Gracia-Morán, J.; Saiz-Adalid, L.; Baraza-Calvo, J.; Gil Tomás, DA.; Gil, P. (2021). Design, Implementation and Evaluation of a Low Redundant Error Correction Code. IEEE Latin America Transactions. 19(11):1903-1911. https://doi.org/10.1109/TLA.2021.9475624

Design, Implementation and Evaluation of a Low Redundant Error Correction Code

J. Gracia-Morán, L.J. Saiz-Adalid, J.C. Baraza-Calvo, D. Gil-Tomás, P.J. Gil-Vicente,
Member, IEEE

***Abstract*—The continuous increment in the integration scale of CMOS technology has provoked an augment in the fault rate. Particularly, a single particle hit in a storage element (such as memory or registers) can provoke a single error in a memory cell (known as Single Cell Upsets or SCU), as well as simultaneous errors in more than one memory cell (known as Multiple Cell Upsets or MCU). A common method to tolerate this type of errors is the use of Error Correction Codes (ECC). However, the addition of an ECC introduces a series of overheads: silicon area, power consumption and delay overheads of encoding and decoding circuits, as well as several extra bits added to detect and/or correct errors. An ECC can be designed focusing on different parameters: low redundancy, low delay, error coverage, etc. The design of ECC is a very active field, and ECC with different properties are continuously proposed. However, usually, these proposals only present the ECC, not showing what happens when they are included in a microprocessor. The idea of this paper is twofold. First, we present the design of an ECC whose main characteristic is its low number of code bits (low redundancy). This ECC adds 15 redundant bits to a 32-bit data word to form a (47, 32) ECC able to correct single and doble errors, and to detect triple errors. Second, we also study the overheads that this ECC introduces when added to a RISC microprocessor, comparing it with some other well-known ECC.**

***Index Terms*—Error Correction Code, Low Redundancy, Fault- Tolerant Systems, Reliability, Single Cell Upsets, Multiple Cell Upsets.**

I. Introducción

HOY en día, y gracias a la agresiva reducción de tamaño de la tecnología CMOS, los sistemas de memoria proporcionan una gran capacidad de almacenamiento. Sin embargo, esta disminución de tamaño ha provocado también un aumento en la tasa de error de la memoria [1][2]. De esta forma, el impacto de una partícula de radiación cósmica puede causar el cambio en una o varias celdas [3][4][5][6][7], efectos conocidos como *Single Cell Upset* (SCU) o *Multiple Cell Upsets* (MCU), respectivamente.

El presente trabajo ha sido parcialmente financiado por el Gobierno de España mediante el proyecto de investigación TIN2016-81075-R, y por el Vicerrectorado de Investigación, Innovación y Transferencia de la Universitat Politècnica de València (UPV), dentro del programa Primeros Proyectos de Investigación (PAID-06-18) bajo la referencia 200190032.

Todos los autores pertenecen al Instituto ITACA, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain.

E-mail: { jgracia, ljsaiz, jcbaraza, dgil, pgil }@itaca.upv.es

Dirección de correo: Escuela Técnica Superior de Ingeniería Informática, Edificio 1G, Despacho 2S7, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, España.

Autor de contacto: J. Gracia Morán

Uno de los mecanismos más utilizados para proteger memorias son los Códigos de Corrección de Errores (ECC), capaces de detectar y/o corregir bits erróneos. La detección implica que el ECC sabe que hay bits erróneos, pero no puede determinar cuáles son. Si se puede determinar dónde están los errores, invertir los bits afectados permite corregirlos. Por ejemplo, es frecuente el uso de códigos SEC o SEC-DED [8][9][10]. Los códigos SEC son capaces de corregir un error de un bit, mientras que los códigos SEC-DED pueden corregir un error de un bit, así como detectar dos errores en dos bits independientes del mismo dato.

En aplicaciones críticas, en las que se espera la ocurrencia de fallos múltiples, se pueden utilizar otros códigos con una mayor tolerancia a fallos, como pueden ser los códigos OLS (*Orthogonal Latin Square*), *Reed-Muller* (RM), *Reed-Solomon* (RS), BCH (*Bose-Chaudhuri-Hocquenghem*) o *Array* [11][12][13][14][15][16][17][18][19].

Sin embargo, cuando se incluye un ECC en un sistema de memoria, se consumen recursos extra, como son el área de silicio ocupada por los circuitos de codificación y decodificación; se introduce un retardo en la lectura/escritura de los datos y se aumenta el consumo de energía. Además, se tienen que añadir varios bits adicionales (también llamados bits de código o bits redundantes), necesarios para detectar/corregir los posibles errores ocurridos, y que se almacenan junto con cada palabra de datos.

En este trabajo presentamos un ejemplo de ECC cuya principal característica es su baja redundancia. En este sentido, vamos a presentar un completo estudio sobre su diseño, sus propiedades de tolerancia a fallos, así como el área, el retardo y la potencia consumida por los circuitos de codificación y decodificación. Además, también hemos estudiado el impacto de la inclusión de este ECC en un microprocesador RISC.

Este trabajo se organiza de la siguiente forma. La Sección II resume los pasos básicos para el diseño de un ECC. La Sección III introduce el ECC de baja redundancia propuesto en este trabajo. La Sección IV presenta una primera evaluación de la sobrecarga del nuevo ECC propuesto, mientras que la Sección V muestra el estudio de la sobrecarga producida al añadir el ECC en un microprocesador RISC. Finalmente, la Sección VI concluye este trabajo.

II. Introducción al Diseño de Códigos Correctores de Errores

A. Introducción a la codificación de los Códigos Correctores de Errores

Los códigos que se van a estudiar en este artículo son códigos lineales binarios de bloque [20]. Los códigos de

corrección de errores de este tipo suelen parametrizarse como [*n*, *k*, *d*] o (*n*, *k*), y codifican una palabra de entrada de *k* bits en una palabra de salida de *n* bits. El término *d* representa la mínima distancia Hamming del código, que determina sus capacidades de detección y corrección de errores [20]. La palabra de entrada $\mathbf{u} = (u_0, u_1, ..., u_{k-1})$ es un vector de *k* bits que representa los datos originales. La palabra codificada $\mathbf{c} = (c_0, c_1, ..., c_{n-1})$ es un vector de *n* bits, al que se le han añadido $(n - k)$ bits denominados bits de paridad, bits de código o bits redundantes. **c** se transmite a través de un canal poco confiable, el cual entrega la palabra recibida $\mathbf{r} = (r_0, r_1, ..., r_{n-1})$. El vector de error $\mathbf{e} = (e_0, e_1, ..., e_{n-1})$ modela el error inducido por el canal. Si no se ha producido ningún error en el *i*-ésimo bit, entonces $e_i = 0$; de lo contrario, $e_i = 1$. Por lo tanto, **r** puede interpretarse como $\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$. La Fig. 1 sintetiza este proceso de codificación, envío y decodificación.

