# Acelerador hardware de simulaciones de códigos LDPC basado en FPGAs

F. Angarita, A. Perez-Pascual, T. Sansaloni y J. Valls [1]

*Resumen*— **En este artículo se presenta un simulador hardware de códigos de comprobación de paridad de baja densidad (LDPC) el cual permite reducir considerablemente los tiempos de simulación comparado con un entorno completamente software. Se ha utilizando la finalización anticipada en el decodificador LDPC y se ha paralelizado la generación de datos para evitar tiempos muertos y así obtener una máxima eficiencia hardware que se traduce en máxima tasa de datos. El simulador ha sido probado con el código LDPC del estándar IEEE 802.3an sobre un dispositivo Altera Cyclone IV montado en una placa de desarrollo de bajo coste, llegando a conseguir una tasa de datos superior a los 600 Mbps con un reloj de 150 MHz y una tasa de error de bit de $10^{-12}$ en menos de 5 días.**

*Palabras clave*— **códigos LDPC, FPGA, emulador hardware.**

## I. Introducción

Los códigos de comprobación de paridad de baja densidad (LDPC) son de gran interés debido a su gran capacidad de corrección, cercana al límite de Shannon. En la actualidad estos códigos han sido adoptados en varios estándares de la industria como la transmisión por satélite de televisión digital (DVB-S2) [1], red de cableado residencial (G.hn/G.9660) [2], red inalámbrica de área local Wi-Fi (IEEE 802.11n) [3], red inalámbrica de acceso metropolitano de banda ancha WiMAX (IEEE 802.16e) [4] y 10-Giagabit Ethernet sobre par trenzado no blindado (IEEE 802.3an) [5]. Además, son utilizados en una gran variedad de aplicaciones, como el almacenamiento de datos, comunicaciones vía satélite y los sistemas de comunicaciones ópticas.

En la práctica las prestaciones de los códigos LDPC se miden mediante el método de simulación de Monte Carlo de acuerdo al esquema de la figura 1. Un paquete de datos binarios es codificado, modulado y transmitido por un canal con ruido aditivo Gaussiano (AWGN). El paquete es recibido y decodificado. La trama de datos decodificada es comparada con la transmitida para así calcular el número de bits erróneos y si éste es mayor a cero, el paquete se considera erróneo. Para obtener un resultado estadísticamente bueno se requiere que el número de paquetes transmitido sea tal que el número de paquetes erróneos sea alto (al menos 100). Por lo tanto, para alcanzar una tasa de error de paquete de $10^{-10}$, como requiere el estándar IEEE 802.3an, es necesario simular por lo menos $10^{12}$ paquetes. Realizar esta cantidad de operaciones en un ordenador de altas prestaciones requiere de varios meses de computación.

Los emuladores hardware basados en FPGA permiten reducir considerablemente los tiempos de simulación. En este trabajo se presenta un emulador hardware para el código del estándar IEEE 802.3an con un bajo coste hardware y alta tasa de decodificación. El diseño ha sido optimizado para una FPGA de bajo coste, alcanzando una tasa de error de bit de $10^{-12}$ en 5 días.

[1]Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universidad Politécnica de Valencia, e-mail: faanpre@doctor.upv.es, {asperez,tmsansal,jvalls}@eln.upv.es.

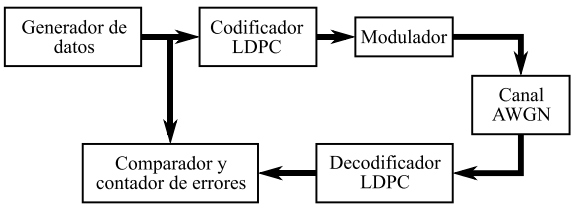


Fig. 1. Esquema de simulación de los códigos LDPC.

Es artículo esta organizado de la siguiente manera. En la sección II se introducen los códigos de comprobación de paridad de baja densidad y los algoritmos básicos de de decodificación. La sección III se describe la arquitectura del simulador, detallando los bloques que componen el emulador hardware y describiendo la parte software. En la sección IV se dan resultados de la implementación hardware así como los de simulación para el caso del código LDPC del estándar IEEE 802.3an. Por último, las conclusiones y el trabajo futuro se presentan en la sección V.

