UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO DE DECODIFICADORES DE ALTAS PRESTACIONES PARA CÓDIGOS LDPC

TESIS DOCTORAL

AUTOR:
Fabián Angarita Preciado

DIRECTORES:
Javier Valls Coquillat
Vicenç Almenar Terré

Valencia, 19 de Julio de 2013.

RESUMEN

En esta tesis se han investigado los algoritmos de decodificación para códigos de comprobación de paridad de baja densidad (LDPC) y las arquitecturas para la implementación hardware de éstos. El trabajo realizado se centra en los algoritmos del tipo de intercambio de mensajes para códigos estructurados los cuales se incluyen en varios estándares de comunicaciones.

Inicialmente se han evaluado las prestaciones de los algoritmos existentes Sum-product, Min-Sum y las principales variantes de este último (Min-Sum con escalado y Min-Sum con *offset*). Además, se ha realizado un análisis de precisión finita utilizando los códigos LDPC de los estándares IEEE 802.3an, IEEE 802.11n e IEEE 802.16e. Posteriormente se han propuesto dos algoritmos basados en el algoritmo Min-Sum, denominados Min-Sum entero y Min-Sum modificado con corrección. La complejidad de éstos es menor que las de los algoritmos estudiados anteriormente y además permiten una implementación *hardware* eficiente. Por otra parte, se han estudiado diferentes métodos de actualización de los algoritmos de decodificación: por inundación, por capas horizontales (*layered*) y por capas verticales (*shuffled*), y se ha propuesto un nuevo método por capas verticales entrelazadas (*x-shuffled*) que consigue mejorar la tasa de decodificación.

Tras el estudio algorítmico, se han realizado implementaciones *hardware* con diferentes arquitecturas para los algoritmos y métodos de actualización evaluados y propuestos. En la mayoría de algoritmos implementados se requiere el cálculo de los dos primeros mínimos, por lo que inicialmente se realiza un estudio de las arquitecturas *hardware* para realizar este cálculo y se ha propuesto una nueva arquitectura de menor complejidad. En segundo lugar se ha realizado una comparación de las prestaciones *hardware* de los diferentes algoritmos con las arquitecturas de referencia: completamente paralela y parcialmente paralela basada en memorias. También se han propuesto dos arquitecturas enfocadas a la alta velocidad, la cuales se implementan con el algoritmo Sum-Product. La primera es una modificación de la arquitectura *Sliced Message-Passing* que consigue una reducción en el área de la implementación, y la segunda, es una arquitectura específica para el método de actualización propuesto *x-shuffled* que alcanza tasas de decodificación muy altas. Finalmente, se han implementado los algoritmos propuestos con la arquitectura *layered* obteniendo implementaciones *hardware* eficientes con baja área y muy alta tasa de decodificación. Estas últimas consiguen un ratio entre tasa de decodificación y área mejor que las implementa-

ciones existentes en la literatura.

Por último, se ha evaluado el comportamiento de los algoritmos de decodificación estudiados en la zona de baja tasa de error, donde las prestaciones se suelen degradar debido a la aparición de un suelo de error. Para ello se ha implementado un simulador *hardware* usando dispositivos FPGA. La tasa de datos alcanzada con el simulador *hardware* diseñado es superior a la de otros simuladores documentados en la literatura. En la zona de baja tasa de error el algoritmo propuesto Min-Sum modificado con corrección presenta un mejor comportamiento que el resto de algoritmos evaluados, consiguiendo bajar el suelo de error varios órdenes de magnitud.

ABSTRACT

In this thesis we research on decoding algorithms for low density parity check codes (LDPC) and hardware architectures for their implementation. This work focuses on the message passing algorithms for structured codes which are included in various communication standards.

We initially evaluate the performance of the existing algorithms Sum-Product, Min-Sum and its major variants (scaled Min-Sum and offset Min-Sum). In addition, a finite precision analysis is performed using different LDPC codes from the IEEE 802.3an, IEEE 802.11n and IEEE 802.16e standards. Subsequently, two algorithms are proposed based on the MS algorithm, called integer Min-Sum and modified Min-Sum with correction. These algorithms have less complexity than the previously studied algorithms and also, they yield efficient hardware implementations. Moreover, we study different methods of updating the nodes information in the decoding algorithms: flooding, horizontal layered and vertical layered (shuffled), and we propose a new interlaced vertical layered method (x-shuffled) obtaining a higher throughput.

After the algorithm study, we implement in hardware the evaluated and proposed algorithms with different architectures. Due to the fact that most of the implemented algorithms require the calculation of the first and second minimum values, we first study the existing hardware architectures to find two minimum, and we also propose a new architecture which reduces the area of the circuit. Secondly, the comparison of the implementation results among the different algorithms is performed with two reference architectures: fully parallel and memory-based partially-parallel. Next, we propose two architectures for high-throughput decoding, which are implemented with the Sum-Product algorithm. The first one is a modification of the Sliced-Message Passing architecture that yields a reduction of the implementation area, and the second one, is a specific architecture for the proposed update method x-shuffled which also achieves high throughput. Finally, the proposed algorithms are implemented with the horizontal layered architecture obtaining low-area and high-throughput hardware implementations. Morever, these implementations achieve better ratio between area and throughput than the existing implementations in the literature.

Lastly, we evaluate the behaviour of the studied decoding algorithms at low-error rate zone, where the performance is usually degraded due to the error floor phenomenon. Therefore, we implement a hardware-based simulator in an FPGA device. The data throughput

achieved by this simulator is higher than other documented simulators in the literature. At low-error rates, one of the proposed algorithms, the modified Min-Sum with correction, shows better performance than the other ones, exhibing an error floor several orders of magnitude lower.

RESUM

En aquesta tesi s'han investigat els algoritmes de descodificació per a codis de comprovació de paritat de baixa densitat (LDPC) y les arquitectures hardware per a la implementació d'aquests. El treball realitzat es centra en els algoritmes del tipus intercanvi de missatges per a codis estructurats, que són inclosos en diversos estàndards de comunicacions.

Inicialment s'han avaluat les prestacions dels algoritmes existents Sum-product, Min-Sum i les principals variants d'aquest últim (Min-Sum amb escalat i Min-Sum amb *offset*). A més, s'ha realitzat una anàlisi de precisió finita utilitzant els codis del estàndards IEEE 802.3an, IEEE 802.11n i IEEE 802.16e. Posteriorment, s'han proposat dos algoritmes basats en l'algoritme Min-Sum, denominats Min-Sum sencer i Min-Sum modificat amb correcció. La complexitat d'aquests és menor que la dels algoritmes estudiats anteriorment i, a més, permeten una implementació hardware eficient. Per altra banda, s'han estudiat diferents mètodes d'actualització dels algoritmes de descodificació: per inundació, per capes horitzontals (*layered*) i per capes verticals (*shuffled*), i s'ha proposat un nou mètode per capes verticals entrellaçades (*x-shuffled*) que aconsegueix millorar la velocitat de descodificació.

Després de l'estudi algorítmic, s'han realitzat implementacions hardware amb diferents arquitectures per als algoritmes i mètodes d'actualització avaluats i proposats. La majoria d'algoritmes implementats requereixen el càlcul dels dos primers mínims, pel que inicialment s'han estudiat les arquitectures hardware per a realitzar aquest càlcul i s'ha proposat una nova arquitectura de menor complexitat. En segon lloc, s'han comparat les prestacions hardware dels diferents algoritmes amb les arquitectures de referència: completament paral·lela i parcialment paral·lela basada en memòries. També s'han proposat dues arquitectures enfocades a l'alta velocitat, que s'implementen amb l'algoritme Sum-Product. La primera és una modificació de l'arquitectura *Sliced Message-Passing* que aconsegueix una reducció de l'àrea de la implementació, i la segona és una arquitectura específica per al mètode d'actualització proposat *x-shuffled* que aconsegueix velocitats de descodificació molt elevades. Finalment, els algoritmes proposats s'han implementat amb l'arquitectura *layered* obtenint implementacions hardware eficients amb baixa àrea i molt alta velocitat de descodificació. Aquestes últimes aconsegueixen una ràtio entre velocitat de descodificació i àrea millor que les implementacions existents a la literatura.

Per últim, s'ha avaluat el comportament dels algoritmes de descodificació estudiats en

la zona de baixa taxa d'error, on les prestacions es solen degradar degut a l'aparició d'un sòl d'error. Per a fer aquesta avaluació s'ha implementat un simulador *hardware* utilitzant dispositius FPGA. La taxa de dades aconseguida amb el simulador *hardware* dissenyat és superior a la d'altres simuladors documentats a la literatura. Un dels algoritmes proposats, el Min-Sum modificat amb correcció, presenta un millor comportament a la zona de baixa taxa d'error que la resta d'algoritmes avaluats, aconseguint baixar el sòl de l'error diversos ordres de magnitud.

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis directores de tesis Dr. Javier Valls y Dr. Vicenç Almenar por el apoyo que me han brindado durante el desarrollo de este trabajo. También agradezco a todo el grupo de investigación GISED, y en especial a M^{a} José Canet, por su inestimable ayuda.

Por otro lado, quiero mostrar mi agradecimiento al Ministerio de Educación y Ciencia que a través de los proyectos TEC20080-6787 y TEC2011-27916 han financiado esta tesis doctoral.

Por último agradecer a Carolina y Júlia por su paciencia y comprensión y a mi hermano y madre por la confianza que han depositado en mí.

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ALGORITMOS

LISTA DE ACRÓNIMOS

ASIC	*Application-Specific Integrated Circuit*
AWGN	*Additive White Gaussian Noise*
BEE	*Berkeley Emulation Engine*
BER	*Bit Error Rate*
BPSK	*Binary Phase Shift Keying*
CMOS	*Complementary Metal-Oxide-Semiconductor*
CNP	*Check-Node Processor*
DVB-S2	*Digital Video Broadcasting Satellite - Second Generation*
FEC	*Forward Error Correction*
FPGA	*Field Programmable Gate Array*
HUE	*Hardware Utilization Efficiency*
ICDF	*Inverse Cumulative Distribution Function*
IEEE	*Institute of Electrical and Electronics Engineers*
IRA	*Irregular Repeat-Accumulate*
LDPC	*Low Density Parity Check*
LE	*Logic Element(s)*
LLR	*Log-Likelihood Ratio*
LUT	*Look-Up Table*

MS	Min-Sum
QC	*Quasi-Cyclic*
PER	*Packet Error Rate*
RAM	*Random-Access Memory*
ROM	*Read-Only Memory*
RS-232	*Recommended Standard 232*
RS	*Reed Solomon*
SNR	*Signal-to-Noise Ratio*
SP	*Sum-Product*
SRL	*Shift Register Lookup Table*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VNP	*Variable-Node Processor*
WLAN	*Wireless Local Area Network*
WPAN	*Wireless Personal Area Network*

Capítulo 1

INTRODUCCIÓN

Los sistemas de comunicación actuales tienen que hacer frente a la creciente demanda de servicios que requieren tasas de transmisión de datos elevadas. Por ejemplo, en los sistemas de comunicaciones inalámbricas se contemplan tasas de transmisión de varios cientos de Mbps e incluso los Gbps, y en los estándares recientes de redes de área local para computadores se contemplan tasas de transmisión de 10 Gbps sobre par trenzado y de 100 Gbps sobre fibra óptica. Para conseguir estas elevadas tasas de transmisión los sistemas de comunicaciones tienen que ser capaces de ejecutar en tiempo real algoritmos de procesado de la señal muy complejos a tasas de muestreo elevadas, y en muchos casos minimizando el consumo de potencia, para que puedan ser incluidos en terminales móviles.

Los esquemas de detección y corrección de errores FEC (*Forward Error Correction*) juegan un papel importante en los sistemas de comunicaciones actuales. Éstos permiten incrementar las tasas de transmisión de datos y además dotan al sistema de una mayor fiabilidad y robustez, haciendo que los requisitos de potencia transmitida sean menores y reduciendo el número de retransmisiones. Los Turbo códigos y los códigos de comprobación de paridad de baja densidad (*Low Density Parity Check* - LDPC) son los esquemas FEC existentes que mejores prestaciones consiguen. Estos últimos están emergiendo como alternativa a los turbo códigos en la mayoría de escenarios debido a que requieren elementos de procesamiento más simples, dado que su decodificación no está basada en el enrejado (*trellis*). Además, la tasa de error por trama es inferior, el suelo de error es más bajo y no requieren de un entrelazador (*interleaver*) complejo para conseguir buenas prestaciones. Por otro lado, su estructura inherentemente paralela los haces atractivos para aplicaciones de alta velocidad. Por ello, estos códigos se incluyen en varios estándares de comunicaciones como el estándar europeo de transmisión por satélite de televisión digital (DVB-S2), las redes de cableado residencial (G.hn/G.9660), las redes inalámbricas de área local Wi-Fi (IEEE 802.11n), las redes inalámbricas de acceso metropolitano de banda ancha WiMAX (IEEE 802.16e), las redes inalámbrica de área personal mmWave WPAN (IEEE 802.15.3c) y 10-Gigabit Ethernet sobre par trenzado no blindado (IEEE 802.3an). También están siendo objeto de investigación para su futuro uso en sistemas avanzados de almacenamiento magnético y magnetico-óptico, y en sistemas de comunicaciones ópticas de largo recorrido.

1.1. Objetivos

El principal objetivo de esta tesis es el diseño y la implementación *hardware* de arquitecturas eficientes para la decodificación de códigos de comprobación de paridad de baja densidad (LDPC) utilizados en aplicaciones que requieren altas tasas de transmisión. La finalidad es conseguir implementaciones que alcancen altas tasas de decodificación (del orden de los Gbps) con la mínima área posible y con unas prestaciones, en términos de capacidad correctora, buenas. Esta tesis se divide en dos partes: una primera en donde se realiza el estudio algorítmico y la evaluación de precisión finita y una segunda dedicada a las arquitecturas y su implementación *hardware*.

Los objetivos específicos del estudio algorítmico y la evaluación de precisión finita se detallan a continuación:

- Evaluar los diferentes códigos, algoritmos de decodificación y métodos de actualización que aparecen en la literatura.
- Proponer algoritmos de decodificación de baja complejidad computacional (esto se debe traducir en una reducción de la complejidad *hardware*) y que presenten bajas pérdidas de prestaciones respecto a los existentes.
- Realizar un análisis de precisión finita, tanto a los algoritmos existentes como a los propuestos, para determinar los anchos de palabra que mejor compromiso entre número de bits y pérdida de prestaciones consiguen.

Los objetivos específicos de la parte dedicada a las arquitecturas y su implementación *hardware* son:

- Evaluar el impacto en las implementaciones *hardware* de los algoritmos propuestos, comparado con los existentes, utilizando arquitecturas conocidas.
- Proponer mejoras para las arquitecturas existentes con el fin de hacerlas más eficientes en términos de área y tasa de decodificación.
- Proponer arquitecturas para alta velocidad eficientes (área y tasa de decodificación) utilizando los algoritmos propuestos o los existentes.

1.2. Metodología

En esta sección se expone la metodología seguida para abordar los objetivos de esta tesis doctoral. En primer lugar se ha realizado una búsqueda bibliográfica para identificar el estado del arte y los algoritmos más relevante que son usados como referencia. Posteriormente se han evaluado las prestaciones y se ha realizado un análisis de precisión finita de estos algoritmos. Para esto, se ha modelado un sistema de comunicaciones en Matlab, con el que se realizan simulaciones para evaluar las prestaciones siguiendo el método de Montecarlo. Con el fin de acelerar las simulaciones los algoritmos de decodificación se han implementado en lenguaje C, utilizando las cabeceras C-Mex para poder integrarlos en el entorno de simulación desarrollado en Matlab. El análisis de precisión finita se realiza también mediante simulación, por lo que ha sido necesaria la implementación de los algoritmos

con precisión finita en lenguaje C. Para cada algoritmo se evalúan diferentes esquemas de cuantificación y se determina el que mejor compromiso entre número de bits y prestaciones consigue.

Para el análisis de los algoritmos propuestos se sigue la misma metodología que para el análisis de los algoritmos de referencia: en primer lugar se han implementado los modelos de punto flotante y precisión finita en lenguaje C y a continuación se han simulado utilizando el entorno desarrollado en Matlab. El principal objetivo de los algoritmos propuestos es conseguir una reducción de la complejidad computacional, y por tanto un reducción en la complejidad *hardware* de las implementaciones, manteniendo unas prestaciones similares a las de los algoritmos de referencia.

Para la implementación *hardware* de los decodificadores se utiliza el lenguaje de descripción *hardware* VHDL (*Very High Speed Integrated Circuit Hardware Description Language*). Los modelos *hardware* primero se verifican funcionalmente a nivel de código. Esta verificación se realiza con la herramienta Modelsim de Mentor Graphics, siguiendo el siguiente flujo de trabajo:

1. Se realiza una simulación con el modelo de precisión finita y se almacenan tanto los datos de entrada como los datos de salida del decodificador (algoritmo de decodificación).
2. En un banco de pruebas, codificado en VHDL, se leen los datos de entrada, se simula el modelo *hardware* y se comparan los datos generados por éste con los obtenidos con el modelo *software* de precisión finita.
3. Si las comparaciones son válidas (los datos son exactamente iguales) se da como verificado el modelo *hardware*.

Una vez validado el modelo *hardware*, éste es implementado en un ASIC o en un dispositivo FPGA. Las implementaciones en un ASIC se realizan usando la librería de celdas estándar Faraday de 90nm con 8 capas metálicas [1]. Para la síntesis se ha empleado la herramienta de Cadence RTL *compiler* y para el emplazado y rutado la herramienta SOC *encounter* de Cadence, la cual integra la herramienta para el análisis de tiempos. Las implementaciones en FPGA se han realizado con las herramientas Quartus II de Altera e ISE de Xilinx, las cuales integran sus respectivas herramientas de análisis de tiempo. Los modelos post emplazado y rutado generados por las diferentes herramientas se verifican funcionalmente siguiendo el mismo flujo que para la verificación a nivel de código.

El flujo de trabajo que se sigue para la implementación en un ASIC se describe a continuación:

1. Se realiza una primera síntesis sin constante de tiempo.
2. Se realiza una segunda síntesis ajustando la constante de tiempo al 80 % del camino crítico obtenido en la síntesis anterior.
3. A continuación se realiza el emplazado y rutado. Este se hace de manera iterativa, ajustando el porcentaje de área requerida hasta obtener una mínima penalización del camino crítico (comparado con el camino crítico de la síntesis). Si el porcentaje de área es inferior al 40 % o la penalización del camino crítico es muy alta (superior al 50 %) se incrementa la constante de tiempo (de la sintesis) y se vuelve al paso 2.

La implementación en un dispositivo FPGA también se realiza de manera iterativa, con el fin de ajustar la frecuencia de trabajo a la máxima posible (mínimo camino critico). Para esto se sigue el siguiente flujo de trabajo.

1. Se realiza una primera compilación con una constante de tiempo alta.
2. Con el resultado de camino crítico, obtenido con la herramienta de análisis de tiempo, se ajusta la constante de tiempo y se realiza de nuevo la compilación.
3. El paso 2 se repite hasta conseguir el mínimo camino crítico sin que se produzcan violaciones de tiempo.

1.3. Contribuciones y publicaciones

Las principales contribuciones de esta tesis se enumeran a continuación:

1. Se han propuesto dos algoritmos, el Min-Sum entero y el Min-Sum escalado con corrección, con el fin de reducir el coste hardware de sus implementaciones y cuyas prestaciones en precisión finita son comparables con las de los algoritmos de referencia (algoritmos conocidos y ampliamente utilizados en la literatura).

2. Se ha propuesto un método de actualización para acelerar la convergencia de los algoritmos con el que se consigue aumentar la tasa de decodificación. Además, con este método se alcanza una eficiencia de utilización *hardware* del 100 %.

3. Se ha propuesto una arquitectura para el cálculo de los dos mínimos, necesario en la mayoría de algoritmos basados en MS, con la que se consigue un coste *hardware* (área) menor que con las arquitecturas existentes.

4. Se ha propuesto una mejora a una arquitectura de la literatura denominada SMP. Con las modificaciones propuestas se ha conseguido una reducción considerable del coste *hardware* manteniendo las prestaciones.

5. Se han propuesto dos arquitecturas para la implementación del método de actualización propuesto *x-shuffled* y del método de actualización existente *shuffled*. Con estas arquitecturas se alcanzan tasas de decodificación muy altas, superiores a los 15 Gbps para el método *x-shuffled* y a los 10 Gbps para el método *shuffled*.

6. Se ha adaptado la arquitectura existente *layered* para la implementación eficiente de los algoritmos propuestos, consiguiendo implementaciones que alcanzan tasas de decodificación muy altas ($>$ 12 Gbps) con un coste *hardware* reducido.

7. Se ha diseñado e implementado un simulador *hardware* que permite el análisis de los códigos LDPC y sus algoritmos de decodificación en tasas de error muy bajas (BER$< 10^{-8}$), donde los simuladores *software* son inviables debido al excesivo tiempo que requieren. El simulador *hardware* propuesto consigue mejores tasas de decodificación que los simuladores de la literatura.

Los resultados de investigación de esta tesis se han publicado en las revistas y se ha presentado en los congresos que se detallan a continuación:

- F. Angarita, J. Marin-Roig, V. Almenar, J. Valls, **"Low-complexity LDPC decoding algorithm for high-speed VLSI implementation"**, *IET Communications*, (a ser publicado) 2012.

- F. Angarita, T. Sansaloni, A. Perez-Pascual, J. Valls, **"Modifed Shuffled Based Architecture for High Throughput Decoding of LDPC Codes"**, *Journal of Signal Processing Systems*, Vol.68, No.2, pp.139-149, Aug.2012.

- F. Angarita, T. Sansaloni, M.J. Canet, J. Valls, **"Improved Sliced Message Passing Architecture for High Throughput Decoding of LDPC Codes"**, *Journal of Signal Processing Systems*, Vol.66, No.2, pp.99-104, Feb.2012.

- F. Angarita, V. Torres, A. Perez-Pascual, J. Valls, **"High-Throughput FPGA-based Emulator for Structured LDPC Codes"**, *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Sevilla, Spain, Dec.2012.

Durante el desarrollo de la tesis también se ha investigado en otras área relacionadas con la implementación eficiente de bloques IP para sistemas de comunicaciones de altas prestaciones. Los resultados de esta investigación se han publicado en las revistas y se han presentado en los congresos que se detallan a continuación:

- F. Angarita, A. Almenar, M.J. Canet, T. Sansaloni, J.Valls, **"Power consumption reduction in a Viterbi Decoder for OFDM-WLAN"**, *Journal of Circuits, Systems and Computers*, Vol.18, No.7, pp.1333-1337, Nov.2009.

- F. Angarita, M.J. Canet, T. Sansaloni, A. Perez-Pascual, J.Valls, **"Efficient mapping of CORDIC Algorithm for OFDM-based WLAN"**, *Journal of Signal Processing Systems*, Vol.52, No.2, pp.181-191, Aug.2008.

- F. Angarita, M.J. Canet, T. Sansaloni, V. Almenar, J. Valls, **"Architectures for the implementation of a OFDM-WLAN Viterbi Decoder"**, *Journal of Signal Processing Systems*, Vol.52, No.1, pp.35-44, Jul.2008.

- F. Angarita, M.J. Canet, T. Sansaloni, V. Almenar and J. Valls, **"Reduction of power consumption in a Viterbi Decoder for OFDM-WLAN"**, *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Marrakech, Morocco, Dec.2007.

- F. Angarita, T. Sansaloni, A. Pérez-Pascual, J. Valls, **"FPGA-based design of a Viterbi decoder for WLAN"**, *IEEE Workshop on Signal Procesing Systems (SiPS)*, Athens, Greece, Nov.2005.

- F. Angarita, T. Sansaloni, A. Pérez-Pascual, J. Valls, **"Efficient FPGA implementation of CORDIC algorithm for circular and linear coordinates"**, *International Conference in Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug.2005.

- E. Todorovich, E. Boemo, F. Angarita, J. Valls, **"Statical power estimation for FPGAs"**, *International Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug.2005.

1.4. Organización de la tesis

Este trabajo está organizado de la siguiente manera:

En el Capítulo 2 se introducen los códigos LDPC y se resumen sus características más importantes. Además, se presentan los códigos LDPC estructurados y se detallan los códigos usados para el desarrollo de esta tesis.

En el Capítulo 3 se describen los algoritmos de decodificación existentes más destacados, así como los métodos de actualización. También se realiza el análisis de prestaciones en punto flotante y el análisis de precisión finita para estos algoritmos y métodos de actualización utilizando los códigos descritos en el capítulo anterior.

En el Capítulo 4 se explican y analizan los algoritmos y métodos de actualización propuestos. Se realiza el análisis de prestaciones y el de precisión finita para cada caso y los resultados se comparan con los obtenidos usando los algoritmos y métodos existentes, obtenidos en el Capítulo 2. En concreto se presentan dos algoritmos y un método de actualización cuyo objetivo es mejorar las prestaciones de las implementaciones *hardware*.

En el Capítulo 5 se trata la implementación *hardware* de los algoritmos expuestos en los capítulos anteriores con diferentes arquitecturas para el código LDPC del estándar IEEE 802.3an, especificado para altas tasas de decodificación. En primer lugar se propone una arquitectura para implementar de manera eficiente el cálculo de los dos mínimos, necesario en la mayoría de algoritmos estudiados. Luego se muestran los resultados de implementación de los diferentes algoritmos con las arquitecturas conocidas, completamente paralela y parcialmente paralela basada en memorias, con el fin de evaluar el impacto que tienen los algoritmos propuestos sobre la implementación *hardware*. En tercer lugar se presenta una modificación de la arquitectura SMP con la que se consigue reducir considerablemente el coste *hardware*. Después se proponen dos arquitecturas para la implementación de los métodos de actualización *shuffled* y *x-shuflled*, las cuales consiguen altas tasas de decodificación. Por último se implementan los algoritmos propuestos con la arquitectura *layered*, consiguiendo implementaciones de bajo coste *hardware* y altas prestaciones.

En el Capítulo 6 se presenta un simulador *hardware* diseñado e implementado para poder estudiar el comportamiento de los códigos y algoritmos a muy bajas tasas de error. Además, se muestran y analizan los resultados de simulación de los algoritmos estudiados y propuestos en la tesis para el código del estándar IEEE 802.3an.

En el Capítulo 7 se exponen las conclusiones de la tesis y se señalan las posibles líneas futuras de trabajo.

Capítulo 2

CÓDIGOS LDPC

Los códigos de comprobación de paridad de baja densidad (LDPC) están dentro de la categoría de códigos de bloque lineales y poseen una gran capacidad de corrección, gracias a la cual es posible diseñar sistemas de comunicaciones que trabajen cerca del límite de Shannon. El principio básico de los códigos LDPC y la decodificación iterativa fue descrito por Gallager en el año 1960 [2], pero su trabajo fue ignorado por más de 35 años debido a las limitaciones en la capacidad de cómputo de la época. Una excepción fue el trabajo de Tanner en 1981 [3], el cual proporcionó una nueva interpretación de estos códigos desde un punto de vista gráfico, llamado grafo de Tanner. A finales de la década de los 90, MacKay y Neal descubrieron nuevamente los códigos LDPC [4], y desde entonces, éstos han sido de gran interés para los investigadores.

En la actualidad los códigos LDPC son utilizados en una gran variedad de aplicaciones, como los sistemas de comunicaciones digitales, la transmisión y difusión de video digital y el almacenamiento de datos. Están incluidos en varios estándares de la industria, como por ejemplo: estándar europeo de transmisión por satélite de televisión digital (DVB-S2) [5], red de cableado residencial (G.hn/G.9660) [6], red inalámbrica de área local Wi-Fi (IEEE 802.11n) [7], red inalámbrica de acceso metropolitano de banda ancha WiMAX (IEEE 802.16e) [8], red inalámbrica de área personal mmWave WPAN (IEEE 802.15.3c) [9] y 10-Gigabit Ethernet sobre par trenzado no blindado (IEEE 802.3an) [10].