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

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

Debe haber un síndrome **s** diferente para cada vector de error corregible **e**. La decodificación del síndrome se realiza a través de una tabla de búsqueda (*lookup table*) que relaciona cada **s** con el vector de error decodificado $\hat{\mathbf{e}}$. Si $\mathbf{s} = \mathbf{0}$, podemos suponer que $\hat{\mathbf{e}} = \mathbf{0}$, y por lo tanto, **r** es correcto. De lo contrario, se ha producido un error. La palabra de código decodificada $\hat{\mathbf{c}}$ se calcula como $\hat{\mathbf{c}} = \mathbf{r} \oplus \hat{\mathbf{e}}$. A partir de $\hat{\mathbf{c}}$, es fácil obtener $\hat{\mathbf{u}}$ descartando los bits de paridad. Si la hipótesis de fallos utilizada para diseñar el ECC es consistente con el comportamiento del canal, $\hat{\mathbf{u}}$ y **u** deben ser iguales con una probabilidad muy alta.

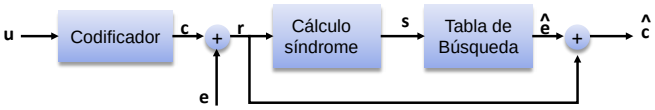

Fig. 1. Procesos de codificación, envío y decodificación de un dato.

B. Diseño de un código de corrección de errores mediante la metodología FUEC

Para diseñar el nuevo ECC de baja redundancia que mostramos en la siguiente sección, hemos utilizado la metodología FUEC [21][22], la cual vamos a resumir a continuación. Aunque esta metodología se diseñó inicialmente para generar códigos FUEC, códigos con varias zonas de cobertura, esta metodología se puede generalizar para el diseño de cualquier ECC [11].

La metodología FUEC formula la búsqueda de una matriz de paridad para un determinado ECC como un problema de satisfacibilidad booleana (*Boolean Satisfiability problem*). A continuación, se resume esta metodología.

Un ECC se define con los siguientes tres pasos:

1. Se determina el conjunto de errores a corregir ($\boldsymbol{E}_+$) y/o detectar ($\boldsymbol{E}_\Delta$), lo que a su vez fija el valor de *d*.
2. Se determina la longitud de la palabra codificada (*n*). Este número se calcula usando los conjuntos definidos en el paso 1, junto con la longitud de la palabra sin codificar (*k*), de tal manera que el número de bits redundantes que se van a utilizar vienen definidos por $(n - k)$.
3. El último paso es encontrar, si existe, la matriz de paridad $\mathbf{H}_{(n-k)\times n}$. Esta matriz definirá el ECC buscado, el cual debe ser capaz de corregir y/o detectar los errores seleccionados, según las fórmulas (2) y (3), en las que se modelan los errores con sus vectores de error **e**.

$$\boldsymbol{H}\cdot\boldsymbol{e}_i^T \neq \boldsymbol{H}\cdot\boldsymbol{e}_j^T; \forall \boldsymbol{e}_i, \boldsymbol{e}_j \in \boldsymbol{E}_+ | \boldsymbol{e}_i \neq \boldsymbol{e}_j \qquad (2)$$

$$\boldsymbol{H}\cdot\boldsymbol{e}_k^T \neq \boldsymbol{H}\cdot\boldsymbol{e}_j^T; \forall \boldsymbol{e}_k \in \boldsymbol{E}_\Delta | \boldsymbol{e}_j \in \boldsymbol{E}_+ \qquad (3)$$

Es decir, los síndromes utilizados para identificar los errores que se corregirán deben ser diferentes entre ellos, y los síndromes utilizados para identificar los errores que se detectarán deben ser diferentes de los síndromes utilizados para la corrección de errores. Es importante remarcar que, si una matriz de paridad **H** cumple con las condiciones (2) y (3), el código resultante tendrá la distancia mínima necesaria para corregir los errores incluidos en $\boldsymbol{E}_+$, y detectar los errores incluidos en $\boldsymbol{E}_\Delta$. Por ejemplo, si $\boldsymbol{E}_+$ incluye todos los errores de un bit, el código resultante tendrá al menos distancia $d = 3$. Si además incluye todos los errores dobles, tendrá distancia $d = 5$. Y si además detecta los errores triples, el código tendrá distancia $d = 6$.

Por tanto, el número de síndromes diferentes debe ser mayor que el número de vectores de error a corregir; y, además, deben existir síndromes diferentes para la detección de errores. Esta cantidad de síndromes viene marcada por el número de bits redundantes introducidos, que vienen determinados por la expresión $|\boldsymbol{E}_+| \leq 2^{n-k}$. Este número de bits redundantes también determinará la cobertura de error del ECC. Un elevado número de bits de código permitirá detectar/corregir un mayor número de errores. Sin embargo, también afectará a la sobrecarga introducida.

Mediante la metodología FUEC, para encontrar la matriz de paridad **H** que define un ECC, se utiliza un algoritmo recursivo que se inicia con una matriz vacía y va añadiendo todos los posibles valores para cada columna. Cuando se añade una columna, se comprueba si la matriz parcial obtenida cumple las condiciones (2) y (3). Si no es así, se descarta y se prueba un nuevo valor. Si cumple, se añade una nueva columna, repitiendo el proceso hasta que se completa la matriz y se cumplen las condiciones. Más detalles acerca del algoritmo, y de la metodología en general, pueden obtenerse en [21][22].

III. CÓDIGO LR DEC-TED [47, 32, 6]

Como ejemplo de diseño mediante la metodología FUEC [21][22], en esta sección presentamos un nuevo ECC de baja redundancia con parámetros [47, 32, 6], de ahora en adelante denominado LR DEC-TED (47, 32). El principal objetivo de este nuevo ECC es definir una buena cobertura de error (en este caso, la corrección de errores simples y dobles, indicado

en el nombre como DEC, y la detección de errores triples o TED) e introduciendo un bajo número de bits de código (de ahí el nombre LR, de *low redundancy*), con el fin de reducir la sobrecarga en la memoria.

