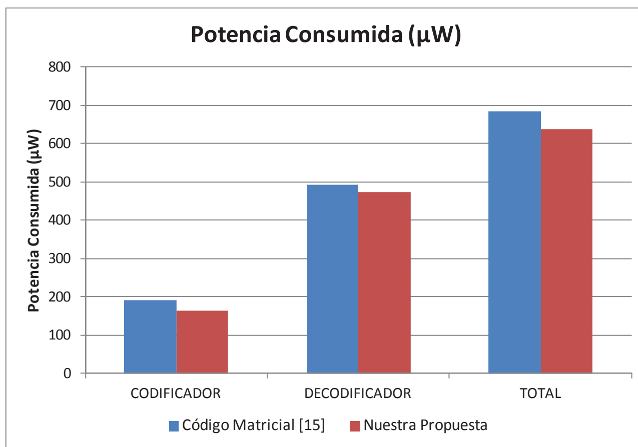

Fig. 7. Potencia consumida por los diferentes ECCs matriciales.

Finalmente, la Fig. 8 muestra la sobrecarga temporal de ambos códigos matriciales. Como se puede observar, nuestra propuesta introduce un menor retardo.

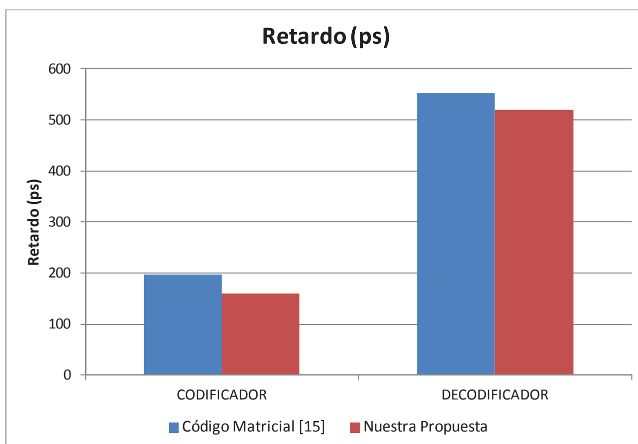

Fig. 8. Sobrecarga temporal de los diferentes ECCs matriciales.

En resumen, podemos concluir que, gracias a la baja redundancia introducida por nuestro código, se disminuye la sobrecarga introducida, pues tanto el codificador como el decodificador son más simples. Es importante remarcar que los datos presentados en esta sección hacen referencia únicamente a los circuitos de codificación y decodificación. Los bits redundantes también ocupan espacio y consumen energía, con lo que nuestra propuesta todavía resulta más ventajosa.

V. Conclusiones

En este trabajo, se ha presentado un nuevo código corrector de errores de tipo matricial que introduce una baja redundancia. Además, este nuevo código es capaz de corregir errores simples y diferentes patrones de errores múltiples.

Para comprobar la eficiencia de nuestra propuesta, hemos comparado tanto la cobertura de corrección de errores como la sobrecarga introducida con respecto al área de silicio ocupada, potencia consumida y retardo con otro código matricial bien conocido.

Con respecto a la cobertura de corrección, nuestra propuesta es capaz de corregir el 100% de los errores simples y de los errores adyacentes dobles, con distintos patrones de ocurrencia.

Por otro lado, esta baja redundancia también provoca que la sobrecarga con respecto al área ocupada, potencia consumida y retardo de los circuitos de codificación y decodificación sea menor.

En general, nuestra propuesta es una opción adecuada para aplicaciones en las que se esperen errores adyacentes dobles. Más allá de este tipo de errores, el rendimiento de nuestro código disminuye notablemente debido a su baja redundancia. Si se espera que ocurran estos errores, se deben emplear ECCs más potentes.

En futuros trabajos, queremos continuar desarrollando códigos con el fin de disminuir la sobrecarga en el área, la potencia consumida y el retardo, manteniendo, o incluso mejorando, la cobertura de error. Por otro lado, también queremos desarrollar otros códigos centrados en los errores múltiples adyacentes de mayor tamaño, que se espera que tengan un impacto cada vez más importante..

Agradecimientos

El presente trabajo ha sido parcialmente financiado por el gobierno de España mediante el proyecto de investigación TIN2016-81075-R.

Referencias

[1] The International Technology Roadmap for Semiconductors 2013. [Online]. Available at: http://www.itrs2.net/2013-itrs.html

[2] S.K. Kurinec and K. Iniewsky. *Nanoscale Semiconductor Memories: Technology and Application*, CRC Press, Taylor & Francis Group, 2014.

[3] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule", IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, July 2010.

[4] G. Tsiligiannis et. al., "Multiple Cell Upset Classification in Commercial SRAMs", IEEE Transactions on Nuclear Science, vol. 61, no. 4, August 2014.