2.1. Matriz de paridad y generadora

Un código LDPC está definido por el espacio nulo de la matriz de comprobación de paridad $\mathbf{H}$. Ésta es una matriz binaria de tamaño $M \times N$, donde N define la longitud de código y M el número de comprobaciones de paridad. El peso se define como el número de unos por fila o columna, siendo ρ_m el peso de la fila m y γ_n el peso de la columna n. Como bien indica su nombre, en la matriz $\mathbf{H}$ de los códigos LDPC los pesos ρ_m y γ_n son muy bajos comparados con N y M, respectivamente, por lo que se dice que la matriz es dispersa o de baja densidad. Para que la matriz de paridad sea válida se debe cumplir que entre dos

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

Figura 2.1: Matriz de comprobación de paridad **H** de tamaño 6×12, rango 5 y pesos de fila y columna 4 y 2, respectivamente.

$$\mathbf{G} = \left[\begin{array}{ccccccc:ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Figura 2.2: Matriz generadora **G** de tamaño 7×12 correspondiente al código LDPC de la Figura 2.1 de longitud $N = 12$ y tasa $R = 7/12$.

filas o dos columnas cualesquiera sólo puede haber como máximo un uno en común (en la misma posición). En una matriz de paridad **H** las filas no son necesariamente linealmente independientes entre si, por lo que la longitud de información, K, depende del rango de la matriz, siendo $K = N - M$ para matrices de rango completo y $K = N - \text{rango}(\mathbf{H})$ en caso contrario. La tasa de código, R, se define como la relación entre longitud de información y longitud de código, siendo $R = K/N$. En la Figura 2.1 se muestra un ejemplo de matriz de paridad **H** cuyas dimensiones son $M = 6$ y $N = 12$. El rango de esta matriz es 5, por lo tanto, la longitud de la información K es 7 y la tasa de código R es igual a 0.58. Los pesos ρ_m son iguales a 4 para todas las filas y los pesos γ_n son iguales a 2 para todas las columnas.

La matriz generadora **G** se obtiene a partir de la matriz de paridad y se utiliza para la codificación. Esta matriz es binaria de tamaño $K \times N$ y está compuesta por la matriz identidad $\mathbf{I}_K$ y una matriz **P**, siendo $\mathbf{G} = [\mathbf{I}_K|\mathbf{P}]$. La matriz **P**, de alta densidad, se obtiene aplicando el método de eliminación de Gauss-Jordan a la matriz **H**. La palabra código $\boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ se obtiene multiplicando la palabra de información $\boldsymbol{c} = (c_1, c_2, \ldots, c_K)$ por la matriz generadora, tal como se muestra en (2.1). Debido a que la matriz generadora contiene la matriz identidad, la palabra código contiene los K bits de información y $N - K$ bits de paridad, en este caso la codificación se denomina sistemática.

$$\boldsymbol{w} = \boldsymbol{c} \cdot \mathbf{G} \tag{2.1}$$

La matriz generadora **G** correspondiente a la matriz de paridad de la Figura 2.1 se muestra en la Figura 2.2. Esta matriz tiene 7 filas y 12 columnas, correspondientes a K y N, respectivamente. Se diferencian claramente las dos submatrices, $\mathbf{I}_K$ y **P**, que determinan la parte sistemática y de paridad del código LDPC.

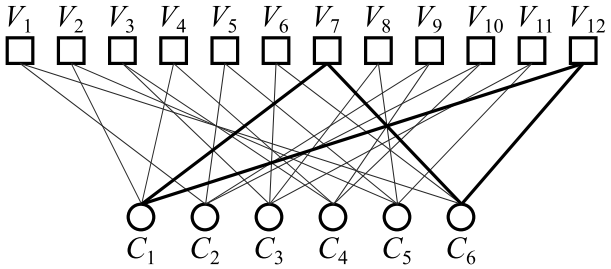


Figura 2.3: Grafo de Tanner definido por la matriz **H** de la Figura 2.1. Los cuadrados corresponden a los nodos C_m y lo círculos a los nodos V_m. El *girth* es 4 y corresponde al ciclo cerrado que se resalta entre los nodos V_7, C_1, V_{12} y C_6.

2.2. Grafo de Tanner

El grafo de Tanner es un grafo bipartito que utiliza **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 si, cuando el elemento $\mathbf{H}_{m,n}$ de la matriz **H** es igual a 1. El número de conexiones de un nodo se define como grado, siendo $d_{C,m}$ el grado de un nodo C_m y $d_{V,n}$ el grado de un nodo V_n. Los grados $d_{C,m}$ y $d_{V,n}$ en el grafo corresponden con los pesos ρ_m y γ_n de la matriz **H**, respectivamente. El *girth* de un grafo se define como la longitud del ciclo simple más corto y se utiliza como medida de las prestaciones de un código LDPC. En general, cuanto mayor es el *girth*, mejor capacidad de corrección tiene el código [11].

El grafo de Tanner correspondiente a la matriz **H** de la Figura 2.1 se muestra en la Figura 2.3. Los nodos C_m se representan mediante cuadrados, los nodos V_m mediante círculos y las conexión entre nodos mediante las líneas. La relación directa entre la matriz **H** y el grafo se puede ver, por ejemplo, en el nodo de bit V_1, el cual está conectado con los nodos de comprobación C_2 y C_6. A su vez, el nodo de comprobación C_1 está conectado con 4 nodos de bit: V_2, V_4, V_7 y V_{12}, coincidiendo así con las posiciones de los unos en la primera columna de la matriz **H**. El *girth* de esta matriz es 4 y corresponde a la longitud del ciclo simple mas corto resaltado en la figura.

2.3. Tipos de código

Los códigos LDPC se pueden dividir en dos grandes grupos: los códigos regulares y los irregulares. Un código es regular cuando la matriz de paridad **H** tiene pesos ρ_m y γ_n constantes, es decir, $\rho_m = \rho$ y $\gamma_n = \gamma$ para todo m y n, respectivamente. En caso contrario, el código se denomina irregular. Desde el punto de vista del grafo de Tanner un código regular es aquel en el que los grados $d_{C,m}$ y $d_{V,n}$ son constantes e iguales a d_C y d_V, respectivamente. Generalmente, los códigos irregulares poseen una mejor capacidad correctora que los códigos regulares [12].

A nivel estructural, los códigos LDPC se pueden dividir en aleatorios y estructurados, siendo los códigos estructurados aquellos en los que la matriz de comprobación de paridad **H** presenta una estructura regular. Los códigos LDPC aleatorios, sin una estructura defini-

$$\mathbf{H} = \left[\begin{array}{cccc:cccc:cccc:cccc:cccc}
1&0&0&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&0&1\\
0&1&0&0&0&0&0&1&0&1&0&0&0&0&1&0&1&0&0&0\\
0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0\\
0&0&0&1&0&1&0&0&0&0&0&1&1&0&0&0&0&0&1&0\\
\hdashline
0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&1&1&0&0&0&0&1&0&0&0&0&1&0\\
1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
\hdashline
0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0\\
1&0&0&0&0&1&0&0&0&0&0&1&1&0&0&0&0&0&1&0\\
0&1&0&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&0&1\\
0&0&1&0&0&0&0&1&0&1&0&0&0&0&1&0&1&0&0&0
\end{array}\right]$$

Figura 2.4: Matriz de comprobación de paridad **H** de un código estructurado con dimensiones $M_b = 3$, $N_b = 5$ y $z = 4$. El tamaño es 12×20 y los pesos ρ_m y γ_n son 5 y 3, respectivamente.

da, se alcanzan unas altas prestaciones. Sin embargo, la implementación *hardware*, tanto del codificador como del decodificador, para estos códigos resulta demasiado compleja. En la práctica se usan códigos estructurados debido a que con éstos se consiguen prestaciones similares a los aleatorios y además, pueden ser diseñados para que las implementaciones *hardware* de los codificadores y decodificadores sean eficientes y de relativa baja complejidad [13]. Los códigos estructurados se generan dividiendo la matriz de comprobación de paridad **H**, en $M_b \times N_b$ submatrices cuadradas, donde cada submatriz tiene un tamaño $z \times z$. El valor de M y N de la matriz **H** resultante es $M_b \cdot z$ y $N_b \cdot z$, respectivamente. Estas submatrices pueden ser matrices regulares con pesos de fila y columna unitarios o matrices nulas, por lo que el máximo valor de ρ_m es N_b y el máximo valor de γ_n es M_b. En la Figura 2.4 se muestra una matriz de paridad **H** de un código estructurado cuyas dimensiones son $M_b = 3$, $N_b = 5$ y $z = 4$. Este código es regular con pesos ρ_m y γ_n iguales a 5 y 3, respectivamente. Otros ejemplos de códigos estructuradas son los códigos cuasi-cíclicos (QC) [14], los códigos irregulares de repetición y acumulación (IRA) [15], y los códigos basados en Reed-Solomon (RS) [16].

Para el desarrollo de este trabajo se han usado diferentes códigos LDPC de los estándares IEEE 802.15.3c, IEEE 802.16e e IEEE 802.3an. Estos códigos son estructurados y sus principales características se resumen en la Tabla 2.1. Las columnas ρ_m y γ_n muestran los valores de peso de la matriz de comprobación de paridad, siendo un único valor en cada caso para códigos regulares y un conjunto de valores cuando el código es irregular.

Tabla 2.1: Características de los códigos LDPC utilizados en este trabajo.

Estándar [Tipo]	M	N	K	z	ρ_m	γ_n	$girth$
IEEE 802.15.3c	336	672	336	21	6, 7, 8	1, 3, 4	6
[QC-irregular]	168	672	504	21	13, 14, 15, 16	1, 2, 3, 4	6
	84	672	588	21	29, 30, 31, 32	1, 2, 3, 4	4
IEEE 802.16e	1152	2304	1152	96	6, 7	2, 3, 6	6
[IRA-irregular]	768	2304	1536	96	10	2, 3, 6	6
	576	2304	1728	96	14, 15, 16	2, 3, 4, 5	6
	384	2304	1920	96	20	2, 3, 4	6
IEEE 802.3an	384	2048	1723	64	32	6	6
[RS-regular]							

Capítulo 3

DECODIFICACIÓN DE CÓDIGOS LDPC

Los algoritmos del tipo de intercambio de mensajes [17] 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 y los nodos de bit, a través de las conexiones del grafo de Tanner. Su diagrama de flujo general se muestra en la Figura 3.1. En la inicialización, el valor de los mensajes λ se carga con la información de entrada l. La fase iterativa consta de dos procesos de actualización: primero se actualizan los nodos de comprobación usando los mensajes generados por los nodos de bit, λ, y después se actualizan los nodos de bit, procesando la información de entrada, l, y los mensajes provenientes de los nodos de comprobación, μ. El proceso iterativo termina cuando se cumple el criterio de finalización, que puede ser un máximo de iteraciones o la detección de una palabra código válida. Si existe un criterio de finalización diferente al número máximo de iteraciones, se dice que el algoritmo dispone de finalización anticipada. Cada elemento l_n del mensaje l corresponde a la fiabilidad intrínseca del canal para el n-ésimo símbolo recibido. Esta fiabilidad puede ser la razón de verosimilitud (*Log-Likelihood Ratio* - LLR), la decisión dura (*hard*) o el propio valor del símbolo.

Entre los algoritmos de intercambio de mensajes destaca el algoritmo Sum-Product (SP) [18], el cual se usa como referencia. Este algoritmo es complejo a nivel computacional, sin embargo, sus prestaciones son muy altas. Otro algoritmo que se usa como referencia es el Min-Sum (MS) [19]. Éste es una aproximación del algoritmo SP, que consigue una complejidad computacional menor a costa de unas peores prestaciones. En [20] se propone el uso de factores de corrección para mejorar las prestaciones del algoritmo MS. Otros, como el algoritmo λ-Min [21] utilizan factores de corrección variables y dependientes de las entradas. En [22] el factor de corrección también es variable pero en este caso depende del estado del nodo de comprobación en cada iteración. Por otra parte, otros trabajos se han centrado en reducir la complejidad aunque las prestaciones sean perores. Por ejemplo, en [23] y [24] se propone la exclusión de la marginalización en el algoritmo MS con el fin de calcular un único mínimo y se aplican factores de corrección para que las pérdidas de las prestaciones no sea excesivas. En el algoritmo *Split-Row* MS, propuesto en [25], se divide la matriz de paridad en bloques y el cálculo de los mínimos se realiza de forma casi

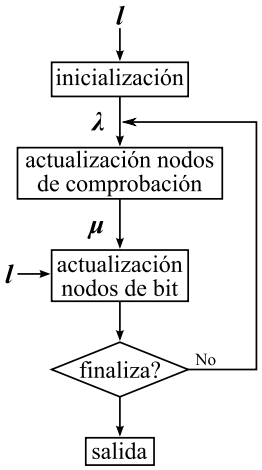


Figura 3.1: Diagrama de flujo de los algoritmos del tipo de intercambio de mensajes basados en dos fases o inundación.

independientes (se intercambia poca información) para cada bloque. Este algoritmo incurre en pérdidas de prestaciones, sin embargo, en [26] los mismos autores proponen el uso de un umbral que permite mejorarlas. En general, la mayoría de algoritmos propuestos, así como las lineas de investigación existentes, se basan en modificaciones de los algoritmos SP o MS. En este capitulo se exponen dichos algoritmos y sus principales variantes.

Para el desarrollo de este trabajo se asume que una palabra código binaria, definida como $\boldsymbol{w} = (w_1, w_2, \ldots, w_N)$, es transmitida por un canal con ruido aditivo blanco Gaussiano (*Additive White Gaussian Noise* - AWGN) de media cero y varianza σ^2, usando la modulación de fase binaria (*Binary Phase Shift Keying* - 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. El conjunto de nodos de bit adyacentes a un nodo de comprobación C_m se define como $N_m = \{n : \mathbf{H}_{m,n} = 1\}$ y al conjunto de nodos de comprobación adyacentes a un nodo de bit V_n como $M_n = \{m : \mathbf{H}_{m,n} = 1\}$. 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 $M_{n \backslash m}$. 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 (Figura 3.2). 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.

3.1. Algoritmo Sum-Product

El algoritmo Sum-Product (SP) es una algoritmo subóptimo para la decodificación de códigos LDPC [18]. Sus prestaciones en términos de capacidad correctora son muy buenas. Sin embargo, su complejidad computacional es elevada debido a que hace uso de funciones no lineales [27]. A pesar de ello el algoritmo SP es ampliamente utilizado y constituye un referente para la mayoría de algoritmo conocidos.

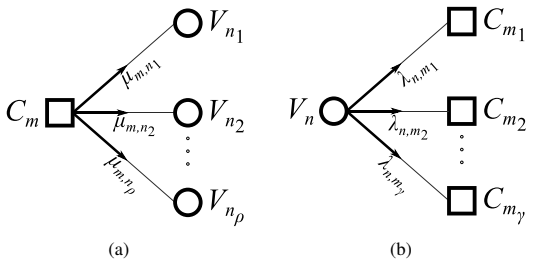


Figura 3.2: Envío de mensajes entre nodos del grafo de Tanner. (a) El nodo de comprobación C_m envía los mensajes $\mu_{m,n}$ al los nodos de bit adyacentes V_n definidos por el conjunto N_m de ρ elementos. (b) El nodo de bit V_n envía los mensajes $\lambda_{n,m}$ a los nodos de comprobación adyacentes C_m definidos por el conjunto M_n de γ elementos.

El proceso de decodificación del algoritmo SP se divide en cuatro pasos que se describen a continuación:

1. **Inicialización:** Para cada $n \in \{1,\dots,N\}$ y $m \in M_n$ se inician los mensajes que enviarán los nodos de bit a los nodos de comprobación.

$$\lambda_{n,m}^{(0)} = l_n$$

2. **Actualización de los nodos de comprobación:** Para cada $m \in \{1,\dots,M\}$ y $n \in N_m$ se calculan los mensajes a enviar por los nodos de comprobación los mensajes recibidos desde los nodos de comprobación de la iteración anterior.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \Psi^{-1}\Big(\sum_{n' \in N_{m\setminus n}} \Psi(|\lambda_{n',m}^{(i-1)}|)\Big) \tag{3.1}$$

   donde, las funciones no lineales $\Psi(x)$ y $\Psi(x)^{-1}$ se definen como $\log(\tanh(|x|/2))$.

   El término $\Gamma_{m,n}^{(i)}$ es la actualización de paridad y se define como,

$$\Gamma_{m,n}^{(i)} = \prod_{n' \in N_{m\setminus n}} \operatorname{sign}(\lambda_{n',m}^{(i-1)})$$

3. **Actualización de los nodos de bit:** Para cada $n \in \{1,\dots,N\}$ y $m \in M_n$ se calculan los mensajes a enviar por los nodos de bit usando la información *a priori* del canal y los mensajes que provienen de los nodos de comprobación del paso anterior.

$$\lambda_{n,m}^{(i)} = l_n + \sum_{m' \in M_{n\setminus m}} \mu_{m',n}^{(i)}$$

   Esta suma en la que se excluye el elemento m del conjunto M_n se puede escribir como la suma de todos los elementos del conjunto M_n menos el elemento n, donde $\lambda_n^{(i)}$ es el LLR *a posteriori* del bit n.

$$\lambda_n^{(i)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(i)} \tag{3.2}$$

$$\lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \mu_{m,n}^{(i)} \tag{3.3}$$

4. **Decisión:** Para cada $n \in \{1,\ldots,N\}$ se realiza la decisión *hard* de los LLR *a posteriori*, obteniendo así, la palabra código estimada $\boldsymbol{z}^{(i)}$.

$$z_n^{(i)} = \begin{cases} 1 & \lambda_n^{(i)} \geq 0 \\ 0 & \lambda_n^{(i)} < 0 \end{cases} \tag{3.4}$$

La comprobación de la paridad se realiza mediante el producto de la palabra código con la matriz de comprobación de paridad $\mathbf{H}$. Si se cumple la ecuación de paridad, $\boldsymbol{z}^{(i)} \cdot \mathbf{H}^{\mathrm{T}} = 0$, el algoritmo finaliza anticipadamente. En caso contrario, el algoritmo vuelve al paso 2, para así continuar con la siguiente iteración, $i = i+1$, hasta alcanzar el número máximo de iteraciones, $i = I_{max}$.

3.2. Algoritmo Min-Sum

El algoritmo Min-Sum (MS), propuesto por Fossorier en [19], es una variación del algoritmo SP donde la función no lineal $\Psi(x)^{-1}$ en (3.1) se aproxima al cálculo del valor mínimo, denotado como $\min(x)$. Con esta aproximación se simplifica el algoritmo debido a que la complejidad del cálculo de los mensajes que envían los nodos de comprobación es menor. Sin embargo, la aproximación añade un error que empeora las prestaciones del algoritmo. La actualización de los nodos de comprobación del algoritmo MS se muestra en (3.5). La inicialización, actualización de los nodos de bit y decisión son exactamente iguales a las del algoritmo SP.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|) \tag{3.5}$$

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 operación $\min(x)$ en (3.5) se puede escribir como se muestra en (3.6). Esto se aprovechará para reducir el la complejidad *hardware* y el coste de almacenamiento de los mensajes en las arquitecturas que implementan este algoritmo.

$$\min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|) = \begin{cases} min1 & \text{si} \quad n' \neq n \\ min2 & \text{si} \quad n' = n \end{cases} \tag{3.6}$$

3.2.1. Min-Sum escalado y con *offset*

En [20] se proponen dos mejoras al algoritmo MS con las que se compensa la perdida de las prestaciones debida a la aproximación de la función $\Psi(x)^{-1}$. Los dos algoritmos se denominan Min-Sum escalado (αMS) y Min-Sum con *offset* (βMS). En éstos se aplica un factor de corrección al mensaje generado por el nodo de comprobación. En el algoritmo αMS el factor de corrección multiplica al mensaje, mientras que en el algoritmo βMS el factor de corrección se resta al mensaje. Las ecuaciones (3.7) y (3.8) muestran la actualización de los nodos de comprobación de los algoritmos αMS y βMS, respectivamente.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \alpha \cdot \min_{n' \in N_{m \setminus n}} (|\lambda_{n',m}^{(i-1)}|) \tag{3.7}$$

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \max\left(\min_{n' \in N_{m \setminus n}} (|\lambda_{n',m}^{(i-1)}|) - \beta, 0 \right) \tag{3.8}$$

Como se muestra en (3.8), el algoritmo βMS requiere una resta y una comparación con cero, por lo que la complejidad añadida de este algoritmo es baja respecto al MS. En cambio, el algoritmo αMS requiere una multiplicación cuya complejidad es mayor. Sin embargo, usando un factor de corrección de la forma $1 - 2^{-x}$ la multiplicación se reduce a una resta, y por lo tanto, la complejidad añadida es aun menor que la del algoritmo βMS.

Las prestaciones de los algoritmos αMS y βMS son óptimas cuando los factores de corrección varían en función del número de iteraciones y de la relación señal a ruido (*Signal-to-Noise Ratio* - SNR) [28]. En la práctica, estos factores son constantes. Esto es debido a la complejidad que añade este hecho y a que las prestaciones con un factor de escalado u *offset* fijo son muy cercanas a las del algoritmo SP.

3.2.2. Min-Sum modificado

Se denomina Min-Sum modificado (mMS) a la versión del algoritmo MS en la que se elimina la marginalización en las operaciones de actualización de comprobación, como se muestra en (3.9). La no exclusión del elemento n en la operación $\min(x)$ hace que la actualización de comprobación sea menos compleja que la del algoritmo MS, debido a que solamente es necesario calcular un mínimo para el conjunto N_m. Sin embargo, las prestaciones de este algoritmo son considerablemente peores que las del algoritmo MS para la mayoría de los códigos LDPC.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_m} (|\lambda_{n',m}^{(i-1)}|) \tag{3.9}$$

3.3. Ordenación de la actualización de los mensajes

El orden en el que se actualizan los mensajes en los nodos de comprobación y de paridad modifica la velocidad de convergencia de los algoritmos de decodificación. Los métodos de actualización más conocidos son el método de dos fases o por inundación [2], el método por capas horizontales o *layered* [29] y el método por capas verticales o *shuffled* [30]. La actualización por inundación es la forma básica mientras las actualizaciones *layered* y *shuffled* son mejoras que aceleran la convergencia de los algoritmos.

3.3.1. Actualización por inundación

En el método por inundación las actualizaciones de los nodos de comprobación y de bit se realiza en dos fases. En la primera fase todos los nodos de comprobación C_m calculan los mensajes que enviarán a los nodos de bit y en la segunda fase todos los nodos de bit V_n calculan los mensajes que van desde los nodos de bit a los nodos de comprobación. Por consiguiente, la actualización de los nodos de comprobación no comenzará hasta que todos los nodos de bit sean actualizados y viceversa.

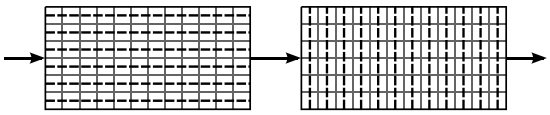


Figura 3.3: Método de actualización por inundación para una matriz **H** de tamaño 6×12, donde las lineas discontinuas verticales representan la actualización de los nodos de bit y las horizontales la actualización de los nodos de comprobación.

La representación gráfica del método de actualización por inundación para una matriz de paridad de tamaño 6×12 se muestra en la Figura 3.3. Las lineas discontinuas verticales representan la actualización de los nodos de bit y las horizontales la actualización de los nodos de comprobación.

A manera de referencia, el pseudocódigo 3.1 describe el algoritmo MS con el método de actualización por inundación. Por simplicidad, en esta descripción no se contempla la finalización anticipada.

Pseudocódigo 3.1 Algoritmo MS con actualización por inundación.

– *inicialización* –
para $n \in \{1,\ldots,N\}$ y $m \in M_n$
    $\lambda_{n,m}^{(0)} = l_n$
– *iteraciones* –
para $i \in \{1,\ldots,I_{max}\}$
    – *actualización de los nodos de comprobación* –
    para $m \in \{1,\ldots,M\}$ y $n \in N_m$
        $\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)$
    – *actualización de los nodos de bit* –
    para $n \in \{1,\ldots,N\}$ y $m \in M_n$
        $\lambda_n^{(i)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(i)}$
        $\lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \mu_{m,n}^{(i)}$
– *decodificación hard* –
para $n \in \{1,\ldots,N\}$
    $z_n = \begin{cases} 1 & \lambda_n^{(I_{max})} \geq 0 \\ 0 & \lambda_n^{(I_{max})} < 0 \end{cases}$

3.3.2. Actualización por capas horizontales - *Layered*

En el método de actualización *layered* los M nodos de comprobación se dividen en G grupos, que son actualizados alternadamente. El número de grupos, G, es un valor entero menor o igual al máximo peso de columna γ_n de la matriz de paridad **H**. Además, el valor de G debe ser tal que el resto de la división $M_G = M/G$ sea cero, siendo M_G el número de nodos de comprobación de cada grupo g. Cada iteración se compone de G subiteraciones, en las que se realizan dos fases de actualización: primero se actualiza el g-ésimo grupo de nodos de comprobación y después se actualizan todos los nodos de bit. El intercambio de mensajes parciales entre los grupos de nodos de comprobación y los nodos de bit provoca

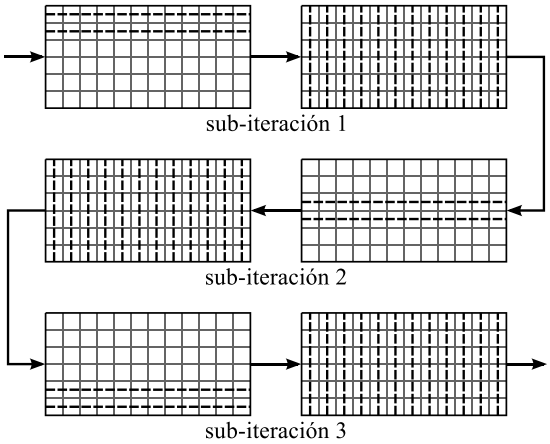


Figura 3.4: Método de actualización *layered* para una matriz **H** de tamaño 6×12, donde las lineas discontinuas verticales representan la actualización de los nodos de bit y las horizontales la actualización de los nodos de comprobación.

que la convergencia del algoritmo de decodificación se acelere. Esto se traduce en que un algoritmo con actualización *layered* alcanza las mismas prestaciones que con la actualización por inundación con menos iteraciones.

En la Figura 3.4 se muestra la representación gráfica del método de actualización *layered* para un número de grupos $G = 3$ y una matriz **H** de tamaño 6×12.

La descripción del algoritmo MS con actualización *layered* se presenta en el pseudocódigo 3.2. Al igual que en la descripción hecha para la actualización por inundación, en esta tampoco se contempla la finalización anticipada.

En el pseudocódigo 3.2 se observa que en la subiteración 1 de la i-ésima iteración, los nodos de bit utilizan los mensajes generados por los nodos de comprobación de los grupos 2 a G de la iteración $i-1$ y los mensajes del grupo 1 de la iteración actual. Además, en la subiteración 2 se utilizan los mensajes de los grupos 3 a G de la iteración $i-1$ y los mensajes de los grupos 1 y 2 de la iteración i. Debido a esto, la suma que se realiza en la actualización de los nodos de bit se puede realizar de manera recursiva, restando el valor de los mensajes generados por los nodos de comprobación correspondientes al grupo g de la iteración anterior y sumando el valor de los mensajes enviados por los nodos de comprobación correspondientes al grupo g de la iteración actual. La línea 13 del pseudocódigo 3.2 escrita de manera recursiva se muestra en (3.10), donde, $(g-1) \cdot M_G < m' \leq g \cdot M_G$ y $\lambda_n^{(0)} = l_n$.

$$\lambda_n^{(i)} = \lambda_n^{(i-1)} - \sum_{m' \in M_n} \mu_{m',n}^{(i-1)} + \sum_{m' \in M_n} \mu_{m',n}^{(i)} \tag{3.10}$$

Pseudocódigo 3.2 Algoritmo MS con actualización *layered*.