En este sentido, nuestra propuesta añade 15 bits de código a una palabra de 32 bits de datos y alcanza una distancia de Hamming mínima de 6. Con estos parámetros, este código es capaz de corregir el 100% de los errores simples y dobles aleatorios, y de detectar el 100% de los errores triples aleatorios.

La Fig. 2 muestra la matriz de paridad **H** que define este código. A partir de esta matriz de paridad, es muy sencillo obtener las diferentes fórmulas para codificar y decodificar los bits de datos. Para implementar la corrección, una vez calculado el síndrome, hay que determinar qué bits son erróneos, es decir, los bits del vector de error estimado. De la misma forma, se obtiene una señal de Error No Recuperable (*Non-Recoverable Error*, NRE). En realidad, se trata de obtener distintas funciones lógicas a partir de los bits de síndrome. Una explicación detallada de este proceso se puede encontrar en [23].

$$\mathbf{H} = \begin{array}{c} \begin{matrix} C_0 C_1 C_2 & \cdots & C_{14} X_0 X_1 & \cdots & X_{31} \end{matrix} \\ \left[\begin{array}{ccccc} 1000000000000001010101010000010100000000100010 \\ 0100000000000001010100000101010000010101000000 \\ 0010000000000001010001010101000000000100010000 \\ 0001000000000001000100010001000100000001000100 \\ 0000100000000001000001000100010001000111011101 \\ 0000010000000000101000100011010111011000101000 \\ 0000001000000000100010011100100101110010000110 \\ 0000000100000000000111100000000111111000011001 \\ 0000000010000000001000110110001101011000100101 \\ 0000000001000000001010001001110101110010001010 \\ 0000000000100000100000100010001000100111110111 \\ 0000000000010000100010001000100010000001000101 \\ 0000000000001000101000101010100000000100010001 \\ 0000000000000100101010000010101000001010100000 \\ 0000000000000010101010101000001010000000010001 \end{array}\right] \end{array}$$

Fig. 2. Matriz de paridad H del código LR DEC-TED (47, 32).

Un aspecto a tener en cuenta es la adaptación de las capacidades de tolerancia a fallos de este ECC. Es decir, se puede utilizar únicamente la corrección, o la corrección y la detección. La detección de un error triple se utilizará para señalar un error irrecuperable y, por lo tanto, la única restricción es que su tiempo de cálculo debe ser más pequeño que el ciclo del reloj. La corrección de errores simples y dobles se debe realizar antes de que la memoria entregue los datos, y, por lo tanto, se agrega directamente al retardo del circuito. Consecuentemente, el impacto en el retardo del circuito se debe a la corrección de datos [24]. Por esta razón, presentamos por separado los datos de corrección y de corrección+detección en el estudio de la sobrecarga que se realiza en las siguientes secciones.

IV. Evaluación de los Codificadores y Decodificadores

En esta sección, presentamos la evaluación del código LR DEC-TED (47, 32). En este sentido, hemos comparado diferentes parámetros de nuestro código con una serie de ECC bien conocidos, con tamaño de datos iguales (32 bits) y capacidades de corrección similares (ECC capaces de corregir el 100% de los errores simples y dobles aleatorios). Específicamente, hemos usado los códigos OLS (55, 32) [25] y BCH (44, 32) [15]. Hemos seleccionado estos ECC porque implementan diferentes métodos en la corrección de errores, aunque su distancia $d = 5$ es inferior a la del código propuesto.

Para la comparación, hemos estudiado la redundancia introducida por los diferentes ECC, las coberturas de detección y corrección de errores, y la sobrecarga introducida por los circuitos de codificación y decodificación.

A. Redundancia

La Tabla I resume la redundancia introducida por los diferentes ECC estudiados en este trabajo. Podemos definir la redundancia de un ECC como la razón entre los bits de código añadidos y la longitud de la palabra de datos, tal y como se muestra en (4):

$$redundancy = \frac{(n-k)}{k} \times 100 \tag{4}$$

Tal y como se puede ver en la Tabla I, la redundancia más baja se corresponde con el código BCH (44, 32), mientras que el código OLS (55, 32) presenta el valor más alto. Estos dos ECC son códigos DEC (del inglés *Double Error Correction*). Es decir, son códigos capaces de corregir el 100% de los errores de 1 bit y de 2 bits aleatorios. Por otra parte, nuestra propuesta introduce una redundancia intermedia, pero aumentando la cobertura de error, ya que nuestra propuesta también es capaz de detectar el 100% de los errores de 3 bits aleatorios.

TABLA I
REDUNDANCIA DE LOS ECC ESTUDIADOS

ECC	Bis de datos (k)	Bits de código ($n-k$)	Redundancia ($n-k$)/k (%)
OLS [55, 32, 5]	32	23	71.88
BCH [44, 32, 5]	32	12	37.50
LR DEC-TED [47, 32, 6]	32	15	46.88

B. Coberturas de Detección y Recuperación

Mediante una herramienta de inyección de fallos basada en simulación desarrollada por los autores [11][26][27][28], hemos inyectado diferentes modelos de error en los circuitos de los distintos ECC. En particular, hemos inyectado errores simples y múltiples aleatorios con el fin de analizar las coberturas de error de los distintos ECC.

El esquema básico de la herramienta de inyección de fallos se muestra en la Fig. 3. Comparando los datos de entrada y salida, esta herramienta puede verificar si el error inyectado provoca una decodificación correcta o incorrecta. Cuando se detecta un error que no puede ser corregido, la herramienta activa la señal NRE.

Para calcular la cobertura de error, en los diferentes experimentos realizados hemos inyectado todos los errores posibles en todos los bits de la palabra de código de un modelo específico (simples y múltiples). Con esta inyección de fallos determinista se pueden calcular el número de errores detectados y/o corregidos con respecto al número total de errores inyectados. De esta forma, la fórmula (5) muestra

cómo se calcula la cobertura de corrección de errores, donde *Errors_Corrected* es el número de errores corregidos por el ECC, y *Errors_Injected* es el número de errores inyectados. Aunque los errores se inyectan tanto en los bits de código como en los bits de datos, consideramos que el error está corregido cuando los bits de datos son correctos tras la decodificación.