## II. Códigos LDPC y algoritmos de decodificación

Un código LDPC está definido por el espacio nulo de la matriz de comprobación de paridad $\mathbf{H}$ de tamaño $M \times N$, donde $M$ es el número de comprobaciones de paridad y $N$ el número de bits del código. La representación gráfica de los códigos LDPC se hace mediante un grafo bipartito denominado grafo de Tanner [6], el cual utiliza $\mathbf{H}$ como matriz incidente, con $M$ nodos de comprobación $C_m$ por un lado y $N$ nodos de bit o variables $V_n$ por el otro. Un nodo de comprobación $C_m$ y un nodo de bit $V_n$ están conectados entre sí cuando el elemento $\mathbf{H}_{m,n}$ de la matriz $\mathbf{H}$ es igual a 1. El número de unos por fila o columna se define como peso y se dice que una matriz es regular cuando el peso de columnas y filas es uniforme, en caso contrario se denomina irregular.

Los códigos LDPC generados aleatoriamente presentan altas prestaciones, pero su implementación hardware resulta demasiado compleja [7]. Otro tipo de códigos, denominados estructurados, consiguen prestaciones similares a los aleatorios y pueden ser diseñados para que la implementación hardware sea eficiente reduciendo su complejidad [8]. Los códigos estructurados se generan dividiendo la matriz de comprobación de paridad $\mathbf{H}$, en $M_b \times N_b$ submatrices cuadradas, donde cada submatriz tiene un tamaño $z \times z$. Por tanto, el valor de $M$ y $N$ de la matriz $\mathbf{H}$ resultante es $M_b \cdot z$ y $N_b \cdot z$, respectivamente. En la figura 2 se muestra un ejemplo de matriz $\mathbf{H}$ de un código regular estructurado cuyo tamaño es

$20 \times 12$ con $z = 4$. Otros ejemplos de códigos estructurados son los códigos cuasi-cíclicos (QC) [9], irregulares de repetición y acumulación (IRA) [10], y los códigos basados en Reed-Solomon (RS) [11].

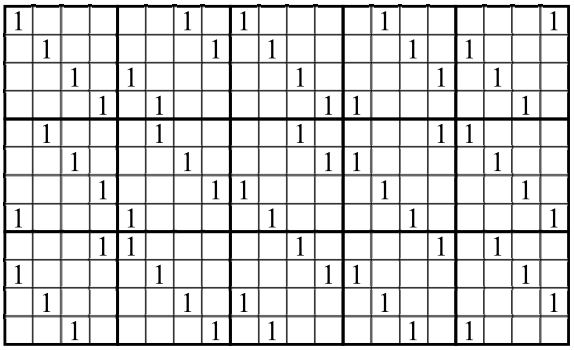

Fig. 2. Matriz de paridad $\mathbf{H}$ de un código LDPC regular estructurado.

Los algoritmos del tipo de intercambio de mensajes [12] son los más utilizados para realizar la decodificación de códigos LDPC. Este tipo de algoritmos consisten en el intercambio de mensajes de forma iterativa entre los nodos de comprobación $C_m$ y los nodos de bit $V_n$, a través de las conexiones del grafo de Tanner. Entre estos, el algoritmo Sum-Product (SP) [13] es la referencia debido a que posee unas altas prestaciones en términos de capacidad correctora, sin embargo, su complejidad computacional es elevada debido a que hace uso de funciones no lineales [14]. Con el fin de reducir la complejidad computacional se propone el algoritmo Min-Sum (MS) [15] en el cual se aproxima la función no lineal al cálculo del valor mínimo incurriendo en pérdidas de prestaciones. Sin embargo, en [16] se proponen dos modificaciones, el algoritmo Min-Sum escalado ($\alpha$MS) y Min-Sum con *offset* ($\beta$MS), cuyas prestaciones son similares a las del algoritmo SP y su complejidad es ligeramente mayor a la del MS.

En el pseudocódigo 1 se describe el algoritmo $\alpha$MS donde $N_m = \{n : \mathbf{H}_{m,n} = 1\}$ denota al conjunto de nodos de bit adyacentes a un nodo de comprobación $C_m$ y $M_n = \{m : \mathbf{H}_{m,n} = 1\}$ al conjunto de nodos de comprobación adyacentes a un nodo de bit $V_n$. El conjunto de nodos de bit $N_m$ excluyendo propio nodo $V_n$ se denota como $N_{m\backslash n}$ y el conjunto de nodos de comprobación $M_n$ excluyendo al nodo $C_m$ como $M_{n\backslash m}$. Se asume que la palabra código binaria $\boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ es transmitida por un canal AWGN de media cero y varianza $\sigma^2$, usando la modulación de fase binaria (BPSK). La secuencia de símbolos recibida se define como $\boldsymbol{r} = (r_1, r_2, \ldots, r_N)$ y las fiabilidades intrínsecas del canal como $\boldsymbol{l} = (l_1, l_2, \ldots, l_N)$, donde $l_n = 2 \cdot r_n / \sigma^2$ es la razón de verosimilitud (LLR) del símbolo $r_n$. Además, se define $\lambda_{n,m}$ como el mensaje enviado por un nodo de bit $V_n$ a un nodo comprobación $C_m$ y $\mu_{m,n}$ como el mensaje de un nodo de comprobación $C_m$ a un nodo de bit $V_n$. El índice de iteración se denota cómo $i$ y está dentro del rango $1, \ldots, I_{max}$, donde $I_{max}$ es el número máximo de iteraciones. El término $\Gamma_{m,n}^{(i)}$ es la actualización de paridad y se define como, $\Gamma_{m,n}^{(i)} = \prod_{n' \in N_{m\backslash n}} \text{sign}(\lambda_{n',m}^{(i-1)})$.