1: – *inicialización* –
2: para $n \in \{1,\dots,N\}$ y $m \in M_n$
3: $\quad \lambda_{n,m}^{(0)} = l_n$
4: – *iteraciones* –
5: para $i \in \{1,\dots,I_{max}\}$
6: $\quad$ – *subiteraciones* –
7: $\quad$ para $g \in \{1,\dots,G\}$
8: $\quad\quad$ – *actualización de los nodos de comprobación (grupo)* –
9: $\quad\quad$ para $m \in \{(g-1)\cdot M_G+1,\dots,g\cdot M_G\}$ y $n \in N_m$
10: $\quad\quad\quad \mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)$
11: $\quad\quad$ – *actualización de los nodos de bit* –
12: $\quad\quad$ para $n \in \{1,\dots,N\}$ y $m \in M_n$
13: $\quad\quad\quad \lambda_n^{(i)} = l_n + \sum_{\substack{m' \in M_n \\ m' > g\cdot M_G}} \mu_{m',n}^{(i-1)} + \sum_{\substack{m' \in M_n \\ m' \le g\cdot M_G}} \mu_{m',n}^{(i)}$
14: $\quad\quad\quad \lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \begin{cases} \mu_{m,n}^{(i-1)} & m > g\cdot M_G \\ \mu_{m,n}^{(i)} & m \le g\cdot M_G \end{cases}$
15: – *decodificación hard* –
16: para $n \in \{1,\dots,N\}$
17: $\quad z_n = \begin{cases} 1 & \lambda_n^{(I_{max})} \ge 0 \\ 0 & \lambda_n^{(I_{max})} < 0 \end{cases}$

3.3.3. Actualización por capas verticales - *Shuffled*

El método de actualización *shuffled*, también llamado *layered* vertical, se basa en el intercambio de resultados parciales como el método de actualización *layered*. En este caso, son los N nodos de bit los que se dividen en G grupos. El valor de G debe ser entero, menor al máximo peso de fila ρ_m y debe cumplir que el resto de la división $N_G = N/G$ sea cero, donde N_G es el número de nodos de bit por grupo. Al igual que en la actualización *layered* cada iteración se compone de G subiteraciones, en las que primero se actualizan todos los nodos de comprobación y luego el g-ésimo grupo de nodos de bit.

Los algoritmos con actualización *shuffled* tienen prestaciones similares a las de la actualización *layered*. La representación gráfica del método de actualización *shuffled* para una matriz $\mathbf{H}$ de tamaño 6×12 se muestra en la Figura 3.5, donde el número de grupos G es igual a 3.

En el pseudocódigo 3.3 se describe el algoritmo MS con actualización *shuffled*. En este caso, el término $\Gamma_{m,n}^{(i)}$ depende de la subiteración g de la misma manera que el mensaje $\mu_{m,n}^{(i)}$. Por lo tanto, el cálculo de $\Gamma_{m,n}^{(i)}$ se hace con los signos de de los mensajes generados por los nodos de bit de la iteración $i-1$ si $n > (g-1)\cdot N_G$ y de la iteración i si $n \le (g-1)\cdot N_G$ como se muestra en (3.11).

$$\Gamma_{m,n}^{(i)} = \prod_{\substack{n' \in N_{m\setminus n} \\ n' > (g-1)\cdot N_G}} \text{sign}(\lambda_{n',m}^{(i-1)}) \cdot \prod_{\substack{n' \in N_{m\setminus n} \\ n' \le (g-1)\cdot N_G}} \text{sign}(\lambda_{n',m}^{(i)}) \tag{3.11}$$

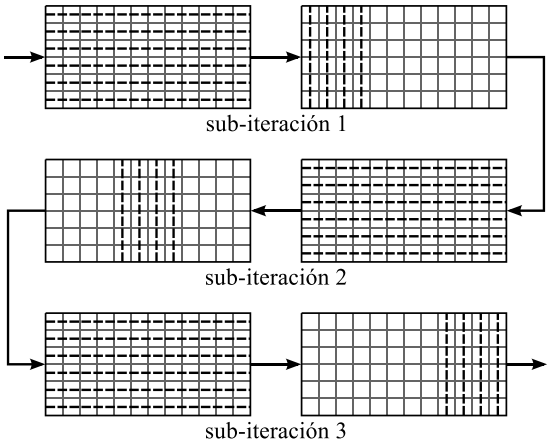


Figura 3.5: Método de actualización *shuffled* para una matriz $\mathbf{H}$ de tamaño 6×12, donde las lineas discontinuas verticales representan la actualización de los nodos de bit y las horizontales la actualización de los nodos de comprobación.

Pseudocódigo 3.3 Algoritmo MS con actualización *shuffled*.

– *inicialización* –
para $n \in \{1,\ldots,N\}$ y $m \in M_n$
  $\lambda_{n,m}^{(0)} = l_n$
– *iteraciones* –
para $i \in \{1,\ldots,I_{max}\}$
  – *subiteraciones* –
  para $g \in \{1,\ldots,G\}$
    – *actualización de los nodos de comprobación* –
    para $m \in \{1,\ldots,M\}$ y $n \in N_m$
      $\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min\Big(\min_{\substack{n' \in N_{m\setminus n} \\ n' > (g-1)\cdot N_G}} (|\lambda_{n',m}^{(i-1)}|), \min_{\substack{n' \in N_{m\setminus n} \\ n' \leq (g-1)\cdot N_G}} (|\lambda_{n',m}^{(i)}|) \Big)$
    – *actualización de los nodos de bit (grupo)* –
    para $n \in \{(g-1)\cdot N_G + 1,\ldots,g\cdot N_G\}$ y $m \in M_n$
      $\lambda_n^{(i)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(i)}$
      $\lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \mu_{m,n}^{(i)}$
– *decodificación hard* –
para $n \in \{1,\ldots,N\}$
  $z_n = \begin{cases} 1 & \lambda_n^{(I_{max})} \geq 0 \\ 0 & \lambda_n^{(I_{max})} < 0 \end{cases}$

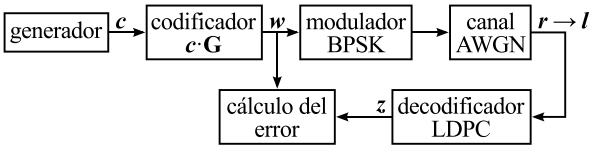


Figura 3.6: Esquema de simulación para la evaluación de prestaciones de los algoritmos de decodificación.

3.4. Análisis de prestaciones

El análisis de prestaciones de los diferentes algoritmos y métodos de actualización se ha realizado usando el esquema de simulación mostrado en la Figura 3.6. El paquete de datos binarios, generado en el bloque generador, se codifica y se transmite por un canal AWGN usando modulación BPSK. La palabra decodificada se compara con la transmitida (información y paridad) y se calcula el número de bits erróneos. Un paquete se considera erróneo si existe por lo menos un bit erróneo en dicho paquete, ya sea en la información o en la paridad. Para cada relación señal a ruido (SNR) se realizan varias simulaciones hasta encontrar un máximo de 50 paquetes erróneos y se calculan la tasa de error de bit (*Bit Error Rate* - BER) y la tasa de error de paquete (*Packet Error Rate* - PER). Los algoritmos simulados implementan la finalización anticipada y en todos se ha fijado el número máximo de iteraciones I_{max} a 30.

La evaluación de los algoritmos SP, MS, αMS, βMS y mMS se ha hecho con el método de actualización por inundación para los códigos LDPC de los estándares IEEE 802.15.3c, IEEE 802.16e e IEEE 802.3an, listados en la Tabla 2.1. Los factores de corrección α que se han utilizados son del tipo $1-2^{-x}$ para que la complejidad de la operación de escalado sea mínima, siendo estos 0.5, 0.75 y 0.875. Por otro lado, los factores de corrección β utilizados van desde 0.25 hasta 2, en pasos de 0.25. Estos valores se han escogido porque su representación binaria requiere pocos bits. El análisis de prestaciones de los métodos de actualización *layered* y *shuffled* se ha realizado con el algoritmo SP para el código del estándar IEEE 802.3an. La actualización *layered* se ha evaluado con 2, 3 y 6 grupos, y la *shuflled* con 2, 4, 8, 16 y 32 grupos.

3.4.1. Prestaciones de los algoritmos

Las curvas de tasa de error de los diferentes algoritmos con los códigos del estándar IEEE 802.15.3c se muestran en las Figuras 3.7, a 3.9. Estos códigos son QC-irregulares de longitud $N = 672$ y tasas $1/2$, $3/4$ y $7/8$. Para los algoritmos αMS y βMS solamente se muestran las curvas con el factor de corrección que mayores prestaciones proporciona. Se observa que el factor α es igual en los tres casos (0.75), mientras que el factor β depende del código (0.75, 1 y 1.25). Para un PER de 10^{-5} las mejores prestaciones las consigue el algoritmo SP en todos lo casos. Sin embargo, los algoritmos αMS y βMS consiguen similares prestaciones con pérdidas menores a 0.07 dB. El algoritmo MS presenta pérdidas mayores, 0.5, 0.3 y 0.25 dB para las tasas $1/2$, $3/4$ y $7/8$, respectivamente. Es importante destacar el deterioro en las prestaciones del algoritmo mMS, cuyas pérdidas son mayores a 4 dB.

Las curvas de tasa de error con los códigos del estándar IEEE 802.16e se muestra en

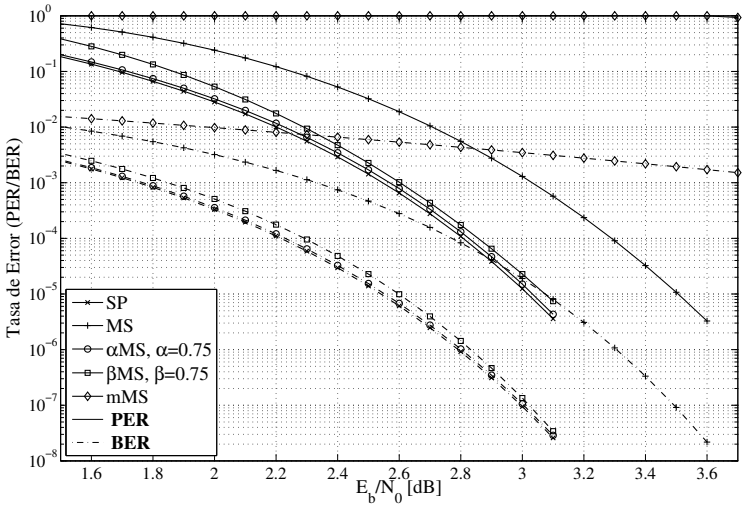


Figura 3.7: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.15.3c de longitud $N = 672$ y tasa $R = 1/2$ con 30 iteraciones.

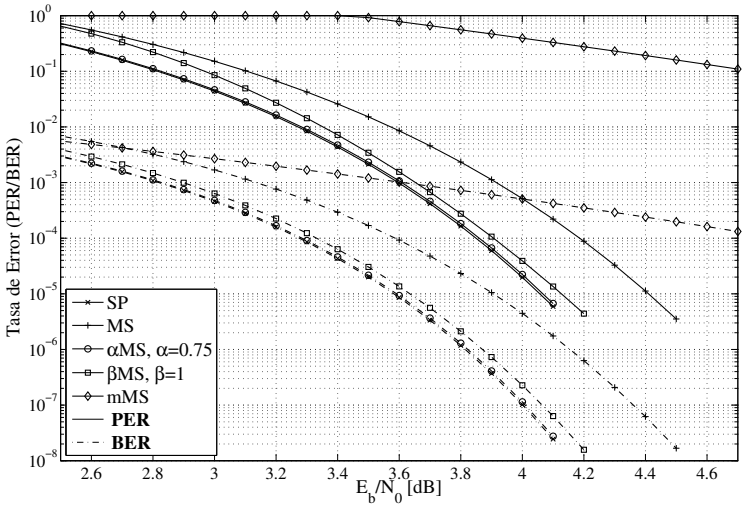


Figura 3.8: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.15.3c de longitud $N = 672$ y tasa $R = 3/4$ con 30 iteraciones.

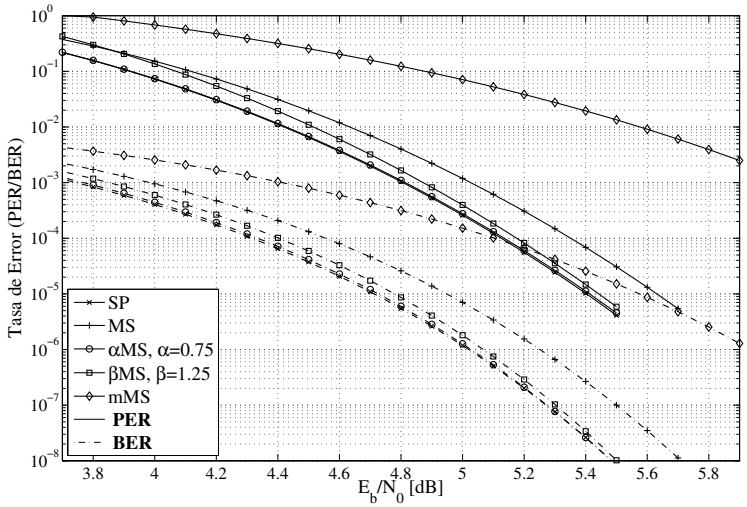


Figura 3.9: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.15.3c de longitud $N = 672$ y tasa $R = 7/8$ con 30 iteraciones.

las Figuras 3.10 a 3.13. Estos códigos son IRA-irregulares de longitud 2304 y tasas $1/2$, $2/3$, $3/4$ y $5/6$. Al igual que con los códigos del estándar IEEE 802.15.3c, para un PER de 10^{-5} el algoritmo SP presenta la mayor capacidad de corrección y los algoritmos αMS y βMS unas pérdidas muy bajas (menos de 0.05 dB). También se repite el comportamiento del algoritmo mMS, con pérdidas considerables, por encima de los 4 dB. Las perdidas del algoritmo MS son de 0.37 dB para las tasas $1/2$, $2/3$ y $3/4$, y de 0.2 dB para la tasa $5/6$. Con estos códigos el factor α no es constante, siendo 0.875 para la tasa $5/6$ y 0.75 en el resto de los casos. El factor β tampoco es constante y varía entre 0.5 y 1. Se puede decir que el comportamiento de los algoritmos con estos códigos es similar al visto con los códigos del estándar IEEE 802.15.3c, cuya matriz de paridad también es irregular, pero con una longitud de código menor.

La Figura 3.14 muestra las curvas de tasa de error para el código del estándar IEEE 802.3an. Este código es del tipo RS-regular, con longitud $N = 2048$ y tasa $R = 1723/2048$. Para un PER de 10^{-5} las perdidas de los algoritmos αMS y βMS respecto al algoritmo SP son de 0.06 y 0.02 dB, respectivamente. Las pocas pérdidas de los algoritmos αMS y βMS contrastan con los 0.56 dB de pérdida de los algoritmos MS y mMS.

3.4.2. Prestaciones de los métodos de actualización

Las curvas de convergencia del algoritmo SP con los métodos de actualización por inundación y *layered* se muestran en la Figura 3.15. En éstas se representa la variación de la relación señal a ruido en función del número de iteraciones para un PER de 10^{-5}. Se observa que con el método de actualización por inundación las perdidas de prestaciones con 20 iteraciones respecto a 30 son de aproximadamente 0.05 dB. Además, con la actualización por inundación con 20 iteraciones se consiguen las mismas prestaciones que con una

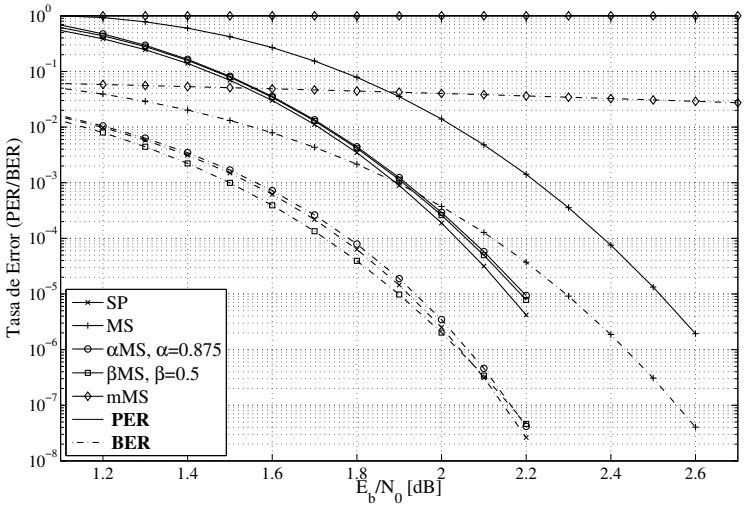


Figura 3.10: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 1/2$ con 30 iteraciones.

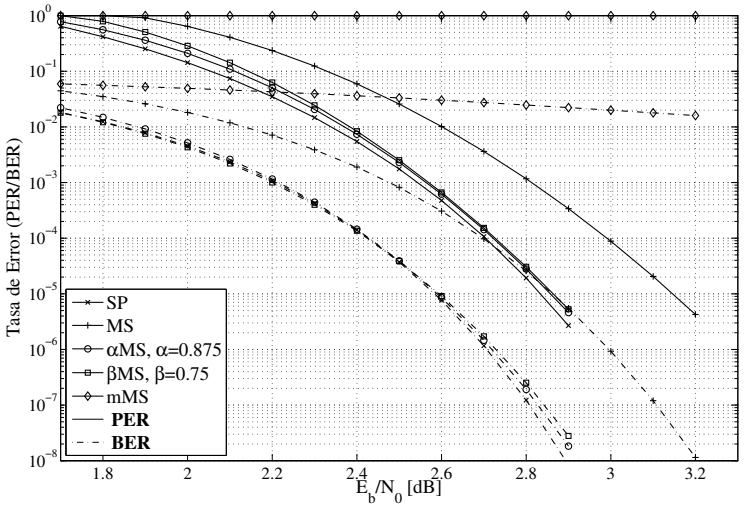


Figura 3.11: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 2/3$ con 30 iteraciones.

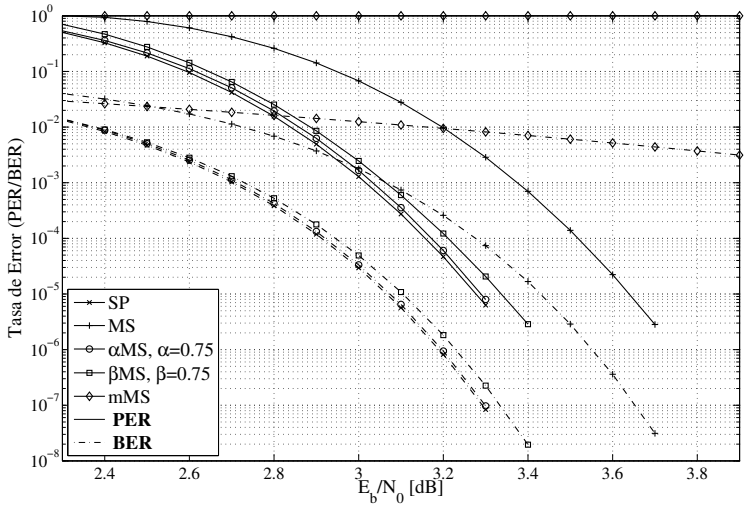


Figura 3.12: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 3/4$ con 30 iteraciones.

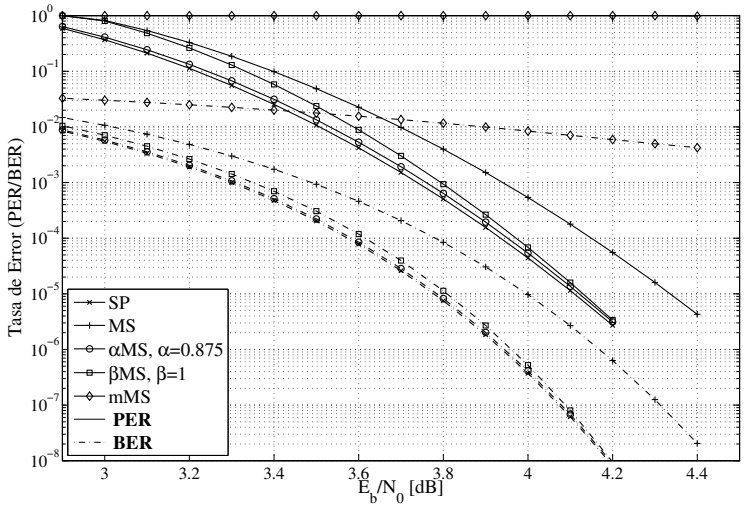


Figura 3.13: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 5/6$ con 30 iteraciones.

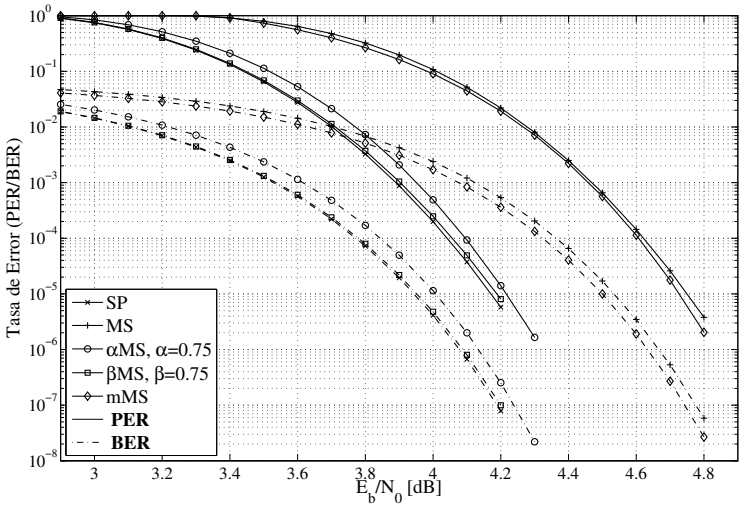


Figura 3.14: Tasa de error de los algoritmos SP y basados en MS para el código del estándar IEEE 802.3an de longitud $N = 2048$ y tasa $R = 1723/2048$ con 30 iteraciones.

actualización *layered* de 6 grupos en tan solo 10 iteraciones. Esta mejora en la convergencia también se produce usando 2 y 3 grupos, aunque en menor medida, siendo necesarias 14 y 12 iteraciones, respectivamente. En general, con la actualización *layered* de 6 grupos se necesita la mitad de iteraciones que con la actualización por inundación para alcanzar las mismas prestaciones.

En la Figura 3.16 se muestran las curvas de convergencia del algoritmo SP con el método de actualización *shuffled* y se incluye la curva con el método por inundación, a manera de referencia. Se observa que con este método hay una mejora en la convergencia, al igual que con el método de actualización *layered*. Por ejemplo, usando la actualización por inundación con 20 iteraciones se consigue el PER de 10^{-5} con una relación señal a ruido de 4.22 dB, la misma que se necesita con la actualización *shuffled* de 2, 4, 8, 16 y 32 grupos con 15, 13, 12, 11 y 11 iteraciones, respectivamente. Otra cosa a destacar es la similitud que hay entre las curvas de convergencia correspondientes a 8 y 16 grupos, siendo prácticamente iguales a partir de 18 iteraciones.

3.5. Análisis de precisión finita

El análisis del efecto que tiene la precisión finita sobre las prestaciones de los algoritmos de decodificación se ha realizado mediante simulación, siguiendo el mismo esquema descrito en la Sección 3.4. Se han evaluado los algoritmos SP, MS, αMS y βMS con diferentes esquemas de cuantificación. Los resultados del algoritmo mMS no se muestran debido a que con precisión finita presenta problemas de convergencia, por tanto, sus prestaciones son muy bajas. Se han utilizado dos tipos de códigos, el código regular de longitud 2048 y tasa 0.84 del estándar IEEE 802.3an y el código irregular de longitud 2304 y tasa 0.83 del estándar IEEE 802.16e. Se ha usado un esquema de cuantificación uniforme con saturación,

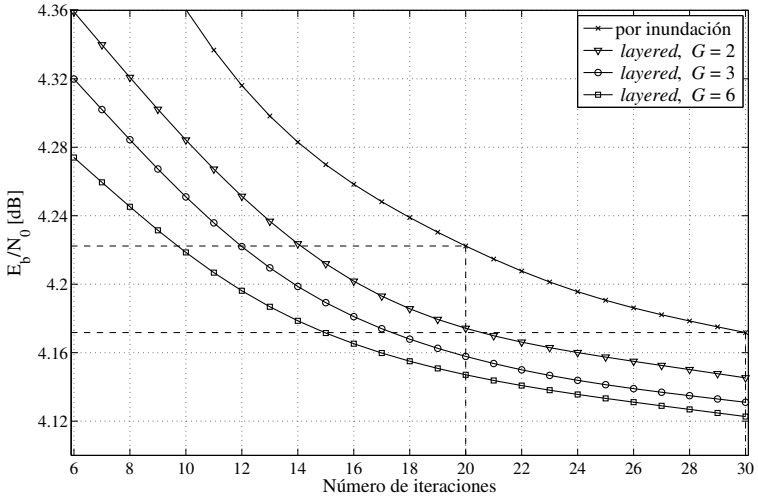


Figura 3.15: Convergencia del algoritmo SP con actualización *layered* para el código del estándar IEEE 802.3an con PER = 10^{-5}.

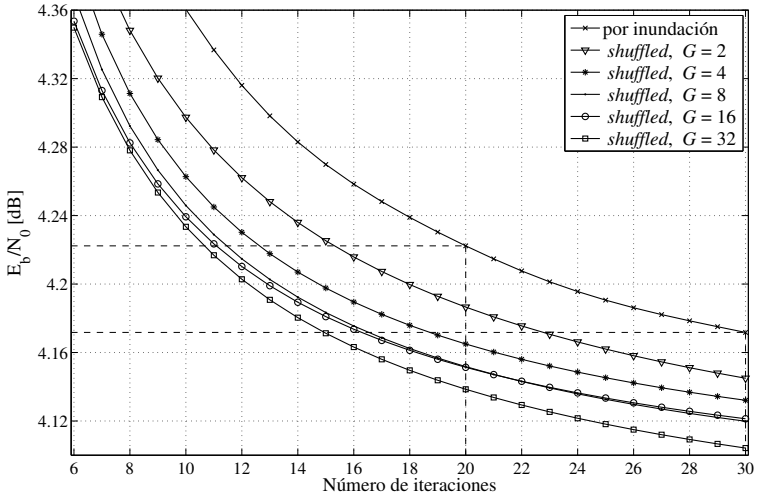


Figura 3.16: Convergencia del algoritmo SP con actualización *shuffled* para el código del estándar IEEE 802.3an con PER = 10^{-5}.

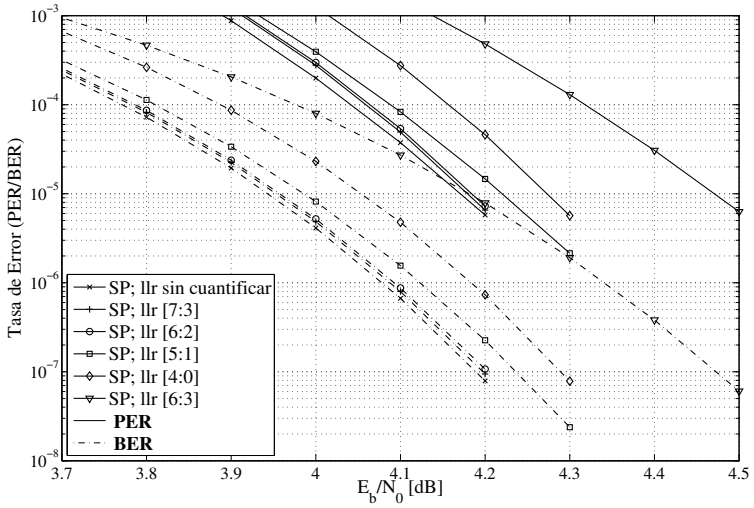


Figura 3.17: Tasa de error del algoritmo SP cuantificado para el código del estándar IEEE 802.3an de longitud $N = 2048$ y tasa $R = 1723/2048$ con 30 iteraciones.

el cual se denota como $[q : f]$, donde q corresponde al número total de bits, incluido el signo, y f al número de bits fraccionales.