Por otra parte, la fórmula (6) muestra el cálculo de la cobertura de detección, donde *Errors_Detected* hace referencia al número de errores detectados, pero no corregidos, por el ECC.

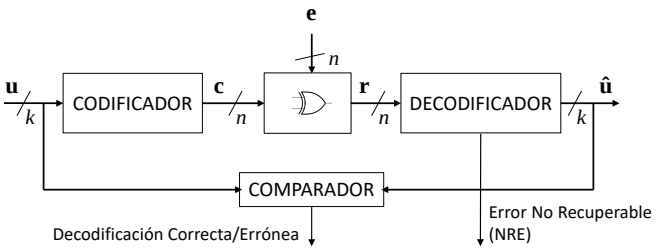

Fig. 3. Diagrama de bloques de la herramienta de inyección de fallos basada en simulación [11].

$$C_{correc} = \frac{Errors\_Corrected}{Errors\_Injected} \times 100 \qquad (5)$$

$$C_{detec} = \frac{Errors\_Corrected + Errors\_Detected}{Errors\_Injected} \times 100 \qquad (6)$$

Las Figs. 4 y 5 muestran las coberturas de corrección y detección de los tres ECC bajo estudio después de inyectar errores simples y múltiples aleatorios de tamaño de entre 2 y 8 bits. Este rango es representativo de los valores típicos de MCU en aplicaciones terrestres y críticas, como por ejemplo aplicaciones espaciales [5][6].

Como se esperaba, en la Fig. 4 se puede ver que todos los códigos pueden corregir el 100% de los errores simples, así como el 100% de los errores aleatorios dobles.

Por otra parte, la cobertura de corrección decrece para errores de tres o más bits. En concreto, para los códigos BCH (44,32) y LR DEC-TED (47, 32), esta bajada es bastante acentuada. Esto se debe a la baja redundancia de estos códigos, que provoca un menor número de síndromes utilizables. Es decir, los síndromes disponibles en los códigos BCH (44,32) y LR DEC-TED (47, 32) se utilizan de una forma más efectiva dentro del rango de errores esperados. Fuera de este rango, la capacidad de corrección se degrada rápidamente.

Con respecto a la cobertura de detección, y tal y como se puede ver en la Fig. 5, los códigos OLS (55, 32) y BCH (44, 32) pueden detectar todos los errores aleatorios de 2 bits (que son los errores que corrigen estos ECC), mientras que nuestra propuesta puede detectar también el 100% de los errores aleatorios de 3 bits. Además, y al contrario que los otros códigos, la cobertura de detección de nuestra propuesta se estabiliza para errores de mayor tamaño, con una leve oscilación debida al uso de paridades solapadas para el cálculo del síndrome. De la misma forma que un bit de paridad detecta el 100% de errores en un número impar de bits, y el 0% de errores en un número par de bits, esa oscilación se mantiene cuando se combinan varias paridades. De esta forma, nuestro LR DEC-TED (47, 32) no sólo detecta el 100% de los errores triples, sino que además detecta más del 90% de los errores mayores de tres bits. Sin embargo, estos porcentajes no se obtienen de forma gratuita. Tal y como veremos a continuación, el circuito decodificador de nuestra propuesta presenta una mayor complejidad, lo que acarrea una mayor sobrecarga.

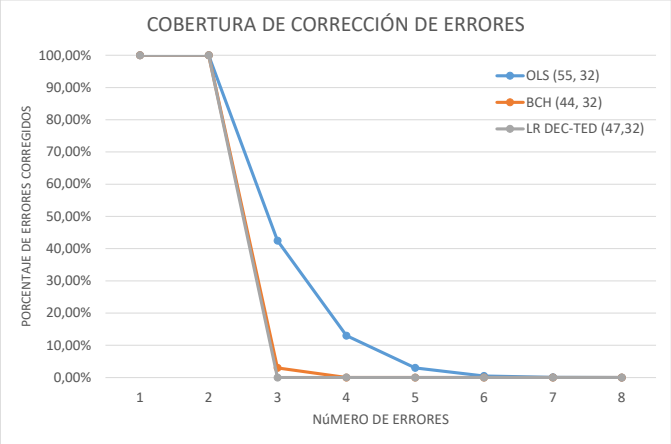

Fig. 4. Cobertura de corrección de errores aleatorios.

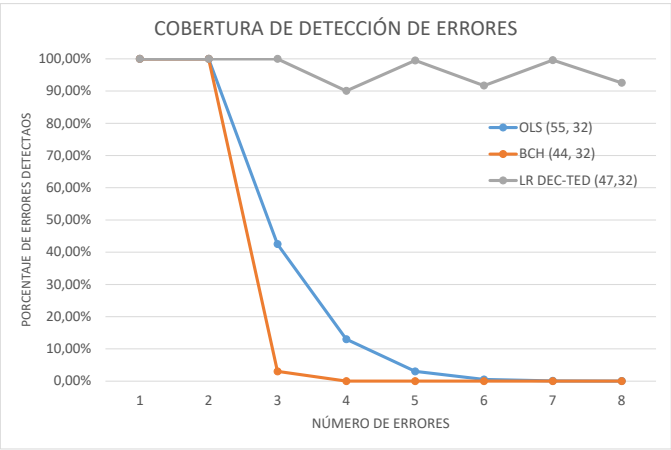

Fig. 5. Cobertura de detección de errores aleatorios.

C. Sobrecarga de los circuitos codificadores y decodificadores. Síntesis con celdas estándar (Standard Cells)

En este apartado se presentan los resultados obtenidos durante la síntesis, mediante el software CADENCE [29], de todos los circuitos codificadores y decodificadores. Esta síntesis se ha utilizado para estudiar la sobrecarga de dichos circuitos con respecto al área de silicio, potencia consumida y retardo. Para ello, hemos modelado en VHDL todos los circuitos y hemos realizado una síntesis lógica en tecnología de 45nm, usando la librería NanGate FreePDK45 [30][31] mediante celdas estándar basadas en las reglas de diseño SCMOS. Las condiciones del voltaje de alimentación y de temperatura han sido de 1.1V y 25ºC, respectivamente. Además, se ha fijado la frecuencia de funcionamiento en 300 MHz. La Tabla II resume los datos obtenidos.

Sobrecarga en área de silicio

El área ocupada por los diferentes circuitos se ha medido en micrómetros cuadrados (μm^2). Como se puede ver, el área de los codificadores es más pequeña que el área de los decodificadores, debido a que estos últimos elementos deben calcular el síndrome, localizar el error y corregirlo.