En el pseudocódigo 1 se observa que el proceso iterativo puede terminar antes de alcanzar el número máximo de iteraciones $I_{max}$ si se cumple la condición de parada. Esta condición se determina con las ecuaciones de paridad: la palabra $z$ es una palabra código válida cuando el vector $p$ tiene un valor nulo en todos sus elementos. El uso de la paridad como condición de parada se denomina finalización anticipada y permite acelerar la tasa media de decodificación debido a que el número de iteraciones es bajo cuando los símbolos recibidos contienen pocos errores, en otras palabras, el número de iteraciones requeridas es bajo cuando la relación señal a ruido (SNR) del canal AWGN es alta.

**Pseudocódigo 1** Algoritmo $\alpha$MS

1: *– iniciación –*
2: para $n \in \{1, \ldots, N\}$ y $n \in M_n$
3: $\quad \lambda_{n,m}^{(0)} = l_n$
4: *– iteraciones –*
5: para $i \in \{1, \ldots, I_{max}\}$
6: $\quad$ *– actualización de los nodos de comprobación –*
7: $\quad$ para $m \in \{1, \ldots, M\}$ y $n \in N_m$
8: $\quad\quad \mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\backslash n}} (|\lambda_{n',m}^{(i-1)}|)$
9: $\quad$ *– actualización de los nodos de bit –*
10: $\quad$ para $n \in \{1, \ldots, N\}$ y $m \in M_n$
11: $\quad\quad \lambda_n^{(i)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(i)}$
12: $\quad\quad \lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \mu_{m,n}^{(i)}$
13: $\quad$ *– condición de parada –*
14: $\quad\quad p = z \cdot \mathbf{H}^{\mathrm{T}}$
15: $\quad\quad$ si $p = 0$ : *break*
16: *– decodificación hard –*
17: para $n \in \{1, \ldots, N\}$
18: $\quad z_n = \begin{cases} 1 & \lambda_n^{(i)} \geq 0 \\ 0 & \lambda_n^{(i)} < 0 \end{cases}$

## III. Arquitectura del simulador

El emulador hardware se compone de cuatro bloques tal y como se muestra en la figura 3. En general el funcionamiento es el mismo que los simuladores software, donde primero se genera una secuencia binaria (paquete de datos), se codifica y se transmite por un canal AWGN. Después, los símbolos recibidos son decodificados y el resultado se compara con los bits transmitidos para evaluar el número de errores. Los parámetros de configuración del emulador son la varianza del ruido $\sigma^2$ y su inversa escalada $2/\sigma^2$, el número máximo de iteraciones $I_{max}$ y la cantidad de paquetes erróneos que se quiere detectar. Estos parámetros se configuran desde un ordenador a través de una interfaz software. Además de configurar el emulador hardware, la interfaz también se encarga de recolectar los datos de la simulación y almacenarlos en ficheros para su posterior análisis.

### A. Generador de símbolos recibidos

La generación de los símbolos recibidos se hace a partir de una palabra código almacenada en una memoria ROM, la cual ha sido generada codificando una trama de datos aleatoria. En el emulador hardware no se ha implementado el codificador debido al alto coste que tiene y además, se han realizado varias pruebas en software para verificar que el uso de una palabra código fija no varía los resultados de las simulaciones. Los bits de la palabra código se modulan en BPSK y se les suma ruido blanco

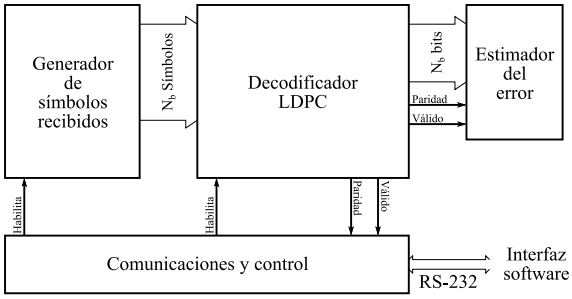


Fig. 3. Diagrama de bloques del emulador hardware.