[5] G.I. Zebrev, "Multiple Cell Upset Cross-Section Uncertainty in Nanoscale Memories: Microdosimetric Approach", 15th European Conference on Radiation and its Effects on Components and Systems (RADECS 2015), September 2015.

[6] N.G. Chechenin and M. Sajid, "Multiple cell upsets rate estimation for 65 nm SRAM bit-cell in space radiation environment", 3rd International Conference and Exhibition on Satellite & Space Missions, May 2017.

[7] N.N. Mahatme, B.L. Bhuva, Y.P. Fang, and A.S. Oates, "Impact of strained-Si PMOS transistors on SRAM soft error rates", IEEE Trans. on Nuclear Science, vol. 59, no. 4, pp. 845–850, August 2012.

[8] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application, Ed. Wiley-Interscience, 2006.

[9] R. W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

[10] C.L. Chen and M.Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", IBM Journal of Research and Development, vol. 58, no. 2, pp. 124–134, March 1984.

[11] G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Fault Tolerant Solid State Mass Memory for Space Applications", IEEE Trans. on Aerospace and Electronic Systems, vol. 41, no. 4, pp. 1353–1372, October 2005.

[12] S. Pontarelli, G.C. Cardarilli, M. Re and A. Salsano, "Error correction codes for SEU and SEFI tolerant memory systems", 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2009), pp. 425-430, 2009.

[13] A. Sánchez-Macián, P. Reviriego, J. Tabero, A. Regadío, and J.A. Maestro, "SEFI protection for Nanosat 16-bit Chip On-Board Computer Memories", IEEE Transactions on Device and Materials Reliability, DOI 10.1109/TDMR.2017.2750718, 2017.

[14] C. Argyrides, D.K. Pradhan, and T. Kocak, "Matrix codes for reliable and cost efficient memory chips", IEEE Transactions on Very Large

ISBN 978-84-09-04334-7

Scale Integration (VLSI) Systems, vol. 19, nº 3, pp.420–428, March 2011.

[15] C. Argyrides, H.R. Zarandi and D.K. Pradhan, "Matrix Codes: Multiple Bit Upsets Tolerant Method for SRAM Memories", 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2007.

[16] H.S. de Castro, et al. "A correction code for multiple cells upsets in memory devices for space applications", 2016 14th IEEE International New Circuits and Systems Conference (NEWCAS 2016), pp.1–4, June 2016.

[17] S. Ahmad, M. Zahra. S.Z. Farooq, and A. Zafar, "Comparison of EDAC schemes for DDR memory in space applications", 2013 International Conference on Aerospace Science & Engineering (ICASE 2013), August 2013.

[18] D.E. Muller, "Application of boolean algebra to switching circuit design and to error detection", IRE Transactions on Electronic Computers, vol. 3, pp. 6–12, 1954.

[19] L.J. Saiz-Adalid et al., "Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels", 32th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2013), pp. 178-189, September 2013.

[20] A. Neubauer, J. Freudenberger, and V. Kühn, Coding Theory: Algorithms, Architectures and Applications. John Wiley & Sons, 2007.

[21] K.A. LaBel, "Proton single event effects (SEE) guideline" submitted for publication on the NASA Electronic Parts and Packaging (NEPP) Program web site, August 2009. Available online at https://nepp.nasa.gov/files/18365/Proton_RHAGuide_NASAAug09.pdf

[22] M. Murat, A. Akkerman, and J. Barak, "Electron and ion tracks in silicon: Spatial and temporal evolution," IEEE Transactions on Nuclear Science, vol. 55, no. 6, pp. 3046–3054, December 2008.

[23] M. Wirthlin, D. Lee, G. Swift, and H. Quinn, "A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and SECDED-protected arrays," IEEE Transactions on Nuclear Science, vol. 61, no. 6, pp. 3080–3087, Dec. 2014.

[24] S. Liu, L. Xiao, J. Li, Y. Zhou, and Z. Mao, "Low Redundancy Matrix-Based codes for Adjacent Error Correction with Parity Sharing", 2017 18th International Symposium on Quality Electronic Design (ISQED 2017, March 2017.

[25] P. Reviriego and J.A. Maestro, "Efficient Error Detection Codes for Multiple-Bit Upset Correction in SRAMs with BICS", ACM Transactions on Design Automation of Electronic Systems (TODAES) Vol. 14 nº 1, January 2009.

[26] M. Zhu, L. Xiao, S. Li, and Y. Zhang, "Efficient Two-Dimensional Error Codes for Multiple Bit Upsets Mitigation in Memory", 2010 25th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2010), pp. 129-135, October 2010.

[27] https://www.cadence.com/

[28] J.E Stine et al., "FreePDK: An Open-Source Variation-Aware Design Kit", IEEE International Conference on Microelectronic Systems Education (MSE'07), June 2007.

[29] http://www.nangate.com/?page_id=2325

ISBN 978-84-09-04334-7