El área de los codificadores es bastante similar en los tres códigos, mientras que el área de los decodificadores sí presenta grandes diferencias. En primer lugar, podemos ver que, aunque el código OLS (55, 32) presenta la mayor redundancia, la sobrecarga que presenta el decodificador es la menor. Esto se debe a que la implementación de este ECC se realiza mediante circuitos muy sencillos basados en votadores mayoritarios, lo que provoca esta sobrecarga tan baja.

Por otro lado, podemos ver que el circuito de detección en el código LR DEC-TED (47, 32) ocupa un área de silicio muy grande. Al quitar esta detección, vemos que se consigue una sobrecarga más baja que la del código BCH (44, 32).

Lo que podemos comprobar es que un número bajo de bits de código, junto con una mayor capacidad de tolerancia a fallos, requiere una mayor utilización de área. Esto se debe a que la matriz de paridad **H** que define el ECC presenta un número elevado de 1s, lo que causa que las fórmulas para generar los bits de paridad y de síndrome sean más complejas. Por estas razones, nuestra propuesta presenta la mayor utilización de área en el decodificador cuando se corrigen y detectan errores, pues el LR DEC-TED (47, 32) presenta un bajo número de bits de código junto a una mayor tolerancia a fallos (hay que recordar que este código es capaz de corregir el 100% de los errores simples y dobles, y de detectar también el 100% de los errores triples).

Sobrecarga en el consumo de potencia

El consumo de potencia de los diferentes codificadores y decodificadores se ha medido en milivatios (mW). Como se puede ver, sigue la misma tendencia que la sobrecarga en el área. Esto es, los decodificadores consumen más que los codificadores.

El código LR DEC-TED (47, 32) presenta el mayor consumo de potencia, principalmente en el decodificador. Tal y como se acaba de comentar, este ECC tiene un bajo número de bits de paridad, y una buena capacidad de tolerancia a fallos. No obstante, casi la mitad de ese consumo se utiliza para implementar la detección de errores triples, cobertura no soportada por los otros códigos. Si solo se implementa la corrección, el consumo es sensiblemente inferior al del código BCH. Por otra parte, el código OLS (55, 32) muestra los valores más bajos, debido a que el aumento de redundancia simplifica su implementación, como ya sucedía con el área de silicio ocupada.

Sobrecarga en el retardo

El retardo de los codificadores y decodificadores mostrados en la Tabla II se ha medido en nanosegundos (ns). Como se puede ver, los codificadores presentan un retardo menor que los decodificadores al ser circuitos menos complejos. Una rápida codificación y decodificación implica ciclos de escritura y lectura rápidos. En este sentido, el retardo más crítico es el de la corrección de los datos en el decodificador, que afecta al tiempo de acceso en el ciclo de lectura [24].

En este sentido, el código OLS (55, 32) presenta la codificación y decodificación más rápida, pues su corrección, implementada por votación, genera circuitos muy simples. Por otro lado, el decodificador más lento corresponde al código BCH (44, 32), por la complejidad del circuito de corrección, que logra una buena cobertura de error con una baja redundancia. El código LR DEC-TED (47, 32) tiene un tiempo de decodificación intermedio, si solo se considera la corrección de datos. Como se ha explicado antes, este parámetro es el que puede afectar a las prestaciones del procesador.

Al considerar la cobertura adicional, y como se esperaba, la señal de detección de errores triples introduce un mayor retardo. No obstante, este retardo no afecta a las prestaciones del sistema, siempre que sea inferior al tiempo de ciclo [24]. Aunque se implemente la detección, el retardo de la corrección no varía, por lo que el impacto sobre el sistema de nuestra propuesta sigue siendo inferior al producido por el código BCH.

V. Evaluación de la Inclusión de los ECC en un Microprocesador RISC

Con los datos obtenidos en la sección anterior, parece claro que el código OLS (55, 32) es el que presenta la menor sobrecarga en los circuitos de codificación y decodificación. Sin embargo, ¿se cumple esta predicción cuando este código se añade a un procesador? En esta sección vamos a responder a esta cuestión, estudiando cuál es la sobrecarga producida cuando se introducen los ECC estudiados en la sección anterior en un microprocesador RISC.

A. Microprocesador Plasma

El microprocesador Plasma [32] utilizado en este trabajo es un pequeño microprocesador sintetizable RISC de 32 bits. Tiene una arquitectura MIPS, con un pipeline de tres etapas. El modelo VHDL del Plasma está descrito en los niveles de abstracción lógico y RT. La Fig. 6 muestra su diagrama de bloques.

TABLA II
SOBRECARGA DE LOS ECC ESTUDIADOS

		OLS (55, 32)	BCH (44, 32)	LR DEC-TED (47,32) (sin detección)	LR DEC-TED (47,32) (con detección)
CODIFICADORES	AREA (μm^2)	316	316	303	303
	CONSUMO (mW)	0.034	0,054	0,045	0,045
	RETARDO (ns)	0.160	0,396	0,337	0,337
DECODIFICADORES	AREA (μm^2)	812	4635	3700	7761
	CONSUMO (mW)	0.160	0,977	0,755	1,499
	RETARDO (ns)	0.421	1,567	1,263	1,729

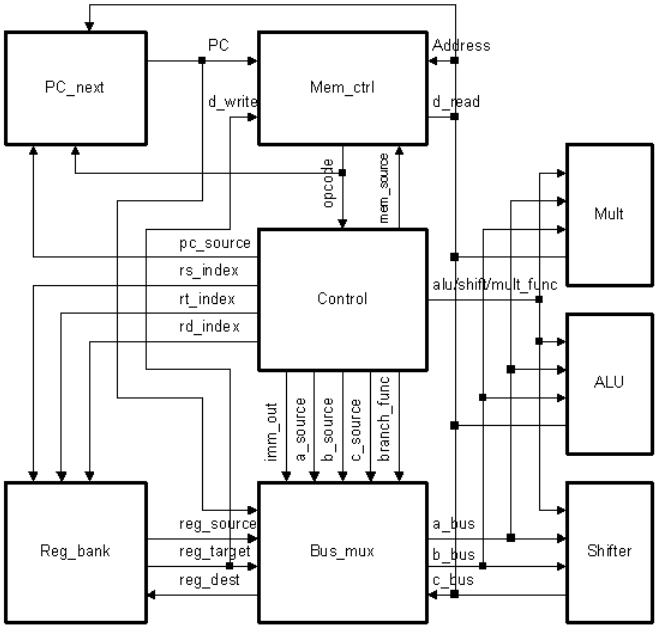

Fig. 6. Diagrama de bloques del microprocesador Plasma [32].