Gaussiano con el fin de emular el canal AWGN.

El ruido blanco Gaussiano se genera utilizando la inversión de la función de distribución acumulada (ICDF). Primero se generan números aleatorios uniformemente distribuidos a partir de una semilla y luego se aplica la inversa de la función de distribución Gaussiano. La inversa se aproxima mediante interpolación polinómica utilizando partición no uniforme como se explica en [17]. El ruido generado tiene varianza unitaria por lo que es necesario escalarlo por $\sigma^2$ para obtener la SNR requerida. El ruido escalado se multiplica por $2/\sigma^2$ para obtener la razón de verosimilitud (LLR) la cual se envía al siguiente bloque. El generador de números aleatorios uniformemente distribuidos está implementado mediante la combinación de dos generadores Tausworthe [18] con una longitud de 64 bits y posee una periodicidad muy alta, de aproximadamente $2^{175}$.

Para SNRs altos el número de iteraciones del decodificador LDPC es bajo, siendo uno en muchos casos. Como se muestra en la sección III-B el número de ciclos que necesita el decodificador depende de $z$ y del número de iteraciones: para una iteración son necesarios $z$ ciclos. Por lo tanto, para que la generación no sea el cuello de botella con SNR alto es necesario paralelizar. Para poder generar datos a la máxima tasa posible de decodificación (un sola iteración) son necesarios $N_b$ generadores, los cuales generan $z$ símbolos cada uno para así completar los $N$ símbolos correspondientes a un paquete. Al tener varios generadores en paralelo se debe garantizar que las secuencias sean independientes. Para esto se han generado varias semillas de forma aleatoria y se ha comprobado que los números generados con cada semilla no coincidan con el resto de las semillas para una cantidad de datos muy alta, superior a $10^{14}$. De esta manera podemos garantizar que se generan por lo menos $10^{14}$ datos independientes.

### *B. Decodificador LDPC*

El decodificador LDPC es el componente más complejo del emulador hardware. Éste debe conseguir una alta tasa de decodificación con una utilización de recursos razonable. En la literatura existen varias arquitecturas para la implementación hardware de decodificadores para códigos LDPC estructurados. Entre las arquitecturas existentes se ha seleccionado la arquitectura parcialmente paralela basada en memorias [19] debido a la buena relación entre área y tasa de decodificación que posee. Esta arquitectura requiere $N_b$ unidades de nodo variable (VNU) y $M_b$ unidades de comprobación (CNU), las cuales implementan las operaciones de los nodos $V_n$ y $C_m$, respectivamente. Cada una de las VNUs está asociada a una columna de submatrices y cada una de las CNUs a una fila, por lo tanto, la primera VNU está asociada a las $z$ primeras columnas de la matriz $\mathbf{H}$ y la primera CNU a las $z$ primeras filas. Las CNUs y VNUs están conectadas a través de memorias como se muestra en la figura 4 para la matriz de paridad de la figura 2 con $M_b = 3$ y $N_b = 5$. En cada memoria se almacenan los mensajes $\lambda_{n,m}$ y $\mu_{m,n}$ correspondientes a las VNUs y CNUs, respectivamente. Las fiabilidades intrínsecas del canal $l_n$, generadas en el bloque generador de símbolos, también se almacenan en memorias. Estas fiabilidades se leen en cada iteración tal y como se ve en la línea 11 del pseudocódigo 1.

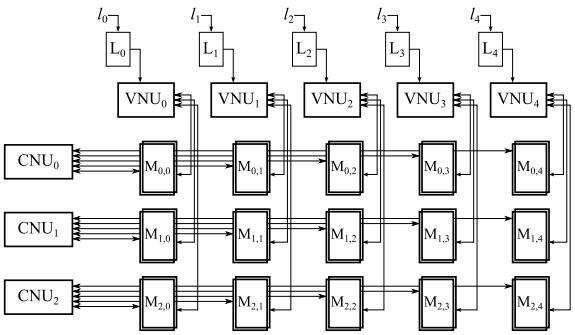


Fig. 4. Arquitectura parcialmente paralela basada en memorias ($M_b = 3$ y $N_b = 5$).

La actualización de las unidades se hace de forma secuencial: cada VNU procesa $z$ columnas de la matriz $\mathbf{H}$ y posteriormente cada CNU procesa $z$ filas. Todas las VNUs trabajan en paralelo, al igual que las CNUs. Por lo tanto, una iteración requiere $2 \cdot z + \eta$ ciclos para ser completada, donde $\eta$ depende de la latencia de la unidades. Para conseguir una máxima tasa, tanto las CNUs como las VNUs, se implementan de forma paralela y completamente segmentadas. Las CNUs implementan las operaciones de las líneas 7 y 8 del pseudocódigo 1, mientras que las VNUs implementan las operaciones de las líneas 10, 11 y 12. El control del decodificador se ha automatizado para adaptarse a diferentes niveles de segmentación debido a que éste varía dependiendo del algoritmo de decodificación.