Como punto de partida se analiza el efecto que tiene cuantificar la fiabilidad intrínseca del canal sobre las prestaciones del algoritmo SP. Para ello se han utilizado varios esquemas de cuantificación y se han evaluado sus prestaciones en cada caso. Las Figuras 3.17 y 3.18 muestran las curvas de tasa de error para los dos códigos utilizados. Se puede ver en ambos casos que las prestaciones del algoritmo con los esquemas $[7 : 3]$ y $[6 : 2]$ para un PER de 10^{-5} son muy similares entre si, y que las perdidas de estos esquemas respecto al no cuantificado son menores a 0.02 dB. Para el código del estándar IEEE 802.3an, se observa que con el esquema $[5 : 1]$ las pérdidas son de 0.09 dB, y con el esquema $[4 : 0]$, aumentan a 0.17 dB. En las curvas correspondientes al código del estándar 802.16e se aprecia que las pérdidas con el esquema $[5 : 1]$ son de aproximadamente 1 dB y la del esquema $[4 : 0]$ mayores a 1.5 dB. Por otro lado, el disminuir el número de bits enteros hace que las perdidas sean considerablemente altas. Con estos resultados se deduce que la mejor relación entre número de bits y prestaciones se obtiene con los esquemas $[6 : 2]$ y $[5 : 1]$. Estos últimos se han usado como referencia para el análisis de precisión finita de los algoritmos.

Para el análisis de precisión finita de los algoritmos se han cuantificado los mensajes generados por los nodos de bit y los nodos de comprobación. En la suma que se realiza dentro de los nodos de bit se han dejado crecer los datos de forma natural. En el algoritmo SP, también se han cuantificado las funciones no lineales $\Psi(x)$ y $\Psi(x)^{-1}$ de los nodos de comprobación. Para los esquemas de referencia, $[6 : 2]$ y $[5 : 1]$, la función $\Psi(x)$ está acotada entre 0 y -2, por lo que sólo son necesarios dos bits enteros para su cuantificación. En la suma de las funciones $\Psi(x)$ se han dejado crecer los datos de forma natural, sin embargo, el resultado se ha saturado y truncado para limitar el ancho de palabra. Mediante simulación se ha obtenido que con un esquema $[9 : 7]$ para la función y un esquema $[6 : 4]$ para el resultado

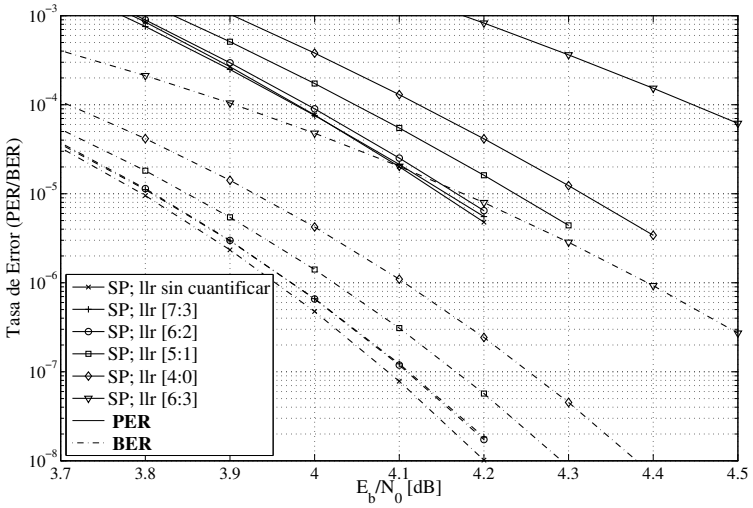


Figura 3.18: Tasa de error del algoritmo SP cuantificado para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 5/6$ con 30 iteraciones.

de la suma se consigue la mejor relación entre las prestaciones y el número de bits para los dos esquemas de referencia. La función inversa $\Psi(x)^{-1}$ se ha cuantificado con el mismo esquema que los mensajes, $[6:2]$ y $[5:1]$. Por lo tanto, se han usado los mismos esquemas de referencia para cuantificar la información del canal, los mensajes enviados por los nodos de bit y los mensajes enviados por los nodos de comprobación.

La Tabla 3.1 muestra las pérdidas de los algoritmos SP, MS, αMS y βMS cuantificados, respecto al algoritmo SP sin cuantificar para un PER de 10^{-5}. Los factores de corrección de los algoritmos αMS y βMS son aquellos con los que se consiguen las mejores prestaciones para cada código.

Las pérdidas debidas a la cuantificación son mayores para el código irregular y están en torno a los 0.2 y 0.3 dB para los esquemas de referencia $[6:2]$ y $[5:1]$, respectivamente. Con el código regular las pérdidas son de aproximadamente 0.1 dB para el primer esquema

Tabla 3.1: Perdidas de prestaciones de los algoritmos SP y basados en MS debidas a la cuantificación para un PER de 10^{-5} y 30 iteraciones.

Código LDPC	**Algoritmo**	**Esquema**		
		sin cuantificar	$[6:2]$	$[5:1]$
IEEE 802.3an	SP	-	0.18 dB	0.26 dB
RS-regular	MS	0.56 dB	0.66 dB	0.73 dB
($N = 2048$, $R = 1723/2048$)	αMS ($\alpha = 0.75$)	0.06 dB	0.18 dB	0.26 dB
	βMS ($\beta = 0.75$)	0.02 dB	0.15 dB	0.25 dB
IEEE 802.16e	SP	-	0.20 dB	0.31 dB
IRA-irregular	MS	0.2 dB	0.41 dB	0.50 dB
($N = 2304$, $R = 5/6$)	αMS ($\alpha = 0.875$)	0.05 dB	0.17 dB	0.28 dB
	βMS ($\beta = 1$)	0.05 dB	0.19 dB	0.29 dB

y 0.2 para el segundo. En general, se observa que la cuantificación afecta en mayor medida al algoritmo SP, en comparación con los algoritmos basados en MS. Por ejemplo, para el código regular, el algoritmo βMS pasa a ser la referencia, y para el código irregular, lo es el algoritmo αMS.

3.6. Conclusiones

Los algoritmos del tipo de intercambio de mensajes son los más utilizados para realizar la decodificación de códigos LDPC. Los más destacados son el algoritmo SP y el algoritmo MS. Este último es una simplificación del algoritmo SP que reduce la complejidad computacional pero sus prestaciones en términos de capacidad correctora son peores. Sin embargo, en [20] se proponen dos mejoras al algoritmo MS que reducen las pérdidas a costa de un bajo incremento en la complejidad. A manera de referencia se ha presentado el algoritmo mMS, en el cual se elimina la marginalización en el cálculo de los mensajes que envían los nodos de bit con el fin de reducir la complejidad computacional, pero como se ha demostrado, sus prestaciones en punto flotante son muy bajas para la mayoría de códigos LDPC y en precisión finita presenta problemas de convergencia.

El orden en el que se actualizan los nodos de comprobación y de bit modifica la velocidad de convergencia del algoritmo de decodificación. El método de actualización básico se conoce con el nombre de método por inundación y consiste en actualizar de manera secuencial los dos tipos de nodos: hasta que no se hayan actualizado todos los nodos de un tipo no se comienza con la actualización de los otros. En las actualizaciones *layered* y *shuffled* se realiza un paso de mensajes parciales entre grupos de nodos de ambos tipos, consiguiendo una mejora en la velocidad de convergencia. En general, con estos métodos de actualización se necesitan la mitad de iteraciones para conseguir las mismas prestaciones que con el método de actualización por inundación.

Se han comparado las prestaciones de los algoritmos SP, MS, αMS y βMS en punto flotante y punto fijo. Se ha observado que el algoritmo SP es el que mejores prestaciones obtiene cuando no está cuantificado, sin embargo, es al que más afecta la cuantificación. Para el análisis de precisión finita se han usado dos esquemas de cuantificación, $[6:2]$ y $[5:1]$. Estos esquemas se han elegido porque son los que mejor relación entre las prestaciones y número de bits tienen para los códigos con los que se ha evaluado las prestaciones. Los resultados muestran que los algoritmos αMS y βMS son los que mejores prestaciones consiguen con precisión finita, y por lo tanto, los que mejor relación entre complejidad y prestaciones presentan.

Capítulo 4

ALGORITMOS Y MÉTODOS DE ACTUALIZACIÓN PROPUESTOS

En este capitulo se presentan los algoritmos y métodos de actualización propuestos en esta tesis. Tanto los algoritmos como los métodos de actualización tienen como finalidad obtener unas prestaciones óptimas cuando son implementados en *hardware*. Entre las prestaciones que se busca mejorar están la complejidad *hardware*, la eficiencia de utilización *hardware* (HUE) y la tasa de decodificación (*throughput*). Se han propuesto dos algoritmos que reducen la complejidad *hardware* y un método de actualización que además de incrementar la HUE mejora la convergencia de los algoritmos permitiendo aumentar la tasa de decodificación. Los detalles de los algoritmos y el método de actualización propuesto se presentan a continuación.

4.1. Algoritmo Min-Sum entero

La mayoría de aplicaciones en las que se utilizan códigos LDPC son de alta velocidad por lo que se requieren implementaciones *hardware* con un alto grado de paralelismo. Esta paralelización provoca que existan un gran número de elementos que computan las actualizaciones de los nodos de comprobación y de bit y además, un elevado número de interconexiones (cables) y de elementos de almacenamiento para los diferentes mensajes. Por lo tanto, una reducción en el ancho de palabra de los mensajes permitiría reducir el área de los elementos de computo así como la de los elementos de almacenamiento y además disminuiría la congestión en las interconexiones. La finalidad del algoritmo que se propone a continuación es la de reducir al máximo los tamaños de los mensajes y conseguir que las pérdidas de prestaciones respecto al algoritmo SP sean bajas.

El algoritmo Min-Sum entero (iMS) es un algoritmo basado en MS y surge a partir del algoritmo MS con *offset*. Primero se define un factor de corrección β variable como se muestra en (4.1), donde el operador $\text{frac}(x)$ denota la parte fraccional de x.

$$\beta_{m,n}^{(i)} = \text{frac}(\min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)) \tag{4.1}$$

En la actualización de los nodos de comprobación del algoritmo βMS el mensaje a enviar se calcula restando el factor de corrección al mensaje generado por el nodo de comprobación y luego el resultado se compara con cero, seleccionando el mayor de ambos, como se muestra en (3.8). En otras palabras, cuando el factor de corrección es mayor que el mensaje el resultado es cero. Debido a que el resultado del operador $\text{frac}(x)$ es siempre menor o igual a x, el factor de corrección es siempre menor o igual al mensaje, por lo tanto, no es necesaria la comparación con cero. La actualización de los nodos de comprobación con esta simplificación se muestra en (4.2).

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \left(\min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|) - \beta_{m,n}^{(i)} \right) \tag{4.2}$$

El operador $\lfloor x \rfloor$ denota la parte entera de x la cual se define como $x - \text{frac}(x)$ para valores de $x \geq 0$. Por consiguiente, la actualización de los nodos de comprobación mostrada en (4.2) se puede escribir en términos del operador $\lfloor x \rfloor$, como se muestra en (4.3).

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \left\lfloor \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)) \right\rfloor \tag{4.3}$$

Dado que la parte entera del mínimo es igual al mínimo de las partes enteras:

$$\left\lfloor \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)) \right\rfloor = \min_{n' \in N_{m\setminus n}} \left(\lfloor |\lambda_{n',m}^{(i-1)}| \rfloor \right), \tag{4.4}$$

la actualización de los nodos de comprobación del algoritmo iMS finalmente queda como se muestra en (4.5). La complejidad computacional del algoritmo iMS parece un poco más alta que la del algoritmo MS debido a que se realiza el cálculo de la parte entera. Sin embargo, la implementación en *hardware* de esta operación no conlleva un coste adicional y además hace que los anchos de palabra sean menores, por lo que el coste de almacenamiento de los mensajes y el coste *hardware* de los operadores del algoritmo iMS es menor. Por lo tanto la complejidad *hardware* en general es menor, tal y como se demostrará en la Sección 5.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\setminus n}} \left(\lfloor |\lambda_{n',m}^{(i-1)}| \rfloor \right) \tag{4.5}$$

Aunque el algoritmo iMS surge como una modificación del algoritmo βMS, se puede aplicar el factor de escalado de la misma manera que con el algoritmo MS. El algoritmo iMS con escalado se denomina αiMS y su actualización de los nodos de comprobación se muestra en la ecuación 4.6. Debido a la reducción de los anchos de palabra la complejidad *hardware* de este algoritmo es menor a la de los algoritmo αMS.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \alpha \cdot \min_{n' \in N_{m\setminus n}} \left(\lfloor |\lambda_{n',m}^{(i-1)}| \rfloor \right) \tag{4.6}$$

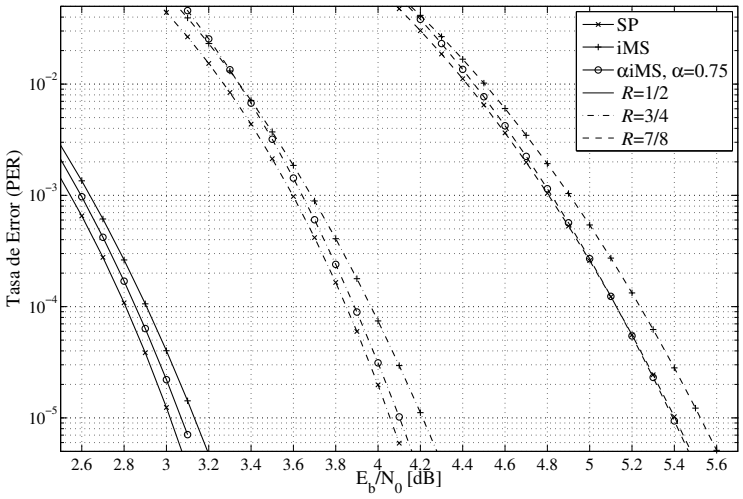


Figura 4.1: Tasa de error de los algoritmos SP y basados en iMS para los códigos del estándar IEEE 802.15.3c de longitud $N = 672$ y tasas $R = 1/2$, $3/4$ y $7/8$ con 30 iteraciones.

4.1.1. Análisis de prestaciones

El análisis de prestaciones de los algoritmos iMS y αiMS se ha realizado por simulación, de la misma forma que en la Sección 3.4. Los algoritmos se han evaluado con el método de actualización por inundación para los códigos de los estándares IEEE 802.15.3c, IEEE 802.16e e IEEE 802.3an. El factor de corrección que se ha usado para el algoritmos αiMS es 0.75. El número de iteraciones en todos los casos es igual a 30.

Las curvas de tasa de error de los algoritmos basados en iMS para los códigos del estándar IEEE 802.15.3c se muestran en la Figura 4.1. En la figura se han incluido las curvas del algoritmo SP a manera de referencia. Se observa que para un PER de 10^{-5} las pérdidas de prestaciones del algoritmo iMS respecto al algoritmo SP no son mayores a 0.15 dB y que las pérdidas del algoritmo αiMS son muy bajas, menores a 0.05 dB para todos los códigos. Comparando estos resultados con los obtenidos en la Sección 3.4 se observa que las prestaciones del algoritmo αiMS son tan buenas como las de los algoritmos αMS y βMS, y que las pérdidas de prestaciones del algoritmo iMS son solamente de 0.08 dB.

Las prestaciones de los algoritmos iMS y αiMS para los códigos del estándar IEEE 802.16e se muestran en la Figura 4.2. Se observa que las pérdidas de prestaciones del algoritmo iMS para un PER de 10^{-5} están en torno a los 0.06 dB, comparables a las obtenidas con los algoritmos αMS y βMS en la Sección 3.4. Las prestaciones del algoritmo αiMS para las tasas $1/2$, $2/3$ y $3/4$ son peores que las del algoritmo iMS. Sin embargo, para el código de tasa $5/6$ las prestaciones del algoritmo αiMS mejoran las del iMS, llegando a ser comparables a las del algoritmo SP.

En la Figura 4.3 se muestran las curvas de tasa de error de los algoritmos SP y basados en iMS para el código del estándar IEEE 802.3an. Para un PER de 10^{-5} el algoritmo αiMS tiene prácticamente las mismas prestaciones que el algoritmo SP, mientras que el algoritmos iMS presenta unas pérdidas de 0.16 dB.

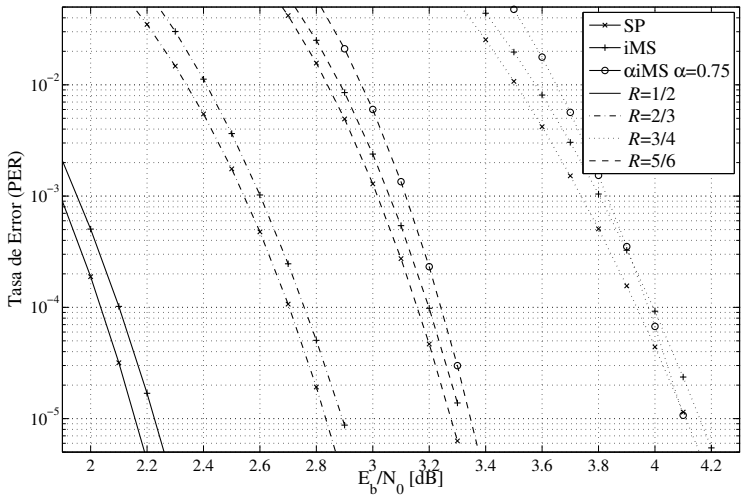


Figura 4.2: Tasa de error de los algoritmos SP y basados en iMS para los códigos del estándar IEEE 802.16e de longitud $N = 2304$ y tasas $R = 1/2$, $2/3$, $3/4$ y $5/6$ con 30 iteraciones.

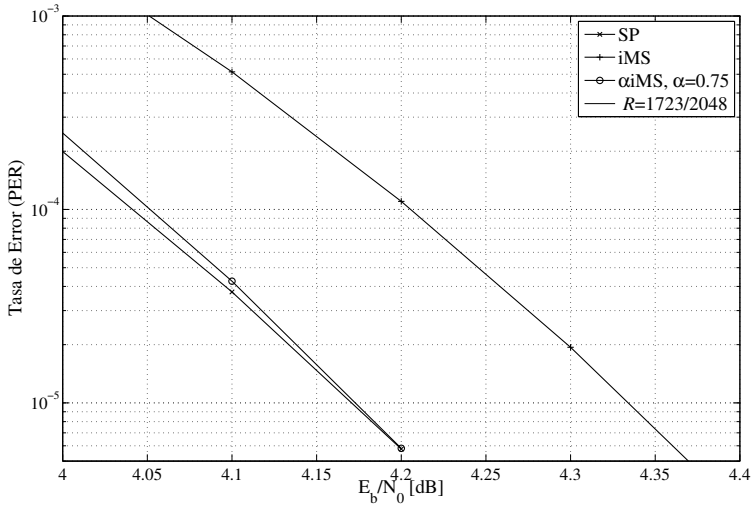


Figura 4.3: Tasa de error de los algoritmos SP y basados en iMS para el códigos del estándar IEEE 802.3an de longitud $N = 2048$ y tasa $R = 1723/2048$ con 30 iteraciones.

Tabla 4.1: Perdidas de prestaciones de los algoritmos SP y basados en MS debidas a la cuantificación para un PER de 10^{-5} y 30 iteraciones.

Código LDPC	**Algoritmo**	**Esquema**		
		sin cuantificar	$[6:2]$	$[5:1]$
IEEE 802.3an	SP	-	0.18 dB	0.26 dB
RS-regular	αMS ($\alpha = 0.75$)	0.06 dB	0.18 dB	0.26 dB
($N = 2048$, $R = 1723/2048$)	iMS	0.16 dB	0.24 dB	0.37 dB
	αiMS ($\alpha = 0.75$)	0.01 dB	0.16 dB	0.23 dB
IEEE 802.16e	SP	-	0.20 dB	0.31 dB
IRA-irregular	αMS ($\alpha = 0.875$)	0.05 dB	0.17 dB	0.28 dB
($N = 2304$, $R = 5/6$)	iMS	0.05 dB	0.16 dB	0.27 dB
	αiMS ($\alpha = 0.75$)	0.01 dB	0.15 dB	0.28 dB

4.1.2. Análisis de precisión finita

La precisión finita de los algoritmos iMS y αiMS se ha evaluado para el código LDPC del estándar IEEE 802.3an y para el código de longitud 2304 y tasa $5/6$ el estándar IEEE 802.16e. Al igual que para el análisis de precisión finita de los algoritmos basados en MS se ha tomado como referencia los esquemas de cuantificación $[6:2]$ y $[5:1]$. Por lo tanto, los mensajes generados por los nodos de bit se han cuantificado con los esquemas $[6:2]$ y $[5:1]$ y los generados por los nodos de comprobación con el esquema $[4:0]$ para ambos casos.

Las pérdidas de prestaciones de los algoritmos iMS y αiMS debidas a la cuantificación se muestran en la Tabla 4.1. En ésta también se muestran las pérdidas de los algoritmos SP y αMS a manera de comparación. Los resultados se han evaluado para un PER de 10^{-5} y están referidos a las prestaciones del algoritmo SP sin cuantificar. Se observa que el efecto de la cuantificación sobre las prestaciones del algoritmo iMS es de aproximadamente 0.1 y 0.2 dB para los esquemas de referencia $[6:2]$ y $[5:1]$, respectivamente. Para el código del estándar IEEE 802.16e las prestaciones de éste son mejores que las del algoritmo αMS. Destacan las buenas prestaciones del algoritmo αiMS con ambos códigos, superando al algoritmo αMS por 0.02 dB con el esquema $[6:2]$. Sin embargo, hay que recordar que las prestaciones del algoritmo αiMS para el resto de tasas del código IRA-irregular no son muy buenas (Figura 4.2).

4.2. Algoritmo Min-Sum modificado con corrección

En los decodificadores de alta velocidad el número de elementos que computan los mensajes de los nodos de comprobación es elevado y en general comprenden más del 60 % de la complejidad *hardware* total. Por lo tanto, una reducción en la complejidad del cálculo de estos mensajes, como la que se obtiene con el algoritmo mMS (Sección 3.2.2), tendría un alto impacto en el área de los elementos de cómputo, y por consiguiente, en el área total del decodificador. Sin embargo, se ha demostrado que las prestaciones del algoritmo mMS son muy bajas en general debido a la ausencia de marginalización, la cual consiste en enviar únicamente la información extrínseca. Esto se traduce en que el cálculo del mensaje $\mu_{m,n}$ se realiza sin excluir el elemento n del conjunto N_m. En [23] y [24] se propone un algoritmo denominado *single minimum* MS (smMS) que aprovecha la baja complejidad que conlleva

el cálculo de un único mínimo en la operación de actualización del nodo de comprobación y añade operaciones extra con el fin de compensar las pérdidas de prestaciones.

En el algoritmo smMS se suma un factor de corrección w al mínimo con el fin de obtener un "segundo" mínimo y así emular el cálculo de los dos mínimos que se realiza cuando hay marginalización y además, se añade un factor de escalado α de manera similar al algoritmo αMS. La actualización de los nodos de comprobación para el algoritmo smMS se muestra en (4.7) donde $\upsilon_m^{(i)}$ es el valor mínimo y $c_m^{(i)}$ es el número de mensajes $\lambda_{n',m}^{(i-1)}$ cuya magnitud es igual a ese mínimo.

$$\begin{aligned} \upsilon_m^{(i)} &= \min_{n' \in N_m} (|\lambda_{n',m}^{(i-1)}|) \\ \mu_{m,n}^{(i)} &= \Gamma_{m,n}^{(i)} \cdot \alpha \cdot \begin{cases} \upsilon_m^{(i)} + w & \upsilon_m^{(i)} = |\lambda_{n,m}^{(i-1)}| \text{ y } c_m^{(i)} = 1 \\ \upsilon_m^{(i)} & \text{en otro caso} \end{cases} \end{aligned} \tag{4.7}$$

El algoritmo smMS presenta menor complejidad que el algoritmo MS y para el código del estándar IEEE 802.3an mejores prestaciones. Sin embargo, las pérdidas respecto al algoritmo SP para un PER de 10^{-5} son mayores a 0.3 dB y para varios códigos irregulares de los estándares IEEE 802.15.3c e IEEE 802.16e el algoritmo presenta problemas de convergencia que se reflejan en la aparición de un suelo de error en las curvas de tasa de error en niveles bajos de PER.

Con el fin de mejorar la convergencia del algoritmo para los códigos irregulares y mejorar las prestaciones para el código del estándar IEEE 802.3an se propone una variación del algoritmo smMS en la que el factor de corrección se hace variable y dependiente de la iteración. Esta modificación se plantea tras el análisis realizado al comportamiento de los valores mínimos en el que se observó que su valor aumenta a medida que se incrementan las iteraciones, por lo tanto, el factor de corrección debe aumentar para mantener la proporción entre el primer y segundo mínimo. Este nuevo algoritmo se denomina vwMS y su actualización de los nodos de comprobación se muestra en (4.8).

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \alpha \cdot \begin{cases} \upsilon_m^{(i)} + w^{(i)} & \upsilon_m^{(i)} = |\lambda_{n,m}^{(i-1)}| \text{ y } c_m^{(i)} = 1 \\ \upsilon_m^{(i)} & \text{en otro caso} \end{cases} \tag{4.8}$$

En las implementaciones *hardware* de muy alta velocidad se requiere un alto grado de paralelización en las operaciones de decodificación, por lo tanto, el evaluar de manera exhaustiva si hay más de un mensaje con valor mínimo tiene un alto impacto sobre la complejidad. Con el fin de reducir esta complejidad se dividen los mensajes en dos grupos, pares e impares, y se calcula el mínimo de cada uno como se muestra en (4.9). El valor mínimo total se obtiene seleccionando el mínimo entre los pares e impares (ecuación 4.10). Para determinar si existe más de un mensaje con valor mínimo se comparan los dos valores mínimos obtenidos anteriormente, de manera que, si son iguales existe más de un mensaje con valor mínimo y si son diferentes se asume que solamente existe uno. La actualización de los nodos de comprobación con esta aproximación se muestra en (4.11). De esta manera se obtiene un error respecto al número real de mensajes iguales al mínimo, sin embargo, su coste es bajo y según las pruebas realizadas no afecta a las prestaciones del algoritmo.

$$\upsilon_{m\,\text{par}}^{(i)} = \min_{\substack{n'\in N_m \\ n'\ \text{par}}} (|\lambda_{n',m}^{(i-1)}|),\ \upsilon_{m\,\text{impar}}^{(i)} = \min_{\substack{n'\in N_m \\ n'\ \text{impar}}} (|\lambda_{n',m}^{(i-1)}|) \tag{4.9}$$

$$\upsilon_m^{(i)} = \min(\upsilon_{m\,\text{par}}^{(i)}, \upsilon_{m\,\text{impar}}^{(i)}) \tag{4.10}$$

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \alpha \cdot \begin{cases} \upsilon_m^{(i)} + w^{(i)} & \upsilon_m^{(i)} = |\lambda_{n,m}^{(i-1)}| \text{ y } \upsilon_{m\,\text{par}}^{(i)} \neq \upsilon_{m\,\text{impar}}^{(i)} \\ \upsilon_m^{(i)} & \text{en otro caso} \end{cases} \tag{4.11}$$

La complejidad del algoritmo vwMS respecto al algoritmo mMS es más alta debido a la aplicación de los factores de corrección y escalado y a las operaciones adicionales que se realizan para determinar si existe más de un mensaje cuyo valor sea igual al mínimo. Sin embargo, sus prestaciones son mejores y comparado con el resto de algoritmos basados en MS presenta menor complejidad debido principalmente a la diferencia que existe entre el cálculo de uno o dos mínimos. El impacto que tiene este algoritmo en la implementación *hardware* del decodificador se analiza con más detalle en el Capítulo 5.