El microprocesador Plasma incorpora un módulo RAM, cuyo diagrama de bloques original se muestra en la Fig. 7. Este módulo se va a proteger añadiendo los ECC vistos en secciones anteriores. En el modelo sin protección hemos configurado el módulo RAM con un tamaño de 512x32, suficiente para la ejecución de un algoritmo de multiplicación de matrices 4x4, que ha sido la carga de trabajo elegida. Con esta carga de trabajo, podemos comprobar si el resultado de su ejecución es el mismo y con la misma temporización antes y después de añadir el ECC correspondiente. Hemos seleccionado esta carga de trabajo porque presenta un uso balanceado de todos los bloques del microprocesador.

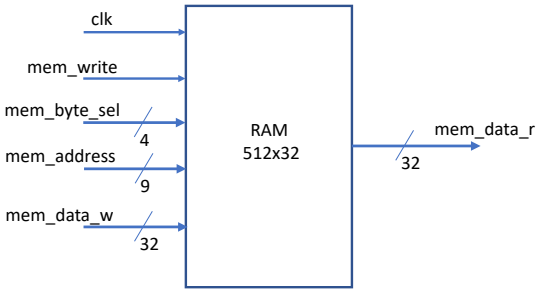

Fig. 7. Diagrama de bloques original del módulo de memoria RAM del microprocesador Plasma [32].

Por otra parte, se ha modificado este bloque de memoria para que pueda incluir un ECC. Estas modificaciones se muestran en la Fig. 8. La idea ha sido generar un nuevo módulo de memoria RAM, con las mismas entradas y salidas que el módulo original, pero incluyendo la circuitería de un determinado ECC. Con este esquema, es muy sencillo cambiar de un ECC a otro. Basta con reemplazar los bloques del codificador y decodificador, y ajustar el tamaño de palabra de la memoria RAM.

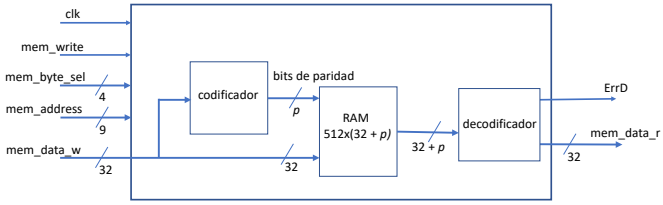

Fig. 8. Diagrama de bloques modificado del módulo de memoria RAM del microprocesador Plasma [32].

B. Resultados de la síntesis

Al igual que con los ECC, también hemos sintetizado diferentes modelos de microprocesador Plasma con y sin ECC. De esta forma, hemos podido comparar la sobrecarga introducida por los diferentes ECC. Hemos utilizado las mismas condiciones que en la síntesis de los ECC, esto es, hemos usado la librería NanGate FreePDK45 para realizar una síntesis lógica en 45nm, con unas condiciones de voltaje de alimentación y de temperatura de 1.1V y 25º respectivamente. También hemos fijado la frecuencia de funcionamiento de todos los modelos del microprocesador a 300 y 600 MHz. En trabajos futuros, se pretende estudiar con más detalle las características temporales del modelo y cómo es afectado por la inclusión de un ECC. En la Tablas III y IV se puede ver un resumen de los resultados de esta síntesis.

Sobrecarga en el área de silicio

Como se puede observar en las Tablas III y IV, el microprocesador Plasma con el OLS (55, 32) es el que presenta una mayor sobrecarga (supera el 55% con respecto al microprocesador Plasma original en ambas síntesis), mientras que el Plasma con el BCH (44, 32) es el que menos sobrecarga presenta (alrededor de un 33% más). Estos resultados nos indican que, más que los circuitos codificadores y decodificadores, el principal factor en el aumento de la sobrecarga del área es el número de bits de código. En este sentido, por ejemplo, para implementar el código OLS (55, 32), se ha cambiado la memoria original de 512x32 por un módulo de memoria de 512x55 (un aumento de casi un 72%).

En el caso de nuestra propuesta, el código LR DEC-TED (47, 32), la sobrecarga que presenta varía entre un 39% si no hay detección y un 43% cuando sí se detectan errores triples. En el primer caso, a igualdad de cobertura de errores, nuestra propuesta reduce notablemente el área utilizada por el código OLS, mientras que reduce la sobrecarga temporal del decodificador con un incremento mínimo en el área ocupada. En el segundo caso, el incremento de área puede ser asumible si se necesita la detección de errores triples.

TABLA III
SOBRECARGA DEL MICROPROCESADOR PLASMA TOLERANTE A FALLOS (FRECUENCIA 300 MHz)

	Área (μm^2)	**Consumo estático (mW)**	**Consumo dinámico (mW)**	**Consumo Total (mW)**
PLASMA	122450	1.182	9.999	11.181
PLASMA + OLS (55, 32)	191053	1.843	9.989	11.831
PLASMA + BCH (44, 32)	162621	1.582	10.053	11.635
PLASMA + LR DEC-TED (47, 32) (sin detección)	170484	1.652	10.001	11.653
PLASMA + LR DEC-TED (47, 32) (con detección)	174543	1.701	10.001	11.702

TABLA IV
SOBRECARGA DEL MICROPROCESADOR PLASMA TOLERANTE A FALLOS (FRECUENCIA 600 MHz)

	Área (μm^2)	**Consumo estático (mW)**	**Consumo dinámico (mW)**	**Consumo Total (mW)**
PLASMA	123104	1.205	18.124	19.329
PLASMA + OLS (55, 32)	191700	1.867	18.257	20.125
PLASMA + BCH (44, 32)	163279	1.606	18.156	19.762
PLASMA + LR DEC-TED (47, 32) (sin detección)	171138	1.676	18.151	19.828
PLASMA + LR DEC-TED (47, 32) (con detección)	175198	1.725	18.151	19.876