La figura 5 muestra la arquitectura de la CNU y la VNU para la matriz de paridad de la figura 2. En los nodos de comprobación $C_m$ se realiza la operación $\min(x)$ para cada $n \in N_m$ excluyendo el elemento $n$. Esta operación se reduce a realizar el cálculo del primer y segundo mínimo, $min1$ y $min2$, del conjunto $N_m$ y luego seleccionar el valor apropiado para cada $n$. Por lo tanto, la CNU se compone básicamente del estimador de los dos mínimos y unos multiplexores (MUX) que seleccionan el valor mínimo correspondiente. La función $\Gamma^{(i)}_{m,n}$ se implementa haciendo una XOR con los signos de los mensajes de entrada. Los mensajes de entrada y salida están en formato signo-magnitud (SM), por lo tanto no es necesario realizar la operación valor absoluto, denotada como $|x|$. En la VNU se convierten los mensajes de entrada a formato de complemento a 2 (2sC) y luego se realizan las operaciones de suma y resta descritas en las líneas

11 y 12 del pseudocódigo 1. Los mensajes de salida son convertidos al formato SM nuevamente por los bloques "2sC-SM".

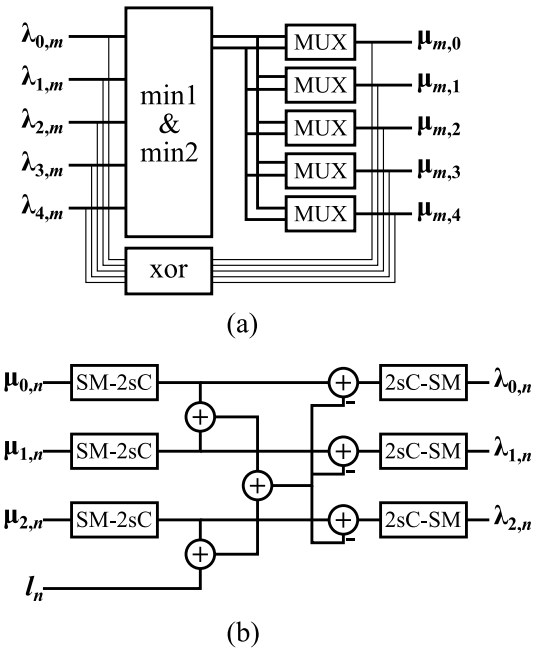


Fig. 5. Arquitectura de las unidades de computo: (a) CNU, (b) VNU.

El direccionamiento de las memorias se hace de acuerdo a la matriz de paridad. Para el ejemplo de la figura 4, cuya matriz de paridad se muestra en la figura 2, en el primer ciclo la $VNU_0$ lee y escribe las posiciones 1, 4 y 2 de las memoria $M_{0,0}$, $M_{1,0}$ y $M_{2,0}$, respectivamente. En el segundo las posiciones 2, 1 y 3, en el tercero la posiciones 3, 2 y 4 y el cuarto las posiciones 4, 3 y 1. Las CNUs leen y escriben en orden, esto es 1, 2, 3 y 4 para los ciclos 1, 2, 3 y 4, respectivamente. Debido a que las unidades procesan los mensajes de forma secuencial, primero VNUs y luego CNUs, es posible optimizar el uso de la memoria almacenando los dos tipos de mensajes ($\lambda_{n,m}$ y $\mu_{m,n}$) en la misma memoria. Para esto se ha dividido la memoria en dos secciones, en la parte baja se almacenan los mensajes que generan las CNUs y en la parte alta los datos que generan las VNUs. El control se encarga de gestionar el direccionamiento de las memorias, así como las señales de lectura y escritura.

Las salidas del decodificador son el paquete de datos decodificado, la paridad y la señal que indica que los datos son válidos. Hay una salida válida en cada iteración y un control externo es el que se encarga de realizar la finalización anticipada siempre y cuando la señal de paridad indique que el paquete decodificado es una palabra código válida. La estructura del decodificador permite implementar nuevos algoritmos del tipo de paso de mensajes de forma rápida debido a que sólo es necesario codificar las unidades de computo, CNU y VNU. Tanto la matriz de paridad como los anchos de palabra se configuran mediante un paquete VHDL el cual es generado automáticamente por una interfaz software desarrollada en Matlab.