4.2.1. Análisis de prestaciones

Las prestaciones del algoritmo vwMS se han evaluado mediante simulación para los códigos de los estándares IEEE 802.15.3c, IEEE 802.16e e IEEE 802.3an. El número de iteraciones utilizado es 30, al igual que en la evaluación de prestaciones de los algoritmos SP y MS. Además, se han evaluado las prestaciones del algoritmo smMS a manera de referencia. Los factores de escalado α que se han utilizado en los algoritmos son 0.5, 0.75 y 0.875. El algoritmo smMS se han evaluado con valores de corrección w que van desde 0.25 a 4 en pasos de 0.25. Para el algoritmo vwMS los factores de corrección variables se definen como vectores de cuatro elementos $w = [w_0; w_1; w_2; w_3]$ donde cada elemento puede ser un valor comprendido entre 0.25 y 4 con pasos de 0.25. Los umbrales de aplicación de los factores de corrección se definen mediante un vector de tres elementos $u = [u_0; u_1; u_2]$ donde cada elemento u_n identifica la última iteración en la que se aplica el factor de corrección w_n. Por ejemplo, si $w = [1; 1.5; 2.75; 3.75]$ y $u = [4; 8; 12]$ el factor de corrección para las iteraciones 1 a 4 es 1, para las iteraciones 5 a 8 es 1.5, para las iteraciones 9 a 12 es 2.75 y para las iteraciones mayores a 12 es 3.75. Aunque en las gráficas se muestran las curvas con las combinaciones de factores que mejores prestaciones obtienen para cada código, es importante hacer notar la gran cantidad de combinaciones entre factores de corrección, de escalado y umbrales que existe y por lo tanto la gran cantidad de simulaciones que se han realizado para obtener los resultados.

Las Figuras 4.4 a 4.6 muestran las curvas de tasa de error de los algoritmos vwMS y smMS para los códigos del estándar IEEE 802.15.3c. A manera de referencia se añaden las curvas de error de los algoritmos SP y MS. Se observa que para las tasas $1/2$ y $3/4$ aparece un suelo de error en las curvas del algoritmo smMS y que para la tasa $7/8$ las mejora en las prestaciones respecto al algoritmo MS es de apenas 0.03 dB para un PER de 10^{-5}. Por otro lado, el algoritmo vwMS mejora las prestaciones del MS para todas las tasas y comparado con el SP presenta pérdidas de 0.28, 0.22 y 0.09 dB para las tasas $1/2$, $3/4$ y $7/8$, respectivamente. En general, el algoritmo vwMS no presenta problemas de convergencia para tasas de error mayores a 10^{-5} y sus prestaciones son mejores que las del algoritmo MS.

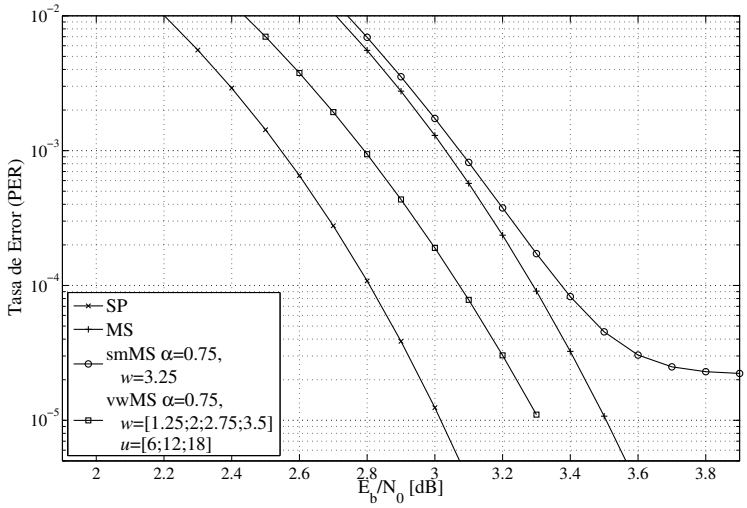


Figura 4.4: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.15.3c de longitud $N = 672$ y tasa $R = 1/2$ con 30 iteraciones.

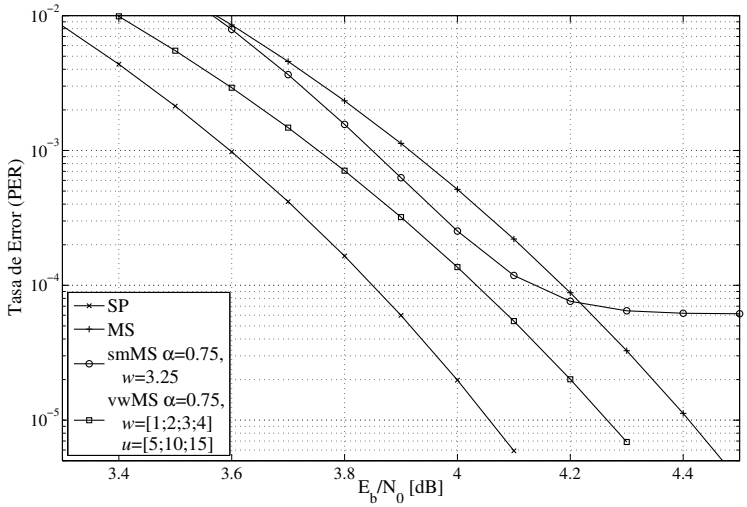


Figura 4.5: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.15.3c de longitud $N = 672$ y tasa $R = 3/4$ con 30 iteraciones.

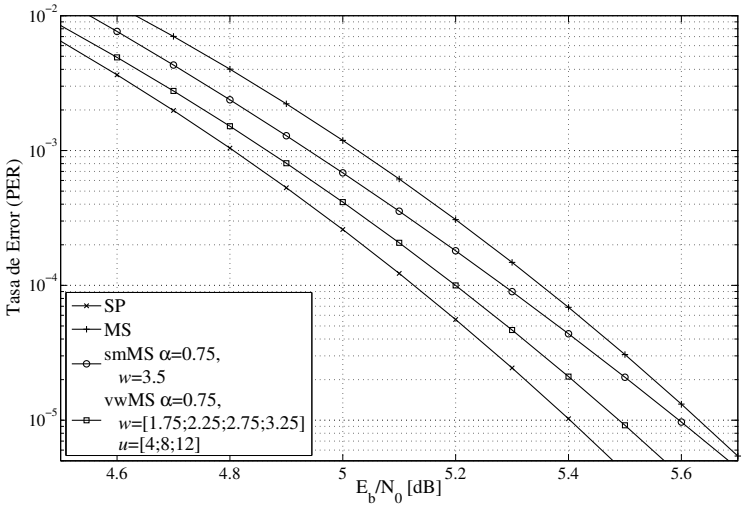


Figura 4.6: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.15.3c de longitud $N = 672$ y tasa $R = 7/8$ con 30 iteraciones.

Las prestaciones de los algoritmos vwMS y smMS para los códigos de longitud 2304 del estándar IEEE 802.16e se muestran en las Figuras 4.7 a 4.10. Para los códigos de tasas $1/2$, $2/3$ y $3/4$ el algoritmo smMS presenta una alta degradación en las prestaciones y para el código de tasa $5/6$ unas pérdidas superiores a 0.1 dB respecto al algoritmo MS para un PER de 10^{-5}. Se observa también que las pérdidas del algoritmo vwMS respecto al MS para las tasas $1/2$ y $2/3$ son inferiores a 0.1 dB y que para las tasas $3/4$ y $5/6$ obtiene mejores prestaciones. Por otro lado, las pérdidas de prestaciones del algoritmo vwMS respecto al algoritmo SP son mayores a 0.2 dB.

Las curvas de tasa de error de los algoritmos vwMS, smMS, MS y SP para el código del estándar IEEE 802.3an se muestran en la Figura 4.11. Se observa que para un PER de 10^{-5} las prestaciones del algoritmo vwMS son muy similares a las del algoritmo SP con apenas 0.03 dB de pérdidas y que la mejora de las prestaciones respecto al algoritmo MS es mayor a 0.5 dB. Para este código el algoritmo smMS mejoras las prestaciones del MS en más de 0.22 dB, sin embargo, las pérdidas respecto al SP son de 0.28 dB. Cabe destacar que el factor de escalado en ambos algoritmos es igual a 0.5. Por lo tanto, en una implementación no se requiere *hardware* adicional para realizar el escalado de los mensajes generados por los nodos de comprobación.

4.2.2. Análisis de precisión finita

La evaluación de la precisión finita para el algoritmo vwMS se ha realizado mediante simulación para el código del estándar IEEE 802.3an y para el código de longitud 2304 y tasa $5/6$ del estándar IEEE 802.16e. Los factores de corrección y escalado son aquellos con los que mejores prestaciones se obtienen. Para el código del estándar IEEE 802.16e el factor α es 0.75, el factor w es $[1;2;3;4]$ y el vector $u = [3;6;9]$. Para el código del estándar IEEE

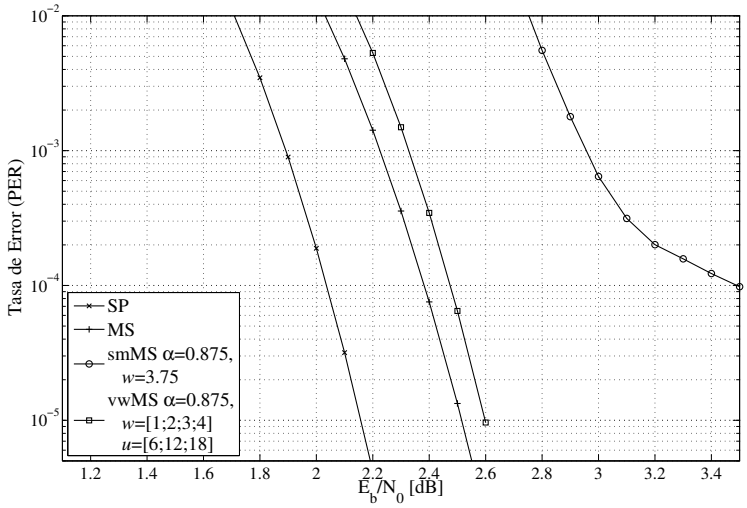


Figura 4.7: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 1/2$ con 30 iteraciones.

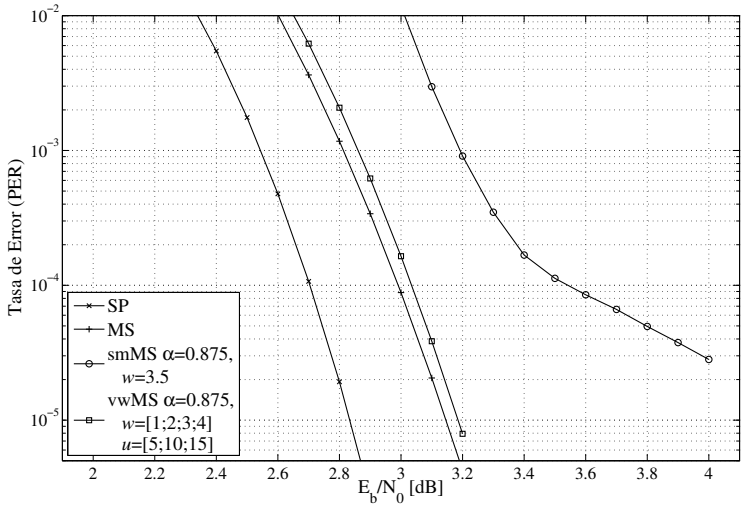


Figura 4.8: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 2/3$ con 30 iteraciones.

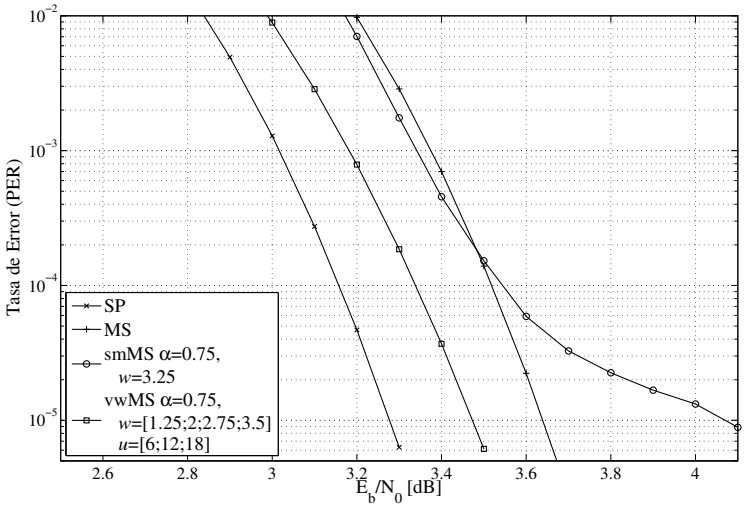


Figura 4.9: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 3/4$ con 30 iteraciones.

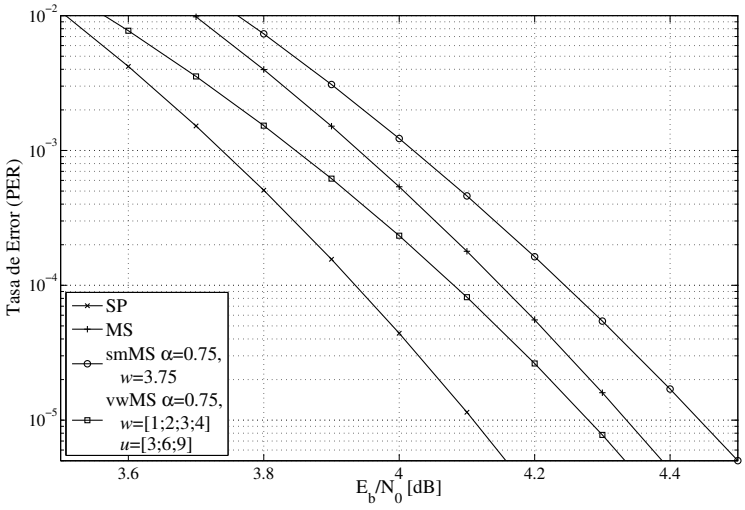


Figura 4.10: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.16e de longitud $N = 2304$ y tasa $R = 5/6$ con 30 iteraciones.

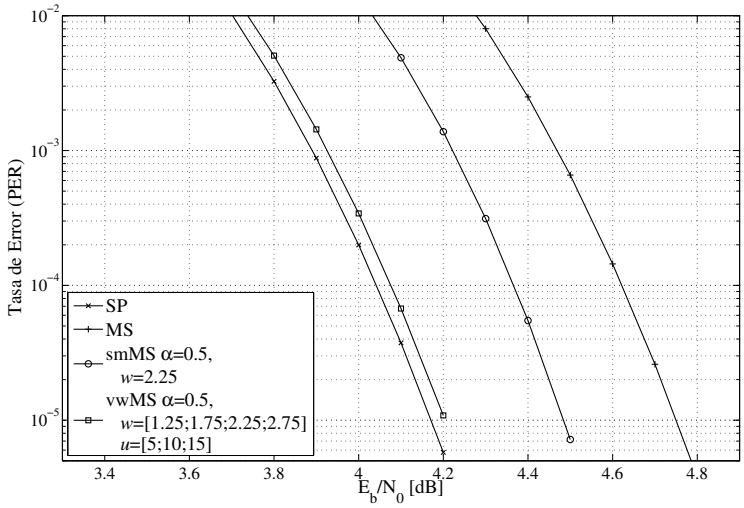


Figura 4.11: Tasa de error de los algoritmos SP, MS, smMS y vwMS para el código del estándar IEEE 802.3an de longitud $N = 2048$ y tasa $R = 1723/2048$ con 30 iteraciones.

802.3an el factor α es 0.5, $w = [1.25; 1.75; 2.25; 2.75]$ y $u = [5; 10; 15]$. La cuantificación es igual tanto para los mensajes de los nodos de bit como para los de comprobación, donde se aplican los esquemas $[6:2]$ y $[5:1]$, al igual que en la Sección 3.4.

La Tabla 4.2 muestra las pérdidas del algoritmo vwMS debidas a la cuantificación para un PER de 10^{-5} con los esquemas $[6:2]$ y $[5:1]$. Los resultados están referidos a las prestaciones del algoritmo SP sin cuantificar y a manera de referencia se incluyen los resultados de los algoritmos MS y αMS. Al igual que con los algoritmos basados en MS de la Sección 3.4, las pérdidas de cuantificación del algoritmo vwMS para el esquema $[6:2]$ son del orden de 0.1 dB con ambos códigos. Sin embargo, con el esquema $[5:1]$ son del orden de 0.3 dB, 0.2 dB más que para el resto de algoritmos basados en MS. Esto se debe a que la resolución de los factores de corrección es de 2 bits fraccionales, y al aplicar un esquema de cuantificación $[5:1]$ se pierde un bit. Por otro lado, se observa que las prestaciones del algoritmo vwMS cuantificado con esquema $[6:2]$ son algo mejores que las del algoritmo αMS con el código del estándar IEEE 802.3an. Para el código del estándar IEEE 802.16e las prestaciones con el esquema $[6:2]$ son 0.12 dB mejores que las del algoritmo MS y 0.12 dB peores que las del algoritmo αMS.

4.3. Actualización *shuffled* entrelazada (*x-shuffled*)

Los métodos de actualización por capas como el *shuffled* o el *layered* aceleran la convergencia de los algoritmos, lo que implica un menor número de iteraciones para alcanzar unas prestaciones dadas. Sin embargo, las implementaciones *hardware* del método *shuffled* suelen tener baja eficiencia de utilización *hardware* (HUE) debido a las dos fases que se realizan en cada subiteración: es necesario actualizar todos los nodos de bit antes de actualizar el g-ésimo grupo de nodos de comprobación y viceversa. Por tanto, mientras se

Tabla 4.2: Perdidas de prestaciones de los algoritmos MS, αMS y vwMS respecto al SP debidas a la cuantificación para un PER de 10^{-5}.

Código LDPC	**Algoritmo**	**Esquema**		
		sin cuantificar	[6 : 2]	[5 : 1]
IEEE 802.3an	SP	-	0.18 dB	0.26 dB
RS-regular	αMS ($\alpha = 0.75$)	0.06 dB	0.18 dB	0.26 dB
($N = 2048$, $R = 1723/2048$)	MS	0.56 dB	0.66 dB	0.73 dB
	vwMS ($\alpha = 0.5$, $w = [1.25; 1.75; 2.25; 2.75]$, $u = [5; 10; 15]$)	0.03 dB	0.15 dB	0.31 dB
IEEE 802.16e	SP	-	0.20 dB	0.31 dB
IRA-irregular	αMS ($\alpha = 0.875$)	0.05 dB	0.17 dB	0.28 dB
($N = 2304$, $R = 5/6$)	MS	0.2 dB	0.41 dB	0.50 dB
	vwMS ($\alpha = 0.75$, $w = [1; 2; 3; 4]$, $u = [3; 6; 9]$)	0.17 dB	0.29 dB	0.38 dB

actualizan los nodos de bit los elementos *hardware* que computan los nodos de comprobación estarán en estado de espera y lo mismo ocurre en caso contrario. Además de la baja HUE la tasa de decodificación también se ve afectada por los ciclos de espera causados por la dependencia de datos en las subiteraciones. Con el método de actualización propuesto se pretende eliminar la dependencia entre datos dentro de una misma subiteración y que el aumento en la velocidad de convergencia sea similar al del método *shuffled*. Cabe destacar que en la actualización *layered* también hay dependencia de datos en las subiteraciones, pero las arquitecturas que normalmente se usan para las implementaciones *hardware* juntan la computación de ambos nodos comprobación en una única celda (se computan las dos fases en un ciclo de reloj), minimizando así el efecto que tiene la dependencia en la HUE.

El método de actualización *x-shuffled* se basa en el intercambio de mensajes parciales entre los nodos de comprobación y grupos de nodos de bit al igual que el método de actualización *shuffled* (Sección 3.3.3). De la misma manera que con la actualización *shuffled*, se dividen los N nodos de bit en G grupos donde G debe ser un número entero tal que el resto de la división $N_G = N/G$ sea cero, siendo N_G el número de nodos de bit por grupo. A diferencia del método *shuffled*, en el que la actualización en cada subiteración se hace de forma secuencial (primero todos los nodos de comprobación y luego el g-ésimo grupo de nodos de bit), los nodos de comprobación y el g-ésimo grupo de nodos de bit se actualizan al mismo tiempo usando los mensajes actualizados en la subiteración anterior. La representación gráfica del método de actualización *x-shuffled* con $G = 3$ y para una matriz $\mathbf{H}$ de 6 filas y 12 columnas se muestra en la Figura 4.12, donde las lineas discontinuas verticales representan la actualización de los nodos de bit y las horizontales la actualización de los nodos de comprobación.

En el pseudocódigo 4.1 se describe el método de actualización *x-shuffled* aplicado al algoritmo MS donde el término $\Gamma_{m,n}^{(i)}$ que representa la actualización de paridad se define como:

$$\Gamma_{m,n}^{(i)} = \prod_{\substack{n' \in N_{m\setminus n} \\ n' > (g-1)\cdot N_G}} \operatorname{sign}(\lambda_{n',m}^{(i-1)}) \cdot \prod_{\substack{n' \in N_{m\setminus n} \\ n' \leq (g-1)\cdot N_G}} \operatorname{sign}(\lambda_{n',m}^{(i)}). \tag{4.12}$$

En las líneas 10 y 13 del pseudocódigo 4.1 se puede ver que el cálculo del mensaje

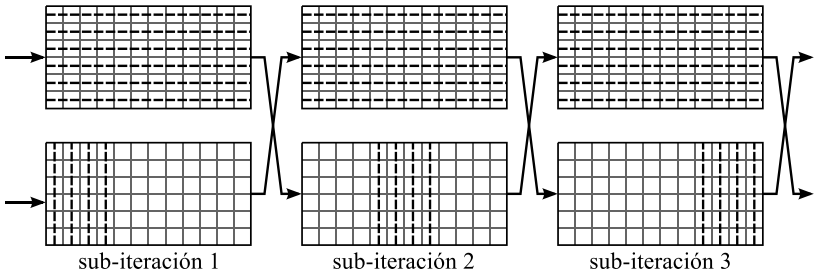


Figura 4.12: Método de actualización *x-shuffled* con $G = 3$ para una matriz $\mathbf{H}$ de tamaño 6×12, donde las lineas discontinuas verticales representan la actualización de los nodos de bit y las horizontales la actualización de los nodos de comprobación.

Pseudocódigo 4.1 Algoritmo MS con actualización *x-shuffled*.

– *inicialización* –
para $n \in \{1,\ldots,N\}$ y $m \in M_n$
  $\lambda_{n,m}^{(0)} = l_n$
– *iteraciones* –
para $i \in \{1,\ldots,I_{max}\}$
  – *subiteraciones* –
  para $g \in \{1,\ldots,G\}$
    – *actualización de los nodos de comprobación* –
    para $m \in \{1,\ldots,M\}$ y $n \in N_m$

$$\mu_{m,n}^{(i,g)} = \Gamma_{m,n}^{(i)} \cdot \min\Big(\min_{\substack{n' \in N_{m\setminus n} \\ n' > (g-1)\cdot N_G}} (|\lambda_{n',m}^{(i-1)}|), \min_{\substack{n' \in N_{m\setminus n} \\ n' \leq (g-1)\cdot N_G}} (|\lambda_{n',m}^{(i)}|) \Big)$$

    – *actualización de los nodos de bit (grupo)* –
    para $n \in \{(g-1)\cdot N_G + 1,\ldots,g\cdot N_G\}$ y $m \in M_n$

$$\lambda_n^{(i)} = l_n + \sum_{\substack{m' \in M_n \\ g>1}} \mu_{m',n}^{(i,g-1)} + \sum_{\substack{m' \in M_n \\ g=1}} \mu_{m',n}^{(i-1,G)}$$

$$\lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \begin{cases} \mu_{m,n}^{(i-1,G)} & g = 1 \\ \mu_{m,n}^{(i,g-1)} & g > 1 \end{cases}$$

– *decodificación hard* –
para $n \in \{1,\ldots,N\}$

$$z_n = \begin{cases} 1 & \lambda_n^{(I_{max})} \geq 0 \\ 0 & \lambda_n^{(I_{max})} < 0 \end{cases}$$

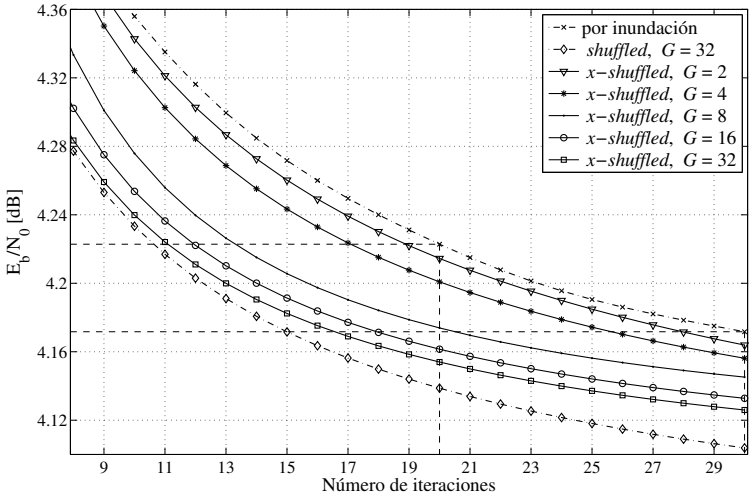


Figura 4.13: Convergencia del algoritmo SP con actualizaciones *shuffled* y *x-shuffled* para el código del estándar IEEE 802.3an con PER = 10^{-5}.

que envía cada nodo de comprobación $\mu_{m,n}$ y de bit $\lambda_{n,m}$ en cada subiteración g se realiza con los mensajes de las subiteraciones anteriores. Por ejemplo, en la primera subiteración de una iteración i la actualización de los nodos de comprobación se hace con los mensajes provenientes de los nodos de bit de la iteración anterior $i-1$ y la actualización de los nodos de bit se realiza con los mensajes $\mu_{m,n}$ de la subiteración G e iteración $i-1$. Por lo tanto, la dependencia que existía entre los mensajes generados por los nodos de comprobación con la actualización de los nodos de bit desaparece. La eliminación de esta dependencia tiene el efecto de eliminar un factor dos en el denominador de la ecuación que determina la tasa de decodificación, permitiendo doblar la velocidad, tal y como se muestra en el Capítulo 5 Sección 5.5.

4.3.1. Análisis de prestaciones

En la Figura 4.13 se muestran las curvas de convergencia del algoritmo SP con el método de actualización *x-shuffled* para un PER de 10^{-5}. A manera de referencia se han incluido las curvas con el método por inundación y con *shuffled* de 32 grupos. Se observa que las prestaciones del algoritmo con actualización por inundación para 20 iteraciones se consiguen con actualización *x-shuffled* de 32 grupos en 11, una más que con actualización *shuffled*. También se observa que la mejora de la convergencia con 8 y 16 grupos es alta, mientras que con 2 y 4 la convergencia es similar a la del método por inundación. En general, la convergencia de la actualización *x-shuffled* es 2 iteraciones mas lenta que la convergencia de la actualización *shuffled*.