Sobrecarga en el consumo de potencia

Como se puede ver en las Tablas III y IV, el consumo estático de potencia es el que presenta el mayor incremento en el consumo de energía (en el código OLS (55, 32) presenta un aumento que oscila entre el 54% y el 56%, en el código LR DEC-TED (47, 32) con detección el incremento no llega al 44% en ambos casos, casi un 40% de aumento en el código LR DEC-TED (47, 32) sin detección, y en el código BCH (44, 32) crece casi un 34%). Este resultado es provocado por la inclusión de los circuitos de codificación y decodificación, y sobre todo, por las celdas extra en memoria necesarias para almacenar los bits redundantes. Por otro lado, el consumo dinámico es casi idéntico para todos los modelos en ambas síntesis, siendo el código BCH (44, 32) el que presenta el mayor consumo con la síntesis a 300 MHz, mientras que es código OLS (55, 32) el que presenta mayor consumo dinámico en la síntesis a 600 MHz. Esto conlleva que el aumento del consumo total de energía de los modelos con ECC incluidos sea menor del 6% con respecto al microprocesador Plasma original en la síntesis a 300 MHz, mientras que el consumo global sube entre un 2% y un 4% en la síntesis a 600 MHz.

Al igual que pasaba con la sobrecarga en el área de silicio, el microprocesador Plasma con el código OLS (55, 32) es el que presenta el mayor consumo en ambas síntesis, mientras que con el código BCH (44, 32) introduce el menor consumo. El código LR DEC-TED (47, 32) presenta un aumento intermedio en ambas síntesis, pero con una mayor tolerancia a fallos. Como sucedía con el área, los incrementos de consumo son muy asequibles considerando la mejora en la respuesta temporal (respecto del código BCH) y la mayor cobertura de errores. Otro dato interesante que se puede observar es que, a mayor frecuencia de funcionamiento, menor es la sobrecarga en el consumo de energía de nuestra propuesta.

VI. CONCLUSIONES

En este trabajo se ha presentado un estudio sobre el diseño, implementación y evaluación de un código de corrección de errores cuya principal característica es su baja redundancia y una gran capacidad de tolerancia a fallos. En concreto, en este trabajo se ha propuesto un ECC que añade 15 bits de código a una palabra de datos de 32 bits, y que es capaz de corregir todos los errores simples y dobles aleatorios, además de detectar todos los errores aleatorios de tres bits.

Hemos evaluado nuestra propuesta con respecto a otros ECC que también pueden tolerar errores múltiples, como son los códigos OLS y BCH.

En primer lugar, hemos medido la cobertura de error de los tres códigos, calculando las coberturas de corrección y detección después de realizar diferentes experimentos de inyección de fallos.

A continuación, hemos sintetizado los circuitos codificadores y decodificadores en tecnología de 45nm y celdas estándar, con el fin de estimar la sobrecarga que introducirían dichos circuitos. La alta tolerancia a fallos de nuestra propuesta, junto con su baja redundancia, provoca que en esta primera síntesis, nuestra propuesta presente la mayor sobrecarga.

El siguiente paso ha consistido en el estudio del impacto de la introducción de estos ECC en el modelo de un microprocesador RISC, con el fin de corroborar los resultados de sobrecarga obtenidos en la primera parte.

Como se esperaba, la inclusión de un ECC introduce un aumento en el área de silicio necesaria, provocado por la circuitería del codificador y decodificador y por el incremento en el tamaño de la memoria RAM para almacenar los bits redundantes. Por otro lado, el incremento en el área también genera un incremento en el consumo de potencia, principalmente en el estático.

También hemos comprobado que el factor determinante en la sobrecarga de un microprocesador cuando se le añade un ECC es el número de bits redundantes. En este sentido, el código OLS (55, 32), que en la primera parte de este trabajo presentaba los mejores números con respecto a la sobrecarga, al incluirlo en el microprocesador provoca la mayor sobrecarga total.

Con respecto a nuestra propuesta, hemos visto que la sobrecarga que produce es aceptable teniendo en cuenta que también aumenta la capacidad de tolerancia a fallos con respecto a los otros ECC. Es decir, debida a su alta capacidad de tolerancia a fallos, nuestra propuesta presenta una mayor sobrecarga en el área de silicio, potencia consumida y retardo principalmente del circuito decodificador. Sin embargo, esta sobrecarga se compensa en gran medida por su baja redundancia, lo que reduce la sobrecarga de la memoria, tal y como se ha mostrado en las dos síntesis realizadas.

En el futuro, queremos seguir desarrollando ECC con diferentes coberturas de error, mientras disminuimos la sobrecarga producida. También queremos continuar estudiando el impacto de incluir diferentes tipos de ECC en diferentes modelos de microprocesadores. Además, también queremos estudiar con más detalle cómo afecta la inclusión de un ECC a los tiempos de lectura/escritura. Por otra parte, también proyectamos extender los experimentos de inyección de fallos a los modelos completos de microprocesador más ECC.

REFERENCIAS

[1] The International Technology Roadmap for Semiconductors 2013. Accessed February 3, 2021. [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] J. Gracia-Moran, L.J. Saiz-Adalid, D. Gil-Tomás, and P.J. Gil-Vicente, “Improving Error Correction Codes for Multiple Cell Upsets in Space Applications”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26(10), pp. 2132-2142, October 2018.

[12] S. Lin and D.J. Costello, Error Control Coding, 2nd edition, Pearson-Prentice Hall, 2004.

[13] 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.

[14] 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, vol. 17(4), pp. 698-707, December 2017.

[15] R. Naseer and J. Draper, “DEC ECC design to improve memory reliability in Sub-100nm technologies”, 2008 15th IEEE International Conference on Electronics, Circuits and Sys-tems, pp. 586-589, August 2008.

[16] C. Argyrides, D.K. Pradhan and T. Kocak, “Matrix codes for reliable and cost efficient memory chips”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19 (3), pp.420–428, March 2011.

[17] K. Kato and S. Choomchuay, “An Analysis of Time Domain Reed Solomon Decoder with FPGA Implementation”, IEICE Transactions on Information and Systems, Volume E100.D, Issue 12, Pages 2953-2961, December 2017.

[18] N. A. Rodríguez-Olivares, A. Gómez-Hernández, L. Nava-Balanzar, H. Jiménez-Hernández and J. A. Soto-Cajiga, "FPGA-Based Data Storage System on NAND Flash Memory in RAID 6 Architecture for In-Line Pipeline Inspection Gauges," IEEE Transactions on Computers, vol. 67, no. 7, pp. 1046-1053, July 2018.

[19] 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.