### C. Estimador del error

En este bloque se calcula el número de bits erróneos por iteración comparando la salida del decodificador con la palabra código transmitida, la cual se encuentra almacenada en una memoria ROM. El paquete se considera erróneo si el número de bits erróneos es diferente de cero. Los resultados se almacenan por iteración hasta alcanzar el número máximo de paquetes erróneos en la máxima iteración. Por tanto, las memorias RAM que almacenan los resultados (bits erróneos y paquetes erróneos) tienen una profundidad igual al máximo número de iteraciones. Para poder estimar las tasas de error de paquete (PER) y de error de bit (BER) es necesario almacenar también el número de paquetes transmitidos. El tener los resultados almacenados por iteración permite analizar los resultados para diferentes valores de iteración, siempre y cuando sean menores al $I_{max}$ con el que se ha hecho la simulación.

### D. Comunicaciones y control

Este bloque se encarga del control interno del emulador hardware y de la comunicación con la interfaz software. Desde la interfaz software se envían los parámetros de configuración del emulador que se almacenan en registros. Además, la interfaz software es la encargada de indicar al emulador cuando comienza la simulación a través de una palabra de control. Cuando la interfaz software detecta que la simulación a concluido envía una palabra de control que indica al emulador que debe transmitir los datos almacenados: número de paquetes erróneos por iteración, número de bits erróneos por iteración, número de paquetes generados y número total de ciclos de reloj que se han necesitado para la simulación. Estos datos también pueden ser leídos mientras el emulador se encuentra en funcionamiento, así mismo, desde la interfaz software se pueden leer los registros de configuración para poder comprobar que lo que se ha enviado ha sido recibido correctamente. La comunicación entre la interfaz software y el emulador hardware se realiza por medio del puerto serie usando el estándar RS232.

El diagrama de estados simplificado del bloque de control se muestra en la figura 6. La simulación comienza cuando se recibe la palabra de inicio desde la interfaz software. En el estado "carga" se genera la señal que habilita el bloque generador, esta señal tiene una duración de $z$ ciclos de reloj. Una vez se han generado los datos se pasa al estado "simula" en el que se habilita el decodificador hasta que éste alcanza el número máximo de iteraciones o se encuentra una palabra código válida (se cumple la paridad). Cuando finaliza la decodificación se vuelve al estado "carga" para generar el siguiente paquete de datos hasta encontrar el número máximo de paquetes erróneos en la máxima iteración. El bloque estimador de error funciona con las señales de control generadas por el bloque decodificador, por lo tanto, los datos de bits y paquetes erróneos se van almacenando mientras la maquina de estados se encuentra en el estado "simula". Cuando se han encontrado el número máximo de paquetes erróneos en la máxima iteración se vuelve al estado "espera" y se indica que la simulación ha terminado.

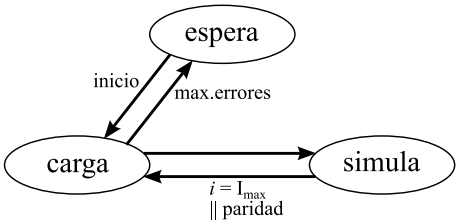


Fig. 6. Diagrama de estados de la maquina de control interna del emulador hardware.

## IV. Resultados

El emulador hardware se ha implementado para el código LDPC del estándar IEEE 802.3an con varios algoritmos de decodificación. Este código es regular estructurado con $N = 2048$, $M = 384$ y $z = 64$. La implementación se ha hecho utilizando la placa de desarrollo Terasic DE2-115 [20] la cual dispone de un dispositivo Altera Cyclone IV [21] que cuenta con 114480 elementos lógicos, 432 memorias embebidas de 9K y 532 multiplicadores embebidos de 9 bits. Los algoritmos que se han evaluado son el MS, αMS y βMS. Los factores de corrección para los algoritmos αMS y βMS son 0.75 y 1, respectivamente. El número máximo de iteraciones $I_{max}$ es 50 y el número de paquetes erróneos para $I_{max}$ es de 100, lo que garantiza un resultado estadísticamente fiable. La nomenclatura para definir el esquema de cuantificación es $[q : f]$, donde $q$ corresponde al número total de bits, incluido el signo, y $f$ al número de bits fraccionales.