4.4. Conclusiones

En este trabajo se han propuesto dos algoritmos y un método de actualización con la intención de mejorar las prestaciones de las implementaciones *hardware*. El algoritmo Min-Sum entero (iMS) y su versión escalada (αiMS) disminuyen la complejidad *hardware* debido a la reducción que generan en los anchos de palabra. Además presentan muy buenas prestaciones en términos de capacidad correctora tanto en punto flotante como en precisión finita, llegando a mejorar a los algoritmos αMS y βMS.

El otro algoritmo propuesto, denominado vwMS, es una modificación del algoritmo smMS ([24]) en el que se emula la marginalización sin tener que calcular dos mínimos. El hecho de no tener que calcular dos mínimos hace que la complejidad de este algoritmo sea baja, aunque mayor a la del mMS debido a la inclusión de factores de corrección variables y de un factor de escalado. Las prestaciones de este algoritmo en punto flotante son comparables a las del algoritmo MS para los códigos del los estándares IEEE 802.15.3c e IEEE 802.16e, al mismo tiempo, para el código del estándar IEEE 802.3an alcanza una prestaciones similares a las del algoritmo SP, con unas pérdidas menores a 0.03 dB. En precisión finita, con un esquema de cuantificación $[6:2]$, las prestaciones del algoritmo vwMS mejoran en 0.03 dB las de los algoritmos SP y αMS con el código del estándar IEEE 802.3an. Por contra, para el código de longitud $N = 2034$ y tasa $R = 5/6$ del estándar IEEE 802.16e este algoritmo presenta pérdidas, 0.09 y 0.12 dB respecto a los algoritmos SP y αMS, respectivamente.

Con el método de actualización *x-shuffled* se consigue eliminar la dependencia entre los mensajes que actualizan los nodos de comprobación y los nodos de bit de cada subiteración, permitiendo que las actualizaciones se hagan al mismo tiempo. Este método realiza un paso de mensajes parciales entre nodos de comprobación y grupos de nodos de bit consiguiendo una mejora en la velocidad de convergencia, comparable a la obtenida por los métodos de actualización *layered* y *shuffled*, que permite aumentar la tasa de decodificación. Además, se ha observado que el número de iteraciones que necesita la actualización *x-shuffled* es ligeramente mayor que con actualización *shuffled* para lograr las mismas prestaciones a una misma tasa de error.

Capítulo 5

ARQUITECTURAS E IMPLEMENTACIÓN HARDWARE

En este capítulo se presentan diferentes arquitecturas para la implementación *hardware* de decodificadores LDPC. Un decodificador LDPC se compone básicamente de dos tipos de unidades de procesado: las unidades de procesado del nodo de comprobación (CNP - *check-node processor*) que computan las operaciones de actualización de los nodos de comprobación (actualización horizontal) y las unidades de procesado del nodo variable (VNP - *variable-node processor*) que computan las operaciones de actualización de los nodos de bit (actualización vertical).

Las arquitecturas que se encuentran en la literatura se pueden clasificar, de acuerdo a la forma en la que se realizan las actualizaciones, en dos grandes grupos: paralelas [25, 26, 31–37] y parcialmente paralelas [29, 38–54]. En las arquitecturas paralelas los procesos de actualización horizontal y vertical se realizan de forma concurrente, por lo que son necesarias tantas unidades CNP como filas tiene la matriz de paridad y tantas unidades VNP como columnas. La gran cantidad de elementos de procesado que requiere esta arquitectura y la gran cantidad de conexiones hace que su complejidad sea elevada. Con las arquitecturas parcialmente paralelas, en las que se combina cierto grado de paralelismo con procesado secuencial, se consigue un buen compromiso entre tasa de decodificación y complejidad. En las arquitecturas con un grado bajo de paralelismo, en las que la mayor parte del procesado es secuencial, se pueden utilizar memorias para almacenar los mensajes generados por las unidades de procesado. Sin embargo, cuando el grado de paralelismo es alto el uso de memorias no es eficiente, debido a que es necesario almacenar una gran cantidad de mensajes a la vez. En este caso se usan registros para almacenar los mensajes generados por las unidades. Las arquitecturas que usan memorias para almacenar los mensajes se denominan basadas en memoria y las que usan registros se denominan basadas en registros.

En [31] se presenta la primera implementación de una arquitectura completamente paralela. La complejidad de esta implementación es alta y la tasa de decodificación que alcanza es baja debido a la alta congestión en el rutado que limita la frecuencia de funcionamiento del decodificador. En [33] se propone realizar la comunicación entre unidades de forma serie, disminuyendo el retardo de propagación debido a la congestión por el rutado, y por consiguiente mejorando la tasa de decodificación. Otra técnica para mejorar la tasa de

decodificación es el denominado *wire partitioning* propuesto en [36], el cual consiste en segmentar los caminos más largos para disminuir el camino crítico. Sin embargo, el incluir dicha segmentación supone una alteración del orden de actualización de los mensajes, lo que redunda en en el deterioro de las prestaciones. Aunque las implementaciones con la arquitectura paralela alcanzan tasas de decodificación muy altas (del orden de Gbps), su complejidad *hardware* es muy elevada y por tanto, el área requerida en estos casos es alta. Una modificación sobre la arquitectura paralela se presenta en [25] la cual se denomina *Split-Row*. En ésta se divide la matriz de paridad en bloques de columnas los cuales se procesan de forma independiente y en paralelo. Con este método es necesario modificar el algoritmo de decodificación e introduce pérdidas en las prestaciones, sin embargo, las tasas de decodificación alcanzadas son altas y el área se reduce considerablemente. Con el fin de mejorar las prestaciones, los mismos autores proponen varias modificaciones en [26, 32, 34, 35, 37]. En la arquitectura *Sliced Message Passing* - SMP, propuesta en [38], también se divide la matriz de paridad en bloques de columnas, pero a diferencia de la arquitectura *Split-Row*, en ésta el procesado se hace de forma secuencial por bloques (parcialmente paralelo). Este procesado permite solapar las fases de actualización de los mensajes sin modificar las prestaciones del algoritmo y por consiguiente, mejorar la tasa de decodificación. La arquitectura SMP se diseñó con un grado de paralelismo alto para conseguir altas tasas de decodificación, por lo que el almacenamiento de los mensajes durante el proceso de decodificación se hace mediante registros.

Se denomina comúnmente arquitectura parcialmente paralela basada en memorias a aquella que implementa el método de actualización por inundación. En este tipo de arquitectura el nivel de paralelismo es relativamente bajo, por lo que su complejidad *hardware* es también baja. Sin embargo, las implementaciones con ésta alcanzan tasas de decodificación altas, del orden de cientos de Mbps, consiguiendo un buen compromiso entra área y tasa de decodificación. Además, presenta una reducida densidad de conexiones, que junto a la baja complejidad, hace que se pueda implementar en dispositivos FPGA. En [39, 40] se presentan implementaciones en FPGA donde se optimiza el control, aprovechando la estructura de los códigos cuasi-cíclicos, y se utiliza cuantificación no uniforme para reducir el almacenamiento. En [41] se propone una arquitectura multimodo para los diferentes códigos contemplados en el estándar IEEE 802.16e. Una arquitectura similar es propuesta en [43], pero a diferencia de la anterior, en ésta se reordenan las matrices para optimizar el almacenamiento. Con el fin de mejorar la tasa de decodificación, en [42] se solapan las fases (actualización horizontal y vertical), incurriendo en pérdidas de prestaciones, sin embargo, se realiza una reordenación de la matriz de paridad para minimizar dichas pérdidas. En [44] se implementa un decodificador en un dispositivo FPGA para el estándar DVB-S2, en el que se reutiliza parte del hardware de las unidades CNP para realizar el cómputo de las unidades VNP. En [45] se paralelizan las unidades y se agrupan las señales en vectores para optimizar el almacenamiento en las implementaciones en dispositivos FPGA.

Otro tipo de arquitecturas parcialmente paralelas propuestas son las *layered* y *shuffled*, las cuales implementan de forma eficiente los métodos de actualización por capas del mismo nombre. En [46] y [47] se presentan dos arquitecturas *shuffled* basadas en registros. Las dos arquitecturas son similares y en ambas es necesario modificar el algoritmo αMS, aproximando el cálculo del primer y segundo mínimo, para reducir la complejidad *hardware*, lo que da lugar a que aparezcan pérdidas en las prestaciones. Por otra parte, en la literatura se encuentran implementaciones de la arquitectura *layered* basadas en memorias [29, 48–50] que alcanzan tasas de decodificación del orden de cientos de Mbps e implementaciones de

la arquitectura *layered* basadas en registros [51–54] para tasas de decodificación muy altas (superiores al Gbps). La arquitectura *layered* basada en memorias se presenta inicialmente en [48]. Con ésta, en [49] se implementa un decodificador multimodo para el estándar 802.16e. En [29] se reordena la matriz para el código IEEE 802.11n, de forma que se pueda paralelizar el procesado de las capas de manera eficiente. Un método similar se propone en [50] para el código de tasa $1/2$ del estándar IEEE 802.16e. La arquitectura *layered* basada en registros se utiliza en [51] con el algoritmo MS. En esta implementación se reduce el almacenamiento aprovechando que en los mensajes de salida de las unidades CNP solamente hay dos magnitudes posibles (dos mínimos). En el trabajo presentado en [52] se realiza un post-procesado con el fin de mejorar las prestaciones a bajas tasas de error. En [53] y [54] se analizan dos esquemas de comunicación entre unidades que permiten reducir la congestión en el rutado.

En este trabajo se han implementado arquitecturas hardware para la decodificación de códigos LDPC. Todas ellas tienen como objetivo alcanzar tasas de decodificación muy altas (del orden de Gbps). En primer lugar, los algoritmos presentados en los capítulos 3 y 4 se han implementado con las arquitecturas completamente paralela y parcialmente paralela basada en memorias. Esto se ha hecho con la finalidad de evaluar el impacto de éstos en los resultados *hardware* (área y tasa de decodificación). Aunque la arquitectura parcialmente paralela basada en memorias no consigue las tasas de decodificación deseadas, ésta nos sirve como referencia y además es la que se utiliza en el emulador hardware presentado en el capítulo 6. En segundo lugar, se han implementando las arquitecturas existentes más adecuadas para tasas de decodificación muy altas: SMP, *shuffled* y *layered* basada en registros. Por otra parte, se ha propuesto una mejora a la arquitectura SMP, denominada ISMP [55], que consigue reducir el área en un $30\,\%$. Además, se ha propuesto una arquitectura para el método de actualización *x-shuffled* (presentado en la Sección 4.3), que consigue tasas de decodificación superiores a los 15 Gbps [56]. Otra contribución de este trabajo es una nueva arquitectura para el cálculo de los dos mínimos, la cual consigue entre un $1.6\,\%$ y un $17.4\,\%$ de reducción en la complejidad *hardware* (área) comparada con las arquitecturas existentes [57–59]. El cálculo de los dos mínimos es necesario en la mayoría de las arquitecturas que utilizan los algoritmos basados en MS y además, representa un alto porcentaje del área total del decodificador.

El código utilizado para todas las implementaciones es el RS(6,32,6) del estándar IEEE 802.3an debido a que está especificado para su operación a muy alta velocidad (10 Gbps). Este código está construido con base en los códigos Reed-Solomon con dos bits de información [16] y tiene longitud $N = 2048$ y tasa de codificación $R = 1723/2048$. Su matriz de paridad $\mathbf{H}$ es regular y estructurada con $M_b = 6$, $N_b = 32$ y $z = 64$. Para una mayor claridad, tanto en la explicación de las arquitecturas como en las representaciones gráficas, se ha usado un código de menor longitud, el RS(3,6,3). La matriz de paridad de este código, la cual se muestra en la Figura 5.1, es regular y estructurada con $M_b = 3$, $N_b = 6$ y $z = 8$.

Todas las arquitecturas que se presentan en este capítulo se han implementado en un ASIC usando la librería de celdas estándar Faraday de 90nm con 8 capas metálicas [1]. Además, las implementaciones con la arquitectura parcialmente paralela basada en memorias se han realizado también para un dispositivo FPGA de bajo coste (Altera Cyclone IV-E).

$$
\mathbf{H} = \left[\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]
$$

Figura 5.1: Matriz de paridad **H** del código RS(3,6,3) de tamaño 24×48 y pesos $\rho_m = 6$ y $\gamma_n = 3$ y . La matriz es regular y estructurada con dimensiones $M_b = 3, N_b = 6$ y $z = 8$.

5.1. Cálculo de dos mínimos

Tal y como se verá en las siguientes secciones, en las arquitecturas para la implementación hardware de decodificadores LDPC de alta velocidad, del orden de Gbps, se requiere un alto grado de paralelismo, siendo necesario la implementación de cientos de unidades CNP y VNP. Por tanto, es evidente que si se consigue reducir la complejidad de estas celdas, el impacto de dicha reducción se verá multiplicado por el número de veces que está replicada en el decodificador. Esta sección se centra en la reducción de complejidad del operador que calcula el primer y segundo mínimo, definidos como $m1$ y $m2$, en la fase de actualización de los nodos de comprobación (ecuación 3.2), de algoritmos basados en MS. Esta operación se realiza en las unidades CNP y constituye el bloque con mayor complejidad de los que la forman.

En este sentido, varios autores han propuesto mejoras de los circuitos para realizar el cálculo de los dos mínimos. En [57] se proponen dos arquitecturas para el cálculo eficiente de los dos mínimos, denominadas *sorting based* (SB) y *tree structure* (TS). La arquitectura SB se basa en el algoritmo de ordenación descrito en [60], mientras que en la arquitectura TS el cálculo de los mínimos se realiza con una estructura en árbol utilizando bloques *Radix-2*, los cuales seleccionan los dos valores mínimos de dos parejas de entrada ($m1$,$m2$). En [51, 58] se presenta una modificación de la arquitectura SB, denominada mSB, que consigue reducir el área de la implementación *hardware*. En [59] se propone el uso de múltiples *Radix-K* (MR) con una estructura en árbol que mejora la velocidad (disminuye el camino crítico), a costa de aumentar la complejidad *hardware* (mayor área). En esta tesis se propone una arquitectura basada en la TS, en la que se optimizan los bloques *Radix-2*, y cuya implementación requiere menor área que las existentes y consigue una buena relación entre área y velocidad.

A continuación se explican en detalle las arquitecturas existentes mSB, TS y MR y la arquitectura propuesta, denominada mTS. Además, se realiza una comparación de los resultados de implementación para diferentes casos (número de entradas).

5.1.1. Arquitectura mSB

En esta arquitectura primero se calcula el valor mínimo y su posición mediante una estructura de comparadores en árbol, tal y como se muestra en la Figura 5.2 para el caso de 8 entradas. En la primera etapa del árbol ($l = 0$), cada bloque mV_n compara dos valores de entrada $v_{2\cdot n}$ y $v_{2\cdot n+1}$ y selecciona el valor mínimo. La salida de cada bloque, $m_n^{l=0}$, se conforma con el valor mínimo y la posición relativa de éste, donde la posición relativa se refiere al índice de la entrada correspondiente en el bloque mV_n (“0” si el mínimo corresponde a la primera entrada y “1” en caso contrario). En las etapas siguientes ($l \geq 1$), cada bloque mV_n selecciona el valor mínimo entre las entradas $m_{2\cdot n}^{l-1}$ y $m_{2\cdot n+1}^{l-1}$ y determina su posición relativa (“0” o “1”). La posición asociada al valor de salida se calcula concatenando la posición asociada al valor de entrada seleccionado como mínimo en el bloque mV_n con la posición relativa. Por ejemplo, si la entrada $m_{2\cdot n}^{l-1}$ corresponde al mínimo y ésta tiene asociada la posición “1”, la posición asociada al valor de salida es “01”. De esta manera, en la última etapa del árbol ($l = L-1$) se obtiene el valor mínimo ($m1 = m_0^{L-1}$) y su posición (p) en formato binario.

Con la posición del primer mínimo (p) se seleccionan los candidatos a segundo mínimo ($m2$) de cada etapa del árbol, donde los candidatos son aquellos valores que se han com-

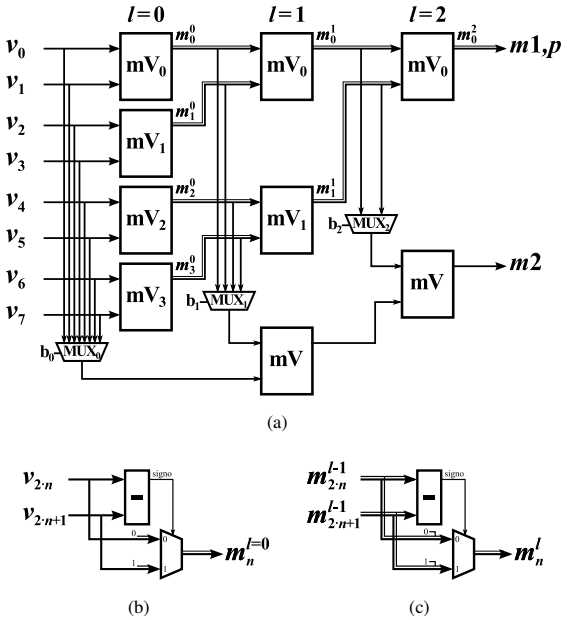


Figura 5.2: Arquitectura *sorting based* modificada (mSB) con 8 entradas. (a) Estructura general. (b) Bloque mV (etapa inicial). (c) Bloque mV.

parado con el primer mínimo. Por ejemplo, si se asume que el valor mínimo corresponde a la entrada v_3 (p ="011"), el candidato de la primera etapa es el valor v_2, el de la segunda etapa es $m_0^{l=0}$ y el de la tercera etapa es $m_1^{l=1}$. Todos los candidatos son comparados en un segundo árbol para determinar el valor mínimo de éstos, el cual corresponde al segundo mínimo ($m2$). Los candidatos se seleccionan mediante los multiplexores $\mathrm{MUX_s}$, donde s es el índice de la etapa del árbol y $\mathrm{b_s}$ el valor de selección en formato binario. En este caso, el $\mathrm{MUX_0}$ debe seleccionar entre 8 valores posibles, el $\mathrm{MUX_1}$ entre 4 y el $\mathrm{MUX_2}$ entre 2, por lo que $\mathrm{b_0}$, $\mathrm{b_1}$ y $\mathrm{b_2}$ tienen anchos de palabra 3, 2 y 1, respectivamente. En la Tabla 5.1 se muestran los valores de b_s para los posibles valores de posición p, donde $\{p_2, p_1, p_0\}$ es la representación binaria, siendo p_2 el bit más significativo.

En la Tabla 5.1 se aprecia que $\mathrm{b_0} = \{p_2, p_1, \overline{p_0}\}$, $\mathrm{b_1} = \{p_2, \overline{p_1}\}$ y $\mathrm{b_2} = \{\overline{p_2}\}$, donde $\overline{p_n}$ es el complemento a 1 de p_n. Por lo tanto, la generación de las palabras de selección se realiza con tres negadores únicamente.

5.1.2. Arquitectura TS

El cálculo de los dos mínimos ($m1$ y $m2$) en esta arquitectura se realiza mediante una estructura en árbol de bloques *Radix-2*, como se muestra en la Figura 5.3 para el caso de 8 entradas. En la primera etapa del árbol ($l = 0$) se realiza una ordenación por parejas de los datos de entrada. Cada bloque *Radix-2* inicial ($\mathrm{R2i}_n$) selecciona el primer mínimo ($m1_n^{l=0}$) y segundo mínimo ($m2_n^{l=0}$) de dos entradas $v_{2\cdot n}$ y $v_{2\cdot n+1}$. En las etapas siguientes, cada bloque *Radix-2* ($\mathrm{R2}_n$) recibe dos parejas ($m1$,$m2$) de la etapa anterior, definidas como

Tabla 5.1: Valores de selección de los candidatos a segundo mínimo de acuerdo a la posición del primer mínimo (p) para 8 valores de entrada.

p $\{p_2, p_1, p_0\}$	$\mathbf{b}_0$	$\mathbf{b}_1$	$\mathbf{b}_2$
000	001	01	1
001	000	01	1
010	011	00	1
011	010	00	1
100	101	11	0
101	100	11	0
110	111	10	0
111	110	10	0

($m1^{l-1}_{2\cdot n}$,$m2^{l-1}_{2\cdot n}$) y ($m1^{l-1}_{2\cdot n+1}$,$m2^{l-1}_{2\cdot n+1}$), y calcula los dos valores mínimos de éstas ($m1^{l}_{n}$,$m2^{l}_{n}$). Primero se realizan las operaciones cruzadas min($m1^{l-1}_{2\cdot n}$,$m1^{l-1}_{2\cdot n+1}$), min($m2^{l-1}_{2\cdot n}$,$m1^{l-1}_{2\cdot n+1}$) y min($m2^{l-1}_{2\cdot n+1}$,$m1^{l-1}_{2\cdot n}$). El valor mínimo de salida ($m1^{l}_{n}$) corresponde al resultado de la primera operación cruzada. El segundo mínimo ($m2^{l}_{n}$) depende de la posición relativa del primer mínimo: si el primer mínimo es $m1^{l-1}_{2\cdot n}$, el segundo mínimo es el resultado de la segunda operación cruzada, y si el mínimo es $m1^{l-1}_{2\cdot n+1}$, el segundo mínimo corresponde al resultado de la tercera operación cruzada.

5.1.3. Arquitectura MR

En esta arquitectura también se utiliza una estructura en árbol, al igual que en la TS. Sin embargo, en ésta los bloques que componen el árbol son *Radix-K*. Cada bloque *Radix-K* calcula los valores $m1$ y $m2$ de K entradas, las cuales son simples en la primera etapa y parejas ($m1$,$m2$) en etapas superiores. En [59] se presentan dos casos de la arquitectura MR, uno en el que K es igual para todos los bloques en una etapa del árbol y otro en el que K puede variar entre los bloques de una misma etapa. Aunque el segundo caso tiene más opciones, se ha demostrado que los resultados son similares a los obtenidos con K fijo en cada etapa. Por lo tanto, en este trabajo solamente se analiza e implementa el primer caso. Cabe destacar que la estructura del *Radix-2* en esta arquitectura no es exactamente igual a la del *Radix-2* de la arquitectura TS.

En la Figura 5.4 se muestran las 4 posibles configuraciones de la arquitectura MR para el caso de 8 entradas. En la primera, MR(8), el árbol se compone únicamente de un bloque *Radix-8* inicial ($\mathrm{R8i}_0$) que calcula los dos mínimos de las 8 entradas. En la segunda configuración (MR(4,2)) el árbol tiene dos etapas (L=2). En la primera etapa los bloques *Radix-4* inicial ($\mathrm{R4i}_n$) calculan los valores $m1^{l=0}_{n}$ y $m2^{l=0}_{n}$ en bloques de 4 entradas (2 bloques) y en la segunda etapa el bloque *Radix-2* ($\mathrm{R2}_0$) calcula los valores $m1^{l=1}_{0}$ y $m2^{l=1}_{0}$ de las dos parejas de mínimos calculadas en la etapa anterior. La tercera configuración (MR(2,4)) es similar a la segunda, el árbol también tiene dos etapas, pero los *Radix-K* están invertidos. En la primera etapa los 4 bloques $\mathrm{R2i}_n$ calcula 4 parejas de mínimos y en la segunda etapa el bloque $\mathrm{R4}_0$ calcula los valores $m1^{l=1}_{0}$ y $m2^{l=1}_{0}$ de las 4 parejas calculadas en la etapa anterior. En la última configuración (MR(2,2,2)) el árbol consta de $L = 3$ etapas y en cada una se utilizan bloques *Radix-2*. Los mínimos $m1$ y $m2$ corresponden a los valores $m1^{L-1}_{0}$ y $m2^{L-1}_{0}$, respectivamente, donde $L-1$ es el índice de la última etapa del árbol.

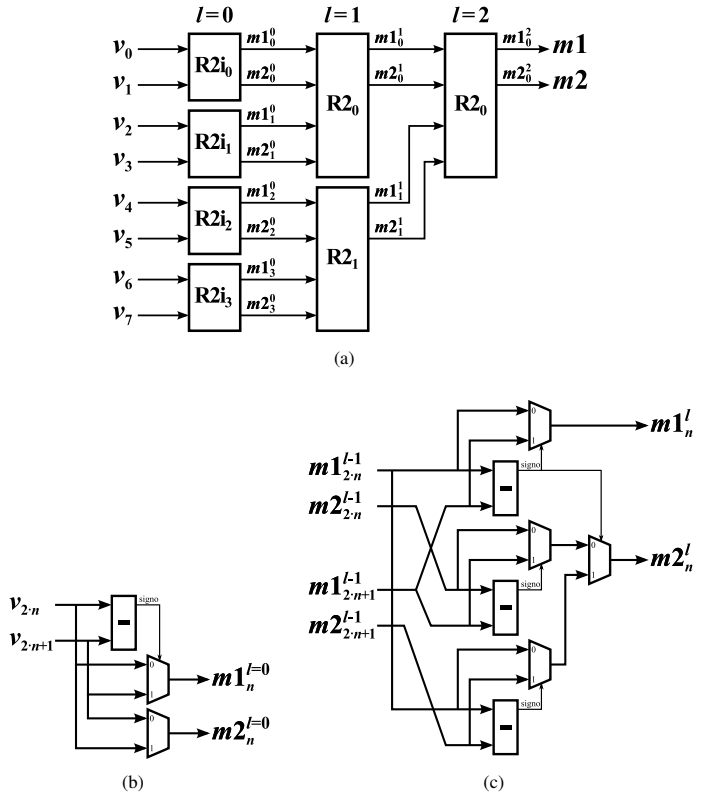


Figura 5.3: Arquitectura *tree structure* (TS) con 8 entradas. (a) Estructura general. (b) Bloque *Radix-2* inicial (R2i). (c) Bloque *Radix-2* (R2).

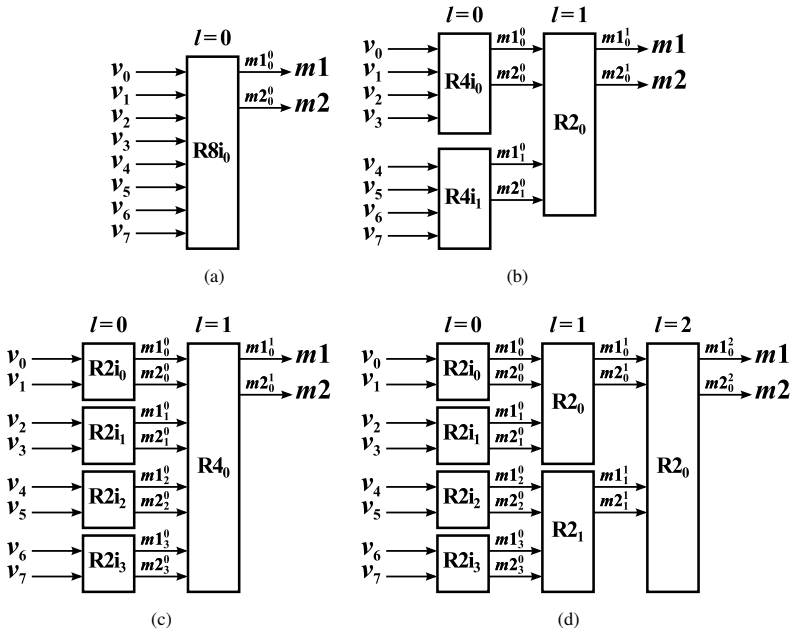


Figura 5.4: Arquitectura *multiple Radix* (MR) con 8 entradas. (a) Configuración MR(8). (b) Configuración MR(4,2). (b) Configuración MR(2,4). (b) Configuración MR(2,2,2).

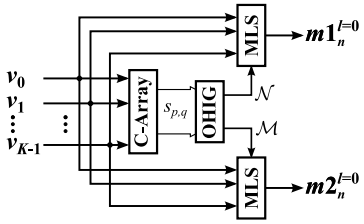


Figura 5.5: Bloque *Radix-K* inicial.