[20] J. H. van Lint, Introduction to Coding Theory, Springer, 1982.

[21] L.J. Saiz-Adalid, P.J. Gil-Vicente, J.C. Ruiz, D. Gil-Tomás, J.C: Baraza and J. Gracia-Morán, “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.

[22] L.J. Saiz-Adalid, “Fallos intermitentes: análisis de causas y efectos, nuevos modelos de fallos y técnicas de mitigación”, Tesis doctoral. Universitat Politècnica de València, 2015. Accessed February 3, 2021. [Online]. http://hdl.handle.net/10251/59452

[23] L.-J. Saiz-Adalid, J. Gracia-Morán, D. Gil-Tomás, J.-C. Baraza-Calvo and P.-J. Gil-Vicente, “Reducing the Overhead of BCH Codes: New Double Error Correction Codes”, Electronics 2020, 9, 1897.

[24] P. Reviriego, S. Pontarelli, J.A. Maestro, M. Ottavi, “A Method to Construct Low Delay Single Error Correction Codes for Protecting Data Bits Only”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 3, pp. 479-483, March 2013.

[25] S. Liu, L. Xiao and Z. Mao, “Extend orthogonal Latin square codes for 32-bit data protection in memory applications”, Microelectronics Reliability, vol. 63, pp. 278-283, 2016.

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

[27] J. Gracia-Morán, L.J. Saiz-Adalid, J.C. Baraza-Calvo, P.J. Gil-Vicente, “Correction of Adjacent Errors with Low Redundant Matrix Error Correction Codes”, 8th Latin-American Symposium on Dependable Computing (LADC 2018), pp. 107-114, October 2018.

[28] J. Gracia-Morán, L.J. Saiz-Adalid, D. Gil-Tomás, J.C. Baraza-Calvo, P.J. Gil-Vicente, “Mejora de un Código de Corrección de Errores para tolerar fallos adyacentes bidimensionales”, IV Jornadas de Computación Empotrada y Reconfigurable (JCER2019), Jornadas SARTECO, pp. 600-605, September 2019.

[29] Cadence: EDA Tools and IP for System Design Enablement. Accessed February 3, 2021. [Online]. https://www.cadence.com/

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

[31] NanGate FreePDK45 Open Cell Library. Accessed February 3, 2021. [Online]. https://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[32] Plasma CPU model. Accessed February 3, 2021. [Online]. Available: https://opencores.org/projects/plasma

Joaquín Gracia-Morán es Diplomado (1995), Licenciado (1997) y Doctor (2004) en Ingeniería Informática por la Universitat Politècnica de València (UPV), España. Actualmente es profesor titular en la

UPV, en el Departamento de Informática de Sistemas y Computadores (DISCA). Es miembro de la línea de investigación en Sistemas Tolerantes a Fallos (STF) del Instituto ITACA de la UPV. Sus intereses de investigación incluyen el diseño y la implementación de sistemas digitales, el diseño y la validación de sistemas tolerantes a fallos, y la inyección de fallos basada en VHDL.

Luis-J. Saiz-Adalid es Licenciado (1995) y Doctor (2015) en Ingeniería Informática por la Universitat Politècnica de València (UPV), España. Después de 15 años en la industria (IBM, 1995-Celestica, 1998), actualmente es profesor Contratado Doctor en el Departamento de Informática de Sistemas y Computadores (DISCA) de la UPV. Es miembro de la línea de investigación en Sistemas Tolerantes a Fallos (STF) del Instituto ITACA de la UPV. Sus intereses de investigación incluyen el diseño y la implementación de sistemas digitales, el diseño y la validación de sistemas tolerantes a fallos, y el diseño de códigos de corrección de errores.

J.-Carlos Baraza-Calvo es Licenciado (1993) y Doctor (2003) en Ingeniería Informática por la Universitat Politècnica de València (UPV), España. Actualmente es profesor titular en el Departamento de Informática de Sistemas y Computadores (DISCA) de la UPV. Es miembro de la línea de investigación en Sistemas Tolerantes a Fallos (STF) del Instituto ITACA de la UPV. Sus intereses de investigación incluyen el diseño y la implementación de sistemas digitales, el diseño y la validación de sistemas tolerantes a fallos, y la inyección de fallos basada en VHDL.

Daniel Gil-Tomás es Licenciado (1985) en Ciencias Físicas, especialidad Electricidad y Electrónica por la Universitat de València, España, y Doctor (1999) en Ingeniería Informática por la Universitat Politècnica de València (UPV), España. Actualmente es profesor titular en la UPV, en el Departamento de Informática de Sistemas y Computadores (DISCA). Es miembro de la línea de investigación en Sistemas Tolerantes a Fallos (STF) del Instituto ITACA de la UPV. Sus intereses de investigación incluyen el diseño y la validación de sistemas tolerantes a fallos, la física de la confiabilidad y la confiabilidad de las nanotecnologías emergentes.

Pedro-J. Gil-Vicente es Diplomado (1979) en Ingeniería Electrónica por la Universitat Politècnica de Catalunya (UPC), Tarragona, España, Licenciado (1985) en Ingeniería Industrial y Doctor (1992) en Ingeniería Informática por la Universitat Politècnica de València (UPV), España. Actualmente es profesor catedrático en el Departamento de Informática de Sistemas y Computadores (DISCA), donde ha sido director. Ahora dirige la línea de investigación en Sistemas Tolerantes a Fallos (STF) del Instituto ITACA de la UPV. Su investigación se centra en el diseño y la validación de sistemas distribuidos tolerantes a fallos en tiempo real, la validación de la confiabilidad mediante inyección de fallos, el diseño y verificación de sistemas integrados, y la evaluación comparativa de la confiabilidad y la seguridad. Ha impartido cursos sobre tecnología informática, diseño digital, redes informáticas y sistemas tolerantes a fallos.

El profesor Gil es autor de más de 150 trabajos de investigación sobre estos temas. También es miembro del Comité de Programa en las principales conferencias sobre confiabilidad y seguridad: *IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN), *European Dependable Computing Conference* (EDCC) y *Latin American Symposium on Dependable Computing* (LADC). Ha sido revisor en revistas internacionales y congresos relacionados con la confiabilidad y la seguridad. Fue presidente general de la conferencia EDCC-8, celebrada en Valencia en abril de 2010 y copresidente general de la 50ª *IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2020), organizado online en Valencia en Junio de 2020.