La tabla I muestra los resultados de implementación del emulador hardware con los diferentes algoritmos de decodificación, número de elementos lógicos (LEs), número de memorias embebidas (M9K) y número de multiplicadores embebidos de 9 bits (P9). Además, se muestra la tasa de error de bit (BER), el tiempo de simulación (T) y el *throughput* medio (Th.) para un SNR de 5 dB. La máxima frecuencia de trabajo del emulador es ligeramente superior a los 150 MHz para todos los algoritmos implementados, por lo que se ha utilizado un reloj de exactamente 150 MHz generado internamente a partir del reloj de la placa DE2-115 que es de 50 MHz. Para conseguir máxima tasa de decodificación se ha implementado la finalización anticipada, con lo que se consigue reducir el número medio de iteraciones cuando el SNR es alto. El esquema de cuantificación utilizado para los mensajes $l_n$ y $\lambda_{n,m}$ en todos los algoritmos es $[6 : 2]$. El esquema de los mensajes $\mu_{m,n}$ en los algoritmos MS y βMS es también $[6 : 2]$ y en el algoritmo αMS es $[8 : 4]$.

TABLA I
Resultados de implementación con esquema de cuantificación $[6 : 2]$.

| | **LE's** | **M9K** | **P9** | **BER** | **T** [horas] | **Th.** [Mbps] |
|---|---|---|---|---|---|---|
| MS | 50753 | 388 | 256 | $2.7e^{-10}$ | 5.29 | 654.2 |
| αMS | 53330 | 388 | 256 | $1.0e^{-11}$ | 32.89 | 664.1 |
| βMS | 50820 | 388 | 256 | $8.0e^{-12}$ | 41.89 | 658.5 |

Analizando los resultados de la tabla se observa que el *throughput* es dependiente del algoritmo. Además, se puede ver que el incremento del tiempo de simulación no es lineal respecto a la tasa de error (BER). También destaca el porcentaje de ocupación de la FPGA que es de aproximadamente el 45% de elementos lógicos, del 90% de memorias M9K y del 48% de multiplicadores embebidos de 9 bits.

Las curvas de error correspondientes a los algoritmos implementados se muestran en la figura 7. En éstas se observa que el cambio de pendiente de los algoritmos αMS y βMS comienza en 4.4 dB y un PER de $10^{-7}$ y converge a $10^{-9}$. El cambio de pendiente del algoritmo MS comienza en $10^{-8}$ de PER, una potencia de 10 más abajo que los otros dos algoritmos, pero converge al mismo sitio. Cabe destacar que para encontrar el suelo de error de estos algoritmos con simulación software han sido necesarios varios meses de simulación mientras que con el emulador hardware se ha conseguido en menos de 5 días.

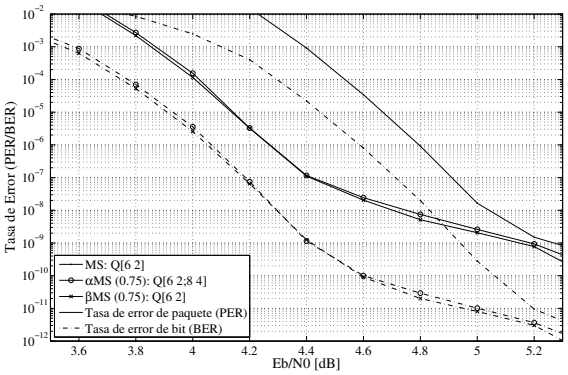


Fig. 7. Curvas de error de los algoritmos MS, αMS y βMS.

En [22] se propone un emulador hardware que consigue una tasa de decodificación de 4.7 Gbps, sin embargo requiere una gran cantidad de dispositivos FPGA que trabajan en paralelo y por lo tanto, un alto coste. Otro trabajo similar se muestra en [23] donde se optimizan los mensajes y su almacenamiento. En este último se consigue una tasa de decodificación similar a la de este trabajo pero únicamente para códigos Quasi-Cíclicos (QC). Por otro lado, la arquitectura propuesta permite la paralelización en varios dispositivos del emulador, similar a lo que se propone en [22], aumentando por lo tanto la tasa y disminuyendo el tiempo de simulación.

## V. Conclusiones

En este trabajo se ha desarrollado un entorno de simulación hardware que permite reducir el tiempo de simulación de códigos LDPC respecto a un entorno completamente software. Muchas aplicaciones requieren conocer el comportamiento de los códigos cuando la tasa de error de bit es muy baja, menor a $10^{-12}$. Con las herramientas software tradicionales las simulaciones pueden durar varios meses, sin embargo, con el entorno de simulación hardware presentado en este trabajo el tiempo se reduce a 5 días. El simulador ha sido probado con el código LDPC del estándar IEEE 802.3an sobre un dispositivo Altera Cyclone IV montado en una placa de desarrollo de bajo coste, llegando a conseguir una tasa de

datos superior a los 600 Mbps con un reloj de 150 MHz. El decodificador LDPC se ha implementado con una arquitectura parcialmente paralela basada en memorias que consigue una buena relación entre complejidad hardware y tasa de decodificación. En el decodificador se ha utilizado la finalización anticipada para reducir el número de iteraciones medias cuando la relación señal a ruido es alta. Además, la generación de ruido Gaussiano se ha paralelizado para evitar que sea el cuello de botella cuando el decodificador trabaja a máxima tasa de datos.