La estructura del bloque *Radix-K* inicial (Rki$_n$) se muestra en la Figura 5.5. Primero se realiza una comparación cruzada (bloque C-Array) entre las entradas $v_{n\cdot K+i}$, para $0 \leq i \leq K-1$. Con estas comparaciones se generan los índices binarios $s_{p,q}$ para $0 \leq p,q \leq K-1$, de manera que $s_{p,q} = 1$ si $v_p \leq v_q$ y $s_{p,q} = 0$ en caso contrario. Mediante los índices de comparación $s_{p,q}$ se construyen los vectores $\mathcal{N}$ y $\mathcal{M}$ de K elementos (bloque OHIG), tal y como se describe en (5.1) y (5.2), respectivamente, donde $\vee$ representa la operación OR y $\wedge$ la operación AND.

$$\mathcal{N}[p] = \bigwedge_{q=0}^{K-1} s_{p,q} \tag{5.1}$$

$$\mathcal{M}[p] = \bigwedge_{q=0}^{K-1} (s_{p,q} \wedge \overline{\mathcal{N}[p]}) \vee \mathcal{N}[q] \tag{5.2}$$

El vector $\mathcal{N}$ identifica la posición del primer mínimo en codificación de un solo bit (*one-hot index*). Esto quiere decir que el vector contiene un único uno en la posición del primer mínimo. De la misma manera, el vector $\mathcal{M}$ identifica la posición del segundo mínimo y se genera enmascarando el primer mínimo. Los vectores $\mathcal{N}$ y $\mathcal{M}$ se usan como señal de selección en los bloques MLS (*Mux-Like structure*), los cuales computan las ecuaciones 5.3 y 5.4 para seleccionar los valores mínimos de salida $m1_n^{l=0}$ y $m2_n^{l=0}$, respectivamente.

$$m1_n^{l=0} = \bigvee_{u=0}^{K-1} v_u \wedge \mathcal{N}[u] \tag{5.3}$$

$$m2_n^{l=0} = \bigvee_{u=0}^{K-1} v_u \wedge \mathcal{M}[u] \tag{5.4}$$

Los bloques *Radix-K* (Rk$_n$) computan los dos mínimos ($m1_n^l$,$m2_n^l$) entre K parejas de entrada ($m1$,$m2$) computadas en la etapa anterior ($l-1$) del árbol y su estructura se muestra en la Figura 5.6. El cálculo del valor mínimo ($m1_n^l$) se realiza de la misma forma que en el bloque R2i$_n$, con los K valores de entrada $m1_{n\cdot K}^{l-1} \ldots m1_{(n+1)\cdot K-1}^{l-1}$. Para el cálculo del segundo mínimo hay dos casos posibles: 1) que el segundo mínimo se encuentre entre los K valores de entrada $m1^{l-1}$, 2) que el segundo mínimo se encuentre entre los K valores de entrada $m2^{l-1}$. Para el primer caso se utiliza la misma estructura que el bloque R2i$_n$. Para el segundo caso, primero se generan los índices $s'_{p,q}$, donde $s'_{p,q} = 1$ si $m2_p^{l-1} \leq m1_q^{l-1} \vee p = q$ y $s'_{p,q} = 0$ en caso contrario. Luego se genera el vector $\mathcal{F}$ de K elementos (ecuación 5.5),

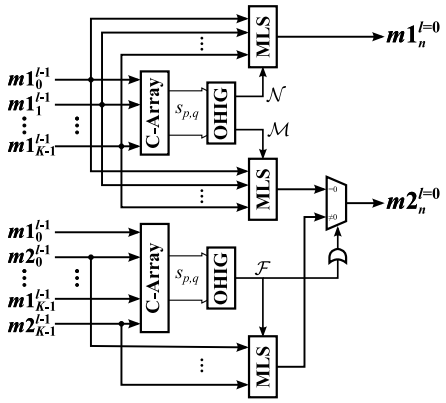


Figura 5.6: Bloque *Radix-K*.

el cual identifica la posición del segundo mínimo si éste se encuentra entre los K valores de entrada $m2^{l-1}$ y en tercer lugar, se realiza la selección del mínimo (segundo caso) con un bloque MLS. Si todos los elementos del vector $\mathcal{F}$ son iguales a 0, el segundo mínimo $m2_n^l$ es el correspondiente al caso 1 y en caso contrario es el correspondiente al caso 2.

$$\mathcal{F}[p] = \bigwedge_{q=0}^{K-1} s'_{p,q} \tag{5.5}$$

5.1.4. Arquitectura mTS

La arquitectura TS modificada (mTS) realiza el cálculo de los dos mínimos mediante una estructura en árbol, de la misma manera que la arquitectura TS (Figura 5.3). La ordenación que se realiza en la primera etapa del árbol es igual que en la arquitectura TS, por lo que los bloques R2i son los mismos. La diferencia entre las dos arquitecturas (TS *vs.* mTS) radica en los bloques *Radix-2* del resto de etapas del árbol ($l \geq 2$). En este trabajo se ha propuesto una optimización de los bloques R2 con la que se reduce el número de operaciones de resta que se realizan en la arquitectura original TS.

En la Figura 5.7 se muestra la estructura del bloque R2 optimizado. En ésta primero se hace una selección del valor mínimo ($m1_n^l$) entre los valores $m1_{2\cdot n}^{l-1}$ y $m1_{2\cdot n+1}^{l-1}$. El valor descartado de la comparación anterior, candidato a segundo mínimo, es comparado con los valores de entrada $m2_{2\cdot n}^{l-1}$ y $m2_{2\cdot n+1}^{l-1}$ para determinar el segundo mínimo ($m2_n^l$). Sin embargo, no es necesario realizar las dos comparaciones, ya que se sabe que $m1 \leq m2$. Por lo tanto, solamente es necesario comparar el valor descartado con el valor $m2$ de la pareja contraria. En otras palabras, si el primer mínimo es $m1_{2\cdot n}^{l-1}$, el valor descartado $m1_{2\cdot n+1}^{l-1}$ solamente se compara con $m2_{2\cdot n}^{l-1}$, ya que se sabe que $m2_{2\cdot n+2}^{l-1} > m1_{2\cdot n+1}^{l-1}$. Por contra, si el primer mínimo es $m1_{2\cdot n+2}^{l-1}$, el descarte $m1_{2\cdot n}^{l-1}$ se compara únicamente con $m2_{2\cdot n+1}^{l-1}$ para determinar el segundo mínimo $m2_n^l$.

Comparando las dos estructuras del bloque R2, se puede ver que en la arquitectura mTS se requieren 2 restas y 4 multiplexores, mientras que en la arquitectura TS son requeridas 3

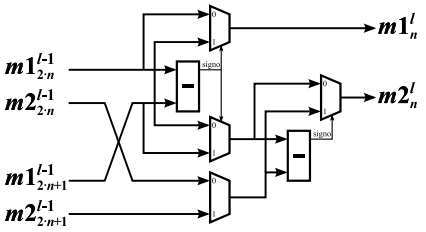


Figura 5.7: Bloque *Radix-2* (R2) optimizado.

restas y 4 multiplexores. Por consiguiente, el bloque R2 de la arquitectura propuesta utiliza una resta menos.

5.1.5. Resultados de implementación

Las arquitecturas para el cálculo de los dos mínimos mSB, TS, MR y mTS se han implementado en un ASIC para diferentes casos (número de entradas). Las cuatro arquitecturas se han evaluado para 8, 12, 16, 24, 30 y 32 entradas. En la arquitectura MR se han implementado todas las combinaciones posibles, para cada caso, pero solamente se muestran las que menor área requieren por número de etapas del árbol. La nomenclatura usada en la arquitectura MR es $(K_1, K_2, \ldots, K_L)$, donde L es el número de etapas del árbol y K_l el orden del *Radix-K* (número de entradas a cada bloque). Además, para comprobar cómo afecta cada topología del cálculo de los mínimos en un decodificador de LDPC, se han evaluado las diferentes arquitecturas implementándolas en un decodificador de códigos LDPC tipo *layered*, cuya estructura se explica en la sección 5.6. Estas implementaciones se han realizado utilizado el algoritmo αMS y para el código RS(6,32,6) del estándar IEEE 802.3an. En este caso se requiere el cálculo de dos mínimos con 32 entradas en las unidades CNP.

Los resultados de implementación con la librería Faraday de 90nm se muestran en las Tablas 5.2 y 5.3. En la primera tabla se comparan los resultados obtenidos para los diferentes casos (número de entradas) de las arquitecturas para el cálculo de los dos mínimos, teniendo en cuenta que los datos de área y camino crítico son obtenidos después de la síntesis. En la segunda tabla se pueden ver los resultados de las implementaciones del decodificador *layered* usando el algoritmo αMS con las arquitecturas para el cálculo de los dos mínimos mSB, TS, MR y mTS. En este caso, los resultados de área y camino crítico se obtienen después del emplazado y rutado. La frecuencia se refiere a la máxima frecuencia de reloj a la que puede operar la implementación y se calcula como la inversa del camino crítico. El ancho de palabra de las señales es de 6 bits para todas las implementaciones.

En la Tabla 5.2 se puede observar que para los casos implementados (8, 12, 16, 24, 30 y 32 entradas) la arquitectura mTS es la que menos área requiere. Respecto a la arquitectura mSB la reducción en área varía entre el 6.2 % y el 9.08 %. Comparado con la arquitectura TS la reducción es ligeramente mayor, de entre el 8.3 % y el 9.92 %. En relación a la arquitectura MR el rango de porcentaje de reducción en el área es más amplio, siendo 1.6 % el porcentaje mínimo y 81.6 % el máximo. Para el caso de interés (32 entradas) la reducción en área de la arquitectura mTS respecto a las arquitecturas mSB y TS es del 6.4 % y 8.8 %, mientras que comparado con la arquitectura MR se consigue un 7.8 %, 7.7 %, 14.3 % y 37.5 % para un configuración de 5, 4, 3 y 2 etapas, respectivamente.

Tabla 5.2: Resultados de implementación en un ASIC de las arquitecturas para el cálculo de los dos mínimos mSB, TS, MR y mTS usando la librería Faraday de 90nm. El ancho de palabra de las señales en todas las implementaciones es de 6 bits.

Número de entradas	**Arquitectura**	**Área** [μm^2]	**Camino crítico** [ns]	**Frecuencia** [MHz]
8	mSB	1189	2.57	389
	TS	1200	1.84	543
	MR(8)	1985	1.18	847
	MR(4,2)	1328	1.39	719
	MR(2,2,2)	1223	1.77	565
	mTS	1081	1.84	543
12	mSB	1901	3.41	293
	TS	1916	2.34	427
	MR(12)	4203	1.41	709
	MR(4,3)	2254	1.68	595
	MR(3,2,2)	1890	1.91	524
	mTS	1757	2.34	427
16	mSB	2589	3.46	289
	TS	2622	2.12	472
	MR(16)	7487	1.79	559
	MR(4,4)	3437	1.83	546
	MR(4,2,2)	2917	1.94	515
	MR(2,2,2,2)	2909	2.13	469
	mTS	2400	2.12	472
24	mSB	4435	3.98	251
	TS	4562	2.61	383
	MR(24)	17619	2.37	422
	MR(6,4)	5759	1.97	508
	MR(4,2,3)	4635	2.20	455
	MR(3,2,2,2)	4229	2.41	415
	mTS	4160	2.61	383
30	mSB	5573	3.98	251
	TS	5782	2.62	382
	MR(30)	26550	3.05	328
	MR(5,6)	7150	1.82	549
	MR(5,2,3)	5947	2.12	472
	mTS	5210	2.62	382
32	mSB	6030	3.99	251
	TS	6190	2.64	379
	MR(32)	30626	3.53	283
	MR(4,8)	9031	2.11	474
	MR(4,2,4)	6587	2.31	433
	MR(4,2,2,2)	6117	2.47	405
	MR(2,2,2,2,2)	6123	2.58	388
	mTS	5644	2.64	379

Tabla 5.3: Resultados de implementación en un ASIC de la arquitectura *layered* con el algoritmo αMS usando diferentes arquitecturas para el cálculo de los dos mínimos. Las implementaciones se han realizado para el código RS(6,32,6) del estándar IEEE 802.3an, usando la librería Faraday de 90nm.

Arquitectura cálculo de mínimos	**Área** [mm^2]	**Camino crítico** [ns]	**Frecuencia** [MHz]
mSB	5.18	6.58	152
TS	5.23	4.99	200
MR(4,8)	6.11	4.67	214
MR(4,2,4)	5.52	4.70	213
MR(4,2,2,2)	5.36	5.06	198
mTS	4.98	4.97	201

En términos de frecuencia, la arquitectura MR es la que mejores resultados presenta, alcanzando frecuencias de hasta 847 MHz (8 entradas, MR(8)). Por el contrario, la arquitectura mSB es la más lenta, por lo que es la que mayor retardo tiene. La frecuencia de trabajo de las arquitecturas TS y mTS es prácticamente la misma en todos los casos. Sin embargo, se puede considerar más eficiente la arquitectura mTS debido a que con menos área consigue la misma frecuencia. Para el caso de interés, con la arquitectura MR de 2 etapas (MR(4,8)) se alcanza una frecuencia de 474 MHz, un 20 % mayor que la conseguida con la arquitectura mTS, no obstante, ésta requiere un 37.5 % más de área. Algo similar ocurre con las configuraciones de 5, 4 y 3 etapas, las cuales consiguen mayores frecuencias (388 MHz, 405 MHz y 433 MHz, respectivamente) pero también requieren más área. En general, el porcentaje de reducción de área con la arquitectura mTS respecto a la MR es mayor que el porcentaje de decremento de la frecuencia.

En los resultados de las implementaciones del decodificador *layered* con el algoritmo αMS, mostrados en la Tabla 5.3, se observa que con la arquitectura mTS se consigue la menor área de implementación. Comparado con las implementaciones que usan las arquitecturas mSB y TS la reducción es del 3.8 % y 4.8 %, respectivamente. Además, comparado con éstas la frecuencia de trabajo es mayor, un 24.5 % respecto a mSB y un 0.4 % respecto a TS. En relación con la arquitectura MR también se consigue menos área en la implementación del decodificador: 18 % menos que con MR(4,8), 9.8 % menos que con MR(4,2,4) y 7.1 % menos que con MR(4,2,2,2). En términos de frecuencia, las implementaciones con la arquitectura MR de 2 y 3 etapas superan a la implementación con la arquitectura mTS, sin embargo, la diferencia es menor a 13 MHz. En el decodificador *layered* las arquitecturas para el cálculo de los mínimos son de 32 entradas, por lo que comparando estos resultados con los correspondientes de la Tabla 5.2, se aprecia que el porcentaje de reducción en el área usando la arquitectura mTS respecto a MR es menor, pero también es menor y en mayor medida la diferencia en la frecuencia.

En conclusión, la arquitectura propuesta mTS es la más óptima en términos de área para todos los casos evaluados. La arquitectura MR es la que mayor frecuencia de trabajo alcanza, sin embargo, en la implementación del decodificador *layered*, cuyo grado de paralelismo es elevado, el uso de ésta apenas representa una ganancia del 6 % en frecuencia y requiere un 18.5 % más de área, comparado con el uso de la arquitectura mTS. En las implementaciones *hardware* de los decodificadores LDPC para el estándar IEEE 803.an, que se exponen en éste capítulo, se requiere un nivel de paralelismo alto, por lo que la arquitectura

para el cálculo de los dos mínimos que mejor se adapta es la mTS.

5.2. Arquitectura Completamente paralela

Esta arquitectura aprovecha el paralelismo inherente de los códigos LDPC, por lo que tiene una relación directa con el grafo de Tanner (Sección 2.2), en la que las unidades CNP corresponden con los nodos de comprobación C_m y las unidades VNP con los nodos de bit V_n. Las implementaciones *hardware* con esta arquitectura [31, 33] requieren una gran cantidad de recursos y presentan una alta densidad de conexiones. El gran número de conexiones genera retardos elevados en la propagación de los mensajes que limitan la frecuencia de funcionamiento, lo que afecta negativamente a la tasa de decodificación [36].

Desde el punto de vista de la matriz de paridad, las unidades CNP están asociadas a las filas y las unidades VNP a las columnas. Por lo tanto, para una matriz de tamaño $M \times N$ se requieren M unidades CNP y N unidades VNP. En cada iteración del proceso de decodificación, primero las unidades VNP procesan en paralelo las N columnas de la matriz de paridad y después las unidades CNP procesan en paralelo las M filas. Debido a que los procesos se realizan en paralelo, los mensajes generados por la unidades VNP y CNP deben ser leídos de forma concurrente, por lo que se usan registros para su almacenamiento. Las conexiones entre unidades están definidas por los unos de la matriz $\mathbf{H}$, de la misma manera que definen las conexiones entre nodos en el grafo de Tanner. Por lo tanto, el número de unidades VNP conectadas a una CNP es igual al peso de fila ρ_m y el número de unidades CNP conectadas a una VNP es igual al peso de columna γ_n. En los códigos regulares los pesos son constantes para todo m y n, por consiguiente todas las unidades VNP y CNP son regulares, con el mismo número de conexiones. El cálculo de la paridad, necesario para implementar la finalización anticipada, se realiza en paralelo para todas las filas de la matriz. Por cada un de las filas se calcula la ecuación de paridad correspondiente mediante puertas XOR, usando los bits decodificados z_n. Para verificar que todas las ecuaciones de paridad se cumplen se realiza una operación AND con todos los resultados.

La Figura 5.8 muestra la arquitectura completamente paralela para el código RS(3,6,3), cuya matriz de paridad (Figura 5.1) es de tamaño 24×48 y pesos $\rho_m = 6$ y $\gamma_n = 3$. La arquitectura consta de 48 unidades VNP y 24 unidades CNP conectadas al bloque "Matriz de Interconexión". Este bloque implementa las conexiones físicas entre unidades y su estructura depende de la matriz de paridad $\mathbf{H}$. Por ejemplo, la unidad VNP_1 está conectada con las unidades CNP_1, CNP_{10} y CNP_{19}, cuyos índices corresponden con las posiciones de los unos en la primera columna de $\mathbf{H}$ y la unidad CNP_1 está conectada con las unidades VNP_1, VNP_9, VNP_{17}, VNP_{25}, VNP_{33} y VNP_{41}, siendo 1, 9, 17, 25, 33 y 41 las posiciones de los unos en la primera fila de $\mathbf{H}$. El almacenamiento de los mensajes $\lambda_{n,m}$ y $\mu_{m,n}$ durante el proceso iterativo de decodificación se hace en los registros LR y MR, respectivamente. Las señales de entrada l_n denotan la información *a priori* del canal y las señales de salida z_n corresponden con los bits decodificados.

La arquitectura de las unidades VNP y CNP depende del algoritmo que se implemente para decodificar. La unidad VNP es común para los algoritmos SP y basados en MS descritos en los capítulos 3 y 4. Cada una de estas unidades implementa las operaciones descritas en la ecuaciones (3.2) y (3.4). El formato usado para los mensajes $\mu_{m,n}$ y $\lambda_{n,m}$ es signo-magnitud. En la Figura 5.9 se muestra la arquitectura de la unidad VNP_n para el código RS(3,6,3). Primero, las señales de entrada $\mu_{1,n}$, $\mu_{2,n}$ y $\mu_{3,n}$, cuyo formato es signo-

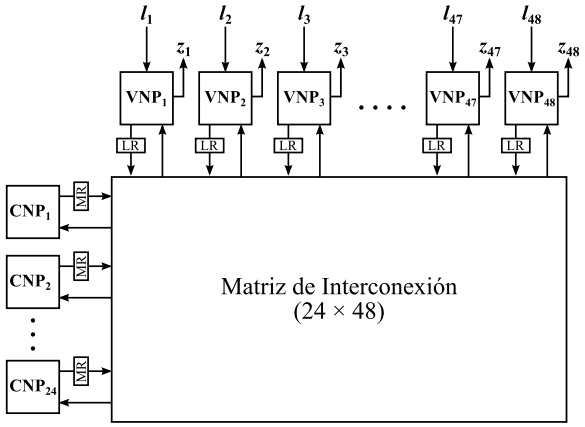


Figura 5.8: Arquitectura completamente paralela para el código RS(3,6,3). Consta de 48 unidades VNP y 24 unidades CNP que conectan entre si. Los mensajes se almacenan en los registros LR y MR.

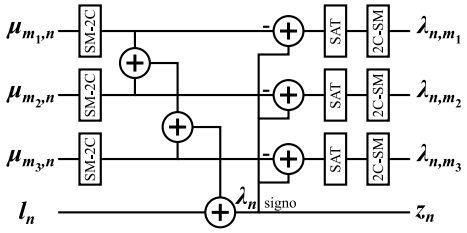


Figura 5.9: Arquitectura de la unidad VNP común a los algoritmos SP y basados en MS para el código RS(3,6,3). La unidad recibe las señales $\mu_{m,n}$ provenientes de las unidades CNP y la señal de entrada l_n y calcula las señales $\lambda_{n,m}$ y el bit decodificado z_n.

magnitud, son convertidas al formato complemento a 2 (bloque SM-2C) y luego sumadas mediante una estructura en árbol. Al resultado se le suma la señal l_n obteniendo el LLR *a posteriori* de bit (λ_n) y en la siguiente etapa, se resta la señal λ_n a cada una de las entradas $\mu_{m,n}$ (en formato complemento a 2). Para evitar desbordamientos se han dejado crecer los datos de forma natural en el árbol de sumadores y en el resto de sumas/restas. Además, a los resultados de las restas se les aplica saturación (bloques SAT) para adaptar los datos al esquema de cuantificación de la salida. Por último, las señales saturadas son transformadas al formato signo-magnitud (bloque 2C-SM) para obtener las señales de salida $\lambda_{n,m}$. El bit decodificado z_n corresponde con el bit de signo (bit más significativo) de la señal λ_n.

A diferencia de las unidades VNP, las operaciones que se realizan en la unidad CNP son diferentes en cada uno de los algoritmos descritos en los capítulos 3 y 4. En la Figura 5.10 se muestran las arquitecturas de las unidades CNP_m para los algoritmos SP y MS, las cuales implementan las ecuaciones (3.1) y (3.5), respectivamente. La unidad CNP del algoritmo MS sirve como referencia para el resto de algoritmos basados en MS, ya que éstos son variaciones del mismo. Las señales de entrada $\lambda_{n,m}$ y salida $\mu_{m,n}$ para ambos algoritmos están en formato signo-magnitud.

En la unidad CNP del algoritmo SP se usan tablas, denominadas PLUT y RLUT, para computar las funciones no lineales $\Psi(x)$ y $\Psi(x)^{-1}$, respectivamente. Cada PLUT es direccionada con la magnitud de la señal $\lambda_{n,m}$ y las salidas son sumadas mediante una estructura en árbol. La exclusión presente en (3.1) se realiza restando la contribución del término n al resultado de la suma, lo que se traduce en restar la salida de la tabla correspondiente al total de la suma. Al igual que en las unidades CNP, los datos en el árbol de sumadores y en el resto de sumas/restas se han dejado crecer de forma natural. La salida de las restas son saturadas y truncadas en los bloques SAT y con los resultados se direccionan las tablas RLUT, calculando de esta manera la función $\Psi(x)^{-1}$. El truncado y saturado se aplica para limitar el tamaño de las tablas RLUT. La salida de las tablas corresponde a las magnitudes de las señales de salida $\mu_{m,n}$.

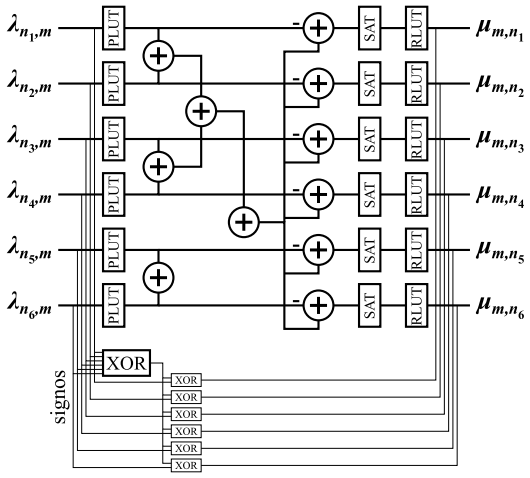
La actualización de paridad $\Gamma_{m,n}$ se realiza en dos pasos, primero se hace una operación XOR entre todos los signos de las señales de entrada $\lambda_{n,m}$ y después se realiza una XOR entre el resultado anterior y cada uno de los signos de la señales de entrada, obteniendo los signos de las señales de salida $\mu_{m,n}$.

La unidad CNP del algoritmo MS es mucho más simple. Primero, el bloque 2MIN calcula el primer y segundo mínimo de las magnitudes de las señales de entrada $\lambda_{n,m}$, como se explica en la Sección 5.1. A continuación, el primer valor mínimo es comparado con las magnitudes de las señales $\lambda_{n,m}$ y si hay coincidencia los multiplexores (MUX) seleccionan el segundo mínimo, en caso contrario seleccionan el primero (ecuación 3.6). La salidas de los bloques MUX corresponden con las magnitudes de las señales de salida $\mu_{m,n}$. Los signos de $\mu_{m,n}$ son calculados de la misma manera que en la unidad CNP del algoritmo SP.

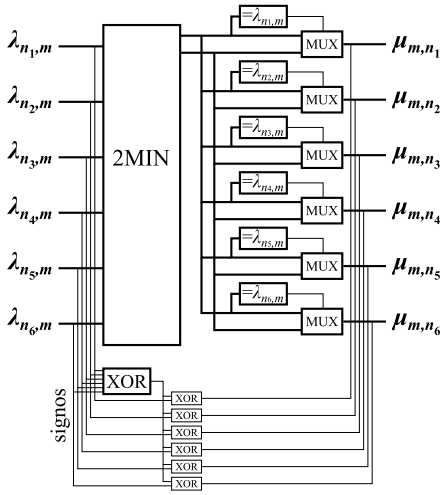
La tasa de decodificación para la arquitectura completamente paralela en términos de la longitud del código N, la frecuencia de reloj f y el número de iteraciones It se muestra en (5.6). El factor c del denominador depende del nivel de segmentación de las unidades, siendo 0 en el caso que las unidades sean completamente combinacionales. Normalmente, al aumentar el nivel de segmentación el camino crítico disminuye y por tanto la frecuencia de reloj aumenta. Sin embargo, en las implementaciones *hardware* con la arquitectura paralela el camino crítico está determinado por los retardos en la matriz de interconexión. Por lo tanto, un aumento en el grado de segmentación de las unidades hace que la tasa de decodificación disminuya, ya que el incremento de la frecuencia de reloj f es bajo y no compensa el aumento del factor $(2+c)$ del denominador. En conclusión, la mayor tasa de decodificación se consigue cuando el factor c es igual a 0.

$$T_{FP} = \frac{N \cdot f}{It \cdot (2+c)} \tag{5.6}$$

El término $2+c$ del denominador en (5.6) se refiere al número de ciclos de reloj que requiere una iteración. Cuando c es igual a 0, el número de ciclos de reloj por iteración es igual a 2. En el primero las unidades VNP procesan las filas de la matriz $\mathbf{H}$ y las unidades CNP están en estado de espera. En el segundo ciclo, son las unidades CNP las que procesan y las unidades VNP las que están en estado de espera. Por consiguiente, la relación entre ciclos de proceso y ciclos totales es de 0.5, lo que implica una HUE del 50 %.



(a)



(b)

Figura 5.10: Arquitectura de la unidad CNP para el código RS(3,6,3). La unidad calcula las señales $\mu_{m,n}$ utilizando las señales $\lambda_{n,m}$ generadas en las unidades VNP. (a) Unidad CNP del algoritmo SP. (b) Unidad CNP del algoritmo MS.