Junto con el emulador hardware se han programado funciones en Matlab para automatizar el proceso de implementación para cualquier código LDPC estructurado y cualquier esquema de cuantificación. Además, la inclusión de un nuevo algoritmo del tipo de paso de mensajes es fácil y rápida ya que únicamente es necesario codificar las unidades de cómputo, CNU y VNU.

## AGRADECIMIENTOS

Este trabajo lo ha financiado el Ministerio Economía y Competitividad mediante el proyecto con referencia TEC2012-27916.

## REFERENCIAS

[1] "Digital video broadcasting (dvb); second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (dvb-s2)," *ETSI EN 302 307 V1.2.1*, aug. 2009.

[2] "G.9960 : Unified high-speed wire-line based home networking transceivers - system architecture and physical layer specification," *ITU-T Recommendations: G Series*, 2011.

[3] "IEEE standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 5: Enhancements for higher throughput," *IEEE Std 802.11n-2009*, oct. 2009.

[4] "IEEE standard for local and metropolitan area networks part 16: Air interface for broadband wireless access systems," *IEEE Std 802.16-2009*, may. 2009.

[5] "IEEE draft standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications (revision of ieee std 802.3-2005 including all approved amendments)," *IEEE Unapproved Draft Std P802.3/D2.2, April 2008*, 2008.

[6] R. Tanner, "A recursive approach to low complexity codes," *Information Theory, IEEE Transactions on*, vol. 27, no. 5, pp. 533 – 547, sep 1981.

[7] D.J.C. MacKay, "Good error-correcting codes based on very sparse matrices," *Information Theory, IEEE Transactions on*, vol. 45, no. 2, pp. 399 –431, mar 1999.

[8] J. Lu and J.M.F. Moura, "Structured ldpc codes for high-density recording: large girth and low error floor," *Magnetics, IEEE Transactions on*, vol. 42, no. 2, pp. 208 – 213, feb. 2006.

[9] M.P.C. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *Information Theory, IEEE Transactions on*, vol. 50, no. 8, pp. 1788 – 1793, aug. 2004.

[10] Hui Jin, Aamod Khandekar, Aamod Kh, and Robert J. McEliece, "Irregular repeat accumulate codes," *Proc. of the Second International Symposium on Turbo Codes and Related Topics*, sep 2000.

[11] I. Djurdjevic, Jun Xu, K. Abdel-Ghaffar, and Shu Lin, "A class of low-density parity-check codes constructed based on reed-solomon codes with two information symbols," *Communications Letters, IEEE*, vol. 7, no. 7, pp. 317 – 319, july 2003.

[12] T.J. Richardson and R.L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 599 –618, feb 2001.

[13] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498 –519, feb 2001.

[14] Y. Kou, S. Lin, and M.P.C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *Information Theory, IEEE Transactions on*, vol. 47, no. 7, pp. 2711 –2736, nov. 2001.

[15] M.P.C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *Communications, IEEE Transactions on*, vol. 47, no. 5, pp. 673 –680, may 1999.

[16] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *Communications, IEEE Transactions on*, vol. 53, no. 8, pp. 1288 – 1299, aug. 2005.

[17] Roberto Gutierrez, Vicente Torres, and Javier Valls, "Hardware architecture of a gaussian noise generator based on inversion method," *on Circuits Systems II, IEEE Transactions*, To be published 2012.

[18] Pierre L'Ecuyer, "Maximally Equidistributed Combined Tausworthe Generators," *Mathematics of Computation*, vol. 65, no. 213, pp. 203–213, 1996.

[19] Zhongfeng Wang and Zhiqiang Cui, "Low-complexity high-speed decoder design for quasi-cyclic ldpc codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 1, pp. 104 –114, jan. 2007.

[20] Terasic, "http://www.terasic.com.tw/en," .

[21] Altera, "http://www.altera.com/devices/fpga/cyclone-iv," .

[22] Yu Cai, Seungjune Jeon, Ken Mai, and B.V.K.V. Kumar, "Highly parallel fpga emulation for ldpc error floor characterization in perpendicular magnetic recording channel," *Magnetics, IEEE Transactions on*, vol. 45, no. 10, pp. 3761 –3764, oct. 2009.

[23] Xiaoheng Chen, Jingyu Kang, Shu Lin, and V. Akella, "Accelerating fpga-based emulation of quasi-cyclic ldpc codes with vector processing," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 1530 –1535.