5.2.1. Resultados de implementación

La arquitectura completamente paralela se ha implementado en un ASIC con los algoritmos clásicos SP, MS, αMS y βMS y los propuestos iMS, αiMS y vwMS, para el código RS(6,32,6) del estándar IEEE 802.3an. La matriz de paridad de este código es regular con dimensiones 384×2048 y los pesos de fila ρ_m y columna γ_n son iguales a 32 y 6, respectivamente. Con el fin de conseguir la máxima tasa de decodificación, las unidades CNP y VNP se han implementado completamente combinacionales. En la implementación *hardware* se ha añadido un registro y un multiplexor por cada dato de entrada l_n con el fin de mantener estas señales durante el proceso de decodificación, ya que son utilizadas en todas las iteración por las unidades VNP. Además, se incluye el *hardware* de control necesario para generar las señales de habilitación y para realizar la finalización anticipada.

Los factores de corrección y escalado utilizados en las implementaciones son aquellos que consiguen las mejores prestaciones, de acuerdo al análisis realizado en capítulos anteriores. El factor α utilizando en los algoritmos αMS y αiMS es 0.75, al igual que el factor β del algoritmo βMS. En el algoritmo vwMS el vector de factores de corrección w y el de umbrales u es $[1.25; 1.75; 2.25; 2.75]$ y $[5; 10; 15]$, respectivamente. Además, el factor de escalado α para este algoritmo es también de 0.75.

La cuantificación usada consigue un buen compromiso entre las prestaciones de los algoritmos y el tamaño de palabra, tal y como se mostró en la Sección 3.4. La información *a priori* del canal l_n se ha cuantificado con un esquema $[6:2]$. Este mismo esquema es usado para las señales $\lambda_{n,m}$ y $\mu_{m,n}$ en los algoritmos SP, MS, αMS, βMS y vwMS. En los algoritmos iMS y αiMS se cuantifican las señales $\lambda_{n,m}$ con un esquema $[6:2]$ y las señales $\mu_{m,n}$ con un esquema $[4:0]$. Para evitar desbordamientos dentro de las unidades VNP, la parte entera de los datos se deja crecer 4 bits. En el algoritmos SP, la función no lineal $\Psi(x)$ se cuantifica con un esquema $[9:7]$. El crecimiento natural de los datos en el árbol de sumadores hace que la parte entera crezca 5 bits. Por lo tanto, en el bloque SAT se saturan y truncan los datos, con esquema $[14:7]$, para adaptarlos al esquema $[6:4]$. Por último, para la función $\Psi(x)^{-1}$ se utiliza el esquema de cuantificado $[6:2]$.

Los resultados de implementación con la librería Faraday de 90nm se muestran en la Tabla 5.4. Para todas las implementaciones se ha usado un porcentaje de utilización del 40 % con el fin de minimizar el efecto que tiene la alta congestión de conexiones sobre el retardo de los caminos. Los datos de área y camino crítico corresponden a los resultados de implementación después del emplazado y rutado. Las conexiones se refieren a la cantidad de cables en la matriz de interconexión y se calculan mediante la ecuación mostrada en (5.7), donde q_λ denota el número de bits de las señales $\lambda_{n,m}$ y q_μ el número de bits de las señales $\mu_{m,n}$. La tasa de decodificación se calcula usando la ecuación 5.6, siendo f la inversa del camino crítico y c igual a cero, debido a que las unidades se han implementado sin segmentación. Para el cálculo de la tasa de decodificación se ha usado un número de iteraciones, It, igual a 15. Con este valor se consigue una buena relación entre tasa de decodificación y prestaciones, ya que las pérdidas, respecto al uso de 30 iteraciones, son inferiores a 0.1 dB.

$$C_{FP} = M \cdot \rho_m \cdot q_\mu + N \cdot \gamma_n \cdot q_\lambda \tag{5.7}$$

En la Tabla 5.4 se aprecia que la implementación de la arquitectura completamente paralela con el algoritmo SP es la que mayor área requiere y menor tasa de decodificación

Tabla 5.4: Resultados de implementación en un ASIC de la arquitectura paralela con los algoritmos SP y basados en MS usando la librería Faraday de 90nm, para el código RS(6,32,6) del estándar IEEE 802.3an. La tasa de decodificación es calculada para un número de iteraciones *It* igual a 15.

Algoritmo	**Área** [mm^2]	**Camino crítico** [ns]	**Conexiones**	**Tasa de decodificación** [Gbps]
SP	19.45	11.72	147456	5.82
MS	17.35	9.07	147456	7.52
αMS	17.49	9.27	147456	7.36
βMS	17.55	9.43	147456	7.23
iMS	11.47	7.22	122880	9.45
αiMS	11.50	7.24	122880	9.43
vwMS	14.52	8.98	147456	7.60

obtiene. Aunque el número de conexiones en las implementaciones con los algoritmos MS, αMS y βMS es igual que con el algoritmo SP, éstas necesitan un 10 % menos de área y consiguen una tasa de decodificación 22 % mayor. La implementación de la arquitectura con el algoritmo propuesto vwMS consigue una tasa de decodificación similar a la de la implementación con el algoritmo MS, sin embargo, su área es un 16 % menor. Destaca la reducción en el área alcanzada por las implementaciones con los algoritmos iMS y αiMS, cuyo número de conexiones es menor. El área de éstas es un 34 % menor, comparado con el algoritmo MS, y alcanza una tasa de decodificación 20 % superior. En general, las tasas de decodificación alcanzadas por estas implementaciones son altas, sin embargo, el coste *hardware* de las implementaciones es elevado, lo que las hace poco populares.

5.3. Arquitectura Parcialmente paralela basada en memorias

En la arquitectura parcialmente paralela basada en memorias se explota la regularidad de los códigos estructurados, por lo que su uso se limitada a éste tipo de códigos. Sin embargo, los códigos estructurados se encuentran en una gran variedad de aplicaciones y se incluyen en varios estándares de comunicaciones. Las implementaciones *hardware* con esta arquitectura [39–43, 45] consiguen una buena relación entre complejidad y tasa de decodificación. Las tasas de decodificación que se alcanzan son del orden de cientos de Mbps, suficientes para varias aplicaciones como por ejemplo Wi-Fi (IEEE 802.11n [7]), WiMAX (IEEE 802.16e [8]) o DVB-S2 [5]. Algunas de estas implementaciones se ha realizado en dispositivos FPGA [39, 40, 45], alcanzando tasas de decodificación de cientos de Mbps.

Para un código estructurado cuya matriz **H** está compuesta por $M_b \times N_b$ submatrices de tamaño $z \times z$ se requieren M_b unidades CNP y N_b unidades VNP. Cada unidad CNP está asociada a un grupo de z filas consecutivas de la matriz **H**, correspondientes a una fila de submatrices. De manera análoga, cada unidad VNP está asociada a un grupo de z columnas consecutivas de **H**, correspondientes a una columna de submatrices. En cada iteración, primero las N_b unidades VNP procesan de forma secuencial las N columnas de la matriz de paridad, por lo que cada unidad procesa z columnas consecutivas en z ciclos de reloj. Después, las M_b unidades CNP procesan de forma secuencial las M columnas, de manera que cada unidad CNP procesa z columnas consecutivas en otros z ciclos de reloj. Los mensajes generados durante la decodificación por las unidades VNP y CNP son

almacenados en memoria. Cada submatriz no nula tiene asociado un par de memorias cuya profundidad es igual a z, una para los mensajes $\lambda_{n,m}$ y otra para los mensajes $\mu_{m,n}$. Para un código regular en el que no existen submatrices nulas, el número total de memorias es $2 \cdot M_b \cdot N_b$. El direccionamiento de las memorias está determinado por las posiciones de los unos en la matriz de paridad, de manera que los mensajes intercambiados entre unidades tengan el mismo origen y destino que las conexiones en el grafo de Tanner. El cálculo de las ecuaciones de paridad se realiza de forma secuencial con los bits decodificados z_n para cada bloque de z filas, correspondiente a una de las N_b submatrices. Se verifican en paralelo N_b ecuaciones de paridad, usando puertas XOR, durante los $2 \cdot z$ ciclos que dura una iteración. De esta manera se evalúan las ecuaciones de paridad correspondientes a las $z \cdot N_b$ filas de la matriz **H**. En cada ciclo se comprueba que se cumple la paridad de las N_b ecuaciones, usando una puerta AND. Si el resultado de la AND es 1 en todos los z ciclos, la paridad es válida y por tanto, se puede finalizar anticipadamente.

La Figura 5.11 muestra la arquitectura parcialmente paralela basada en memorias para el código RS(3,6,3). La matriz de paridad de este código (Figura 5.1) está compuesta por 3 filas y 6 columnas de submatrices cuyo tamaño es 8×8. Por lo tanto, el decodificador requiere 3 unidades CNP, 6 unidades VNP y 36 memorias de 8 posiciones. En el proceso de decodificación, las memorias $L_{m,n}$ son leídas por las unidades CNP y escritas por las unidades VNP de forma secuencial, mientras que las memorias $M_{m,n}$ son escritas por las unidades CNP y leídas por las unidades VNP. Las direcciones de lectura de las memorias $L_{m,n}$ corresponden con los índices de columna de los unos en cada submatriz, mientras que las de escritura son ordenadas de 1 a z. Para las memorias $M_{m,n}$ las direcciones de lectura corresponden con los índices de fila y las de escritura son ordenadas entre 1 y z. Por ejemplo, la unidad VNP_1 lee secuencialmente las posiciones 1, 6, 7, 8, 2, 3, 4 y 5 de la memoria $M_{1,1}$ y escribe en posiciones consecutivas y ordenadas, 1 a 8, de la memoria $L_{1,1}$. En el caso de la CNP_1, ésta lee las posiciones 1, 5, 6, 7, 8, 2, 3 y 4 de la memoria $L_{1,1}$ y escribe de forma ordenada en $M_{1,1}$. Las señales $l_{n..m}$ corresponden a bloques de información *a priori* del canal que entran al decodificador y los bloques de señales $z_{n..m}$ a bits decodificados. Las unidades VNP y CNP son iguales que las expuestas en la arquitectura completamente paralela (Sección 5.2), Figuras 5.9 y 5.10.

La ecuación 5.8 muestra la tasa de decodificación para la arquitectura parcialmente paralela, donde N es la longitud del código, f la frecuencia de reloj, z el número de filas o columnas de las submatrices e It el número de iteraciones. El factor c está determinado por el grado de segmentación de las unidades y a diferencia de la arquitectura paralela, el aumento en la frecuencia de reloj es notable cuando el grado de segmentación aumenta. Además, en los códigos usados en aplicaciones reales el factor c es mucho menor que z, por lo tanto, el numerador de la ecuación aumenta en mayor proporción que el denominador a medida que aumenta el grado de segmentación. En conclusión, el grado de segmentación de las unidades debe ser máximo para obtener un camino crítico bajo y así conseguir la mayor tasa de decodificación.

$$T_{PP} = \frac{N \cdot f}{It \cdot (2 \cdot z + c)} \tag{5.8}$$

El número de ciclos de reloj que requiere una iteración es igual a $2 \cdot z + c$ y al igual que en la arquitectura paralela, mientras las unidades VNP procesan las filas de la matriz las unidades CNP están en estado de espera y *viceversa*. Si se asume el mismo grado de segmentación en las unidades VNP y CNP, el número total de ciclos de reloj que requiere

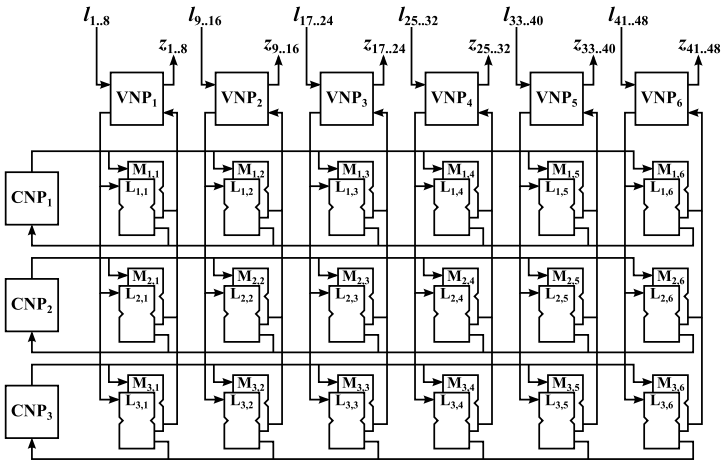


Figura 5.11: Arquitectura parcialmente paralela basada en memorias para el código RS(3,6,3). Consta de 3 unidades CNP, 6 unidades VNP y 36 memorias de profundidad 8 para el intercambio de mensajes.

cada unidad para procesar es igual a $z + c/2$. Esto implica que las unidades están en estado de espera la mitad de los ciclos de reloj que dura una iteración, por consiguiente, la HUE es del 50 %.

5.3.1. Resultados de implementación

Los algoritmos expuestos en los capítulos 3 y 4 se han implementado con la arquitectura parcialmente paralela en un ASIC y en un dispositivo FPGA para el código RS(6,32,6) del estándar IEEE 802.3an. Este código es estructurado regular y su matriz de paridad **H** está compuesta por 6 filas y 32 columnas de submatrices, cuyo tamaño es 64×64. Como se explicó anteriormente, la mayor tasa de decodificación se consigue cuando el nivel de segmentación de las unidades es máximo, por lo tanto, las unidades VNP y CNP están completamente segmentadas. En las implementaciones *hardware* se añaden N_b memorias de profundidad z para almacenar los mensajes de entrada l_n, los cuales son utilizados por las unidades VNP en cada iteración del proceso de decodificación. Además, se incluye el control del decodificador encargado de generar las direcciones de escritura y lectura de las diferentes memorias, las señales de habilitación y la gestión de la finalización anticipada.

Los factores de corrección y escalado de los algoritmos implementados, así como los esquemas de cuantificación, son los mismos que los utilizados en las implementaciones con la arquitectura completamente paralela. Para los algoritmos αMS, αiMS y vwMS se utiliza un factor α igual a 0.75, igual que el factor β del algoritmo βMS. El vector de factores de corrección w para el algoritmo vwMS es $[1.25; 1.75; 2.25; 2.75]$ y el de umbrales $u = [5; 10; 15]$. En los algoritmos SP, MS, αMS, βMS y vwMS, los mensajes l_n, $\lambda_{n,m}$ y $\mu_{m,n}$ se cuantifican con un esquema $[6 : 2]$. En los algoritmos iMS y αiMS se cuantifican las señales l_n y $\lambda_{n,m}$ con un esquema $[6 : 2]$ y las señales $\mu_{m,n}$ con un esquema $[4 : 0]$. Por otro lado, dentro de las unidades VNP la suma crece 4 bits. En las unidades CNP de

Tabla 5.5: Resultados de implementación de la arquitectura parcialmente paralela basada en memorias con los algoritmos SP y basados en MS para el código RS(6,32,6) del estándar IEEE 802.3an, en un dispositivo FPGA Cyclone IV. La tasa de decodificación es calculada para un número de iteraciones It igual a 15.

Algoritmo	**Área**		**Camino crítico**	**Almacenamiento**	**Tasa de decodificación**
	[LE]	[M9K]	[ns]	[kb]	[Mbps]
SP	24993	324	5.44	148	173.1
MS	18426	313	5.34	148	178.8
αMS	18713	314	5.38	148	176.2
βMS	18713	314	5.43	148	174.6
iMS	14304	291	5.32	123	179.5
αiMS	14603	292	5.38	123	176.2
vwMS	16798	312	5.42	148	174.9

la implementación con el algoritmo SP se cuantifican las funciones $\Psi(x)$ y $\Psi(x)^{-1}$ con los esquemas $[9:7]$ y $[6:2]$, respectivamente. Las sumas internas crecen 5 bits ($[14:7]$) y luego se truncan y saturan para adaptarlas al esquema $[6:4]$.

Los resultados de implementación en un ASIC con la librería Faraday de 90nm y en un dispositivo FPGA Cyclone IV de Altera se muestran en las Tablas 5.5 y 5.6, respectivamente. El porcentaje de utilización en las implementaciones sobre un ASIC es del 80 %. En este caso la congestión de las conexiones es baja por lo que un 20 % de espacio para el rutado es suficiente. Además, con este valor se obtiene la mejor relación entre área y camino crítico. Los datos de área y camino crítico para la implementación en ASIC son después del emplazado y rutado. Cabe destacar que en el ASIC las memorias se han implementado de manera distribuida. En el caso de la implementación en FPGA los resultados de área se dan en términos del número de elementos lógicos (LEs) y número de memorias embebidas M9K. El almacenamiento se refiere a la cantidad total de memoria requerida, la cual se calcula de acuerdo a la ecuación 5.9, donde q_λ denota el número de bits de las señales $\lambda_{n,m}$ y q_μ el número de bits de las señales $\mu_{m,n}$. Para el cálculo de la tasa de decodificación se utiliza (5.8), donde la frecuencia de reloj f se calcula como la inversa del camino crítico. Debido a la diferencia que hay entre las unidades CNP de los diferentes algoritmos, su nivel de segmentación es diferente para cada uno y por consiguiente, el factor c. Para los algoritmos MS e iMS el factor c es igual a 15, para los algoritmos αMS, βMS, αiMS y vwMS es igual a 16 y para el algoritmo SP es igual a 17. Al igual que en la arquitectura completamente paralela, el número de iteraciones (It) para el cálculo de la tasa de decodificación es igual a 15, con el fin de conseguir una buena relación entre tasa de decodificación y prestaciones.

$$C_{FP} = M_b \cdot N_b \cdot z \cdot (q_\mu + q_\lambda) \tag{5.9}$$

En los resultados de las Tablas 5.5 y 5.6 se puede ver que el camino crítico es similar para las implementaciones de los diferentes algoritmos y por consiguiente, las tasas de decodificación también se parecen. Sin embargo, en el área sí que se aprecian diferencias significativas. Al igual que con la arquitectura completamente paralela, la implementación con el algoritmo SP es la que más área requiere, siendo un 26 % y 10 % mayor que el área de la implementación con el algoritmo MS en FPGA y ASIC, respectivamente. Se aprecia también que con los algoritmos αMS y βMS hay un incremento en el área del 1.5 % respecto a la implementación del algoritmo MS en FPGA, mientras que en ASIC

Tabla 5.6: Resultados de implementación en un ASIC de la arquitectura parcialmente paralela basada en memorias con los algoritmos SP y basados en MS usando la librería Faraday de 90nm, para el código RS(6,32,6) del estándar IEEE 802.3an. La tasa de decodificación es calculada para $It = 15$.

Algoritmo	**Área** [mm^2]	**Camino crítico** [ns]	**Almacenamiento** [kb]	**Tasa de decodificación** [Mbps]
SP	5.35	2.78	148	338.7
MS	4.84	2.72	148	351.0
αMS	4.88	2.72	148	348.6
βMS	4.91	2.74	148	346.0
iMS	3.58	2.70	123	353.6
αiMS	3.62	2.72	123	348.6
vwMS	4.42	2.70	148	351.2

el algoritmo αMS requiere tan sólo un 0.8 % más de área y el algoritmo βMS un 1.4 %. Con el algoritmo propuesto vwMS se consigue una reducción en el área del 8 % en ASIC y FPGA. Más significativa es la reducción con los algoritmos iMS y αiMS, superior al 20 % en FPGA y al 25 % en ASIC, debida principalmente a que se requiere menores recursos de almacenamiento (registros y/o memorias).

Como se ha comentado anteriormente, las tasas de decodificación conseguidas con esta arquitectura son relativamente bajas, un orden de magnitud menor que las conseguidas con la arquitectura completamente paralela (Tabla 5.4). Sin embargo, el área que requieren las implementaciones es reducida. Las implementaciones con los algoritmos SP, MS, αMS y βMS requieren un 72 % menos de área, mientras que las implementaciones con los algoritmos iMS, αiMS y vwMS consiguen un ahorro del 69 %.

5.4. Arquitectura SMP mejorada (ISMP)

En la arquitectura SMP (*sliced message passing*), propuesta en [38], se divide la matriz de paridad en bloques de columnas que se procesan de forma secuencial. Esta división permite reducir el número de unidades VNP, comparado con la arquitectura completamente paralela. Además, se reduce la complejidad de las unidades CNP debido a que la actualización horizontal (unidades CNP) se realiza de forma recursiva. La arquitectura SMP es óptima cuando es implementada con códigos estructurados y consigue altas tasas de decodificación, del orden de Gbps, con una complejidad considerablemente menor a la de las implementaciones con la arquitectura completamente paralela (Sección 5.2). En esta tesis se ha propuesto una mejora de la arquitectura SMP, la cual consiste en la optimización de las unidades CNP aprovechando la regularidad de los códigos LDPC estructurados. La arquitectura propuesta se ha denominado ISMP (*improved*-SMP) [55] y consigue una reducción de área de un 33 % cuando es implementada con el algoritmo SP. La arquitectura SMP, en [38], solamente se implementa para el algoritmo SP. Por consiguiente, la arquitectura ISMP solamente se ha implementado para este algoritmo a manera de comparación. Sin embargo, ambas arquitecturas también son válidas para los algoritmo basados en MS, aprovechado la reducción en la complejidad de éstos.

El número de bloques en el que se divide la matriz de paridad del código estructurado

se denomina p y debe ser un valor entero tal que el resto de la división N_b/p sea cero. Cada bloque se denomina *slice* y su tamaño es de $M_b \times N_b/p$. Tanto en la arquitectura SMP como en la ISMP se requieren $z \cdot N_b/p$ unidades VNP que procesan las columnas de los *slices* y $M = z \cdot M_b$ unidades CNP que procesan las filas. El proceso de actualización vertical (unidades VNP) se realiza de forma secuencial por *slices* y el proceso de actualización horizontal (unidades CNP) se ejecuta de forma recursiva. Cada iteración i del proceso de decodificación se realiza en $p+1$ ciclos de reloj. En el primer ciclo, las unidades VNP procesan en paralelo las columnas del *slice* 1 utilizando los mensajes generados por las unidades CNP en la iteración anterior $(i-1)$. En la primera iteración los valores de estos mensajes son cero. Posteriormente, en los ciclos 2 a p, las unidades CNP procesan de manera recursiva las M filas del *slice* correspondiente utilizando los mensajes generados por las unidades VNP en el ciclo anterior y, al mismo tiempo, las unidades VNP procesan las columnas. En el último ciclo, $p+1$, las unidades CNP terminan de calcular los mensajes $\mu_{m,n}$ utilizando todos los resultados parciales. Los mensajes calculados por las unidades CNP en el último ciclo de la iteración i son los utilizados por las VNP en la iteración $i+1$. El cálculo de las ecuaciones de paridad se realiza de forma recursiva, al igual que la actualización de los nodos de comprobación. En cada *slice* se calculan parcialmente las M ecuaciones de paridad realizando una operación XOR con lo bits z_n correspondientes. Los resultados parciales se acumulan y en el último ciclo, se realiza una operación AND con los M resultados acumulados para evaluar si la paridad se cumple.

Al igual que en la arquitectura paralela, los mensajes se almacenan en registros debido a que la lectura y escritura de éstos se realiza de manera concurrente. Las conexiones entre unidades se realizan a través de multiplexores que seleccionan los registros apropiados en cada ciclo, tanto para las unidades VNP como para las unidades CNP. La selección de los multiplexores se realiza de acuerdo a las posiciones de los unos en la matriz de paridad, de manera que los mensajes intercambiados entre unidades tengan el mismo origen y destino que las conexiones en el grafo de Tanner. El número de unidades CNP conectadas a una unidad VNP es el mismo que en la arquitectura completamente paralela, sin embargo, el número de unidades VNP conectadas a una unidad CNP es p veces menor. Por lo tanto, el número de unidades CNP conectadas a una unidad VNP es igual a γ_n y el número de unidades VNP conectadas a una unidad CNP es ρ_m/p.

La Figura 5.12 muestra la estructura común a las arquitectura SMP e ISMP para el código RS(3,6,3) con un número de *slices* $p = 3$. Cada *slice* corresponde con 16 columnas de la matriz de paridad de la Figura 5.1, por lo que el número de unidades VNP que se necesitan es 16. El número de unidades CNP en este caso es 24 debido a que la matriz $\mathbf{H}$ tiene 24 filas. Los registros LR, que almacenan los mensajes $\lambda_{n,m}$ generados por las unidades VNP, se actualizan en cada uno de los $p+1$ ciclos de reloj que dura una iteración. En la arquitectura ISMP, los registros que almacenan los mensajes $\mu_{m,n}$ generados por las unidades CNP también se actualizan en cada ciclo, sin embargo, en la arquitectura SMP éstos solamente se actualizan en el ciclo $p+1$. Los multiplexores MUX_L, asociados a las unidades VNP, se encargan de seleccionar las conexiones (bloque "Matriz de Interconexión") con las unidades CNP en cada ciclo. La selección se hace de acuerdo a la posición de los unos en la columna correspondiente de la matriz $\mathbf{H}$. Por ejemplo, en el primer ciclo el MUX asociado a la VNP_1 selecciona las conexiones con las unidades CNP_1, CNP_{10} y CNP_{19}, mientras que en el segundo ciclo selecciona las conexiones con las unidades CNP_1, CNP_{14} y CNP_{23}. En el tercer ciclo las conexiones de la VNP_1 se realizan con las unidades CNP_1, CNP_{16} y CNP_{19}. Por otro lado, los multiplexores asociados a las unidades CNP, denominados MUX_M, realizan

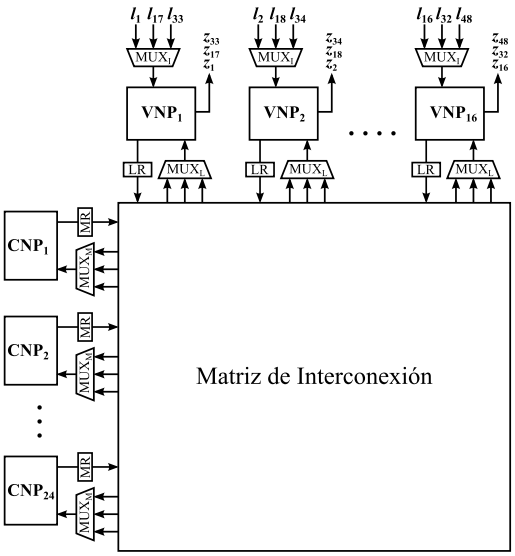


Figura 5.12: Arquitectura SMP e ISMP para el código RS(3,6,3) y un número de *slices* $p = 3$. Consta de 16 unidades VNP y 24 unidades CNP que se conectan entre si a través de los multiplexores (MUX_L y MUX_M). En los resgistros MR y LR corresponden se almacenan los mensajes y los multiplexores MUX_I seleccionan la información *a priori* de canal.

la selección de acuerdo a la posición relativa de los unos en la fila correspondiente de los *slices*. Por ejemplo, el MUX asociado a la unidad CNP_2 selecciona las unidades VNP_5 y VNP_{10} en el ciclo 2 (primer *slice*), las unidades VNP_5 y VNP_{11} en el ciclo 3 (segundo *slice*) y las unidades VNP_3 y VNP_{10} en el ciclo 4 (tercer *slice*). La información *a priori* del canal, l_n, correspondiente a cada bloque, se selecciona a través de los multiplexores MUX_I. Los bits decodificados, z_n, salen en bloques de tamaño N/p bits durante los ciclos 1 a p. Para este ejemplo, en el ciclo 1 la salida es el bloque de bits z_1 a z_{16}, en el ciclo 2 es el bloque z_{17} a z_{32} y en el ciclo 3 la salida corresponde al bloque de bits z_{33} a z_{48}.

Las unidades VNP, para las arquitecturas SMP e ISMP, son iguales que en la arquitectura completamente paralela (Sección 5.2) y su estructura se muestra en la Figura 5.9. En cambio, las arquitecturas de las unidades CNP son diferentes debido básicamente al procesado recursivo que éstas realizan. Sea $N_m[c]$ el subconjunto de nodos de bit adyacentes a un nodo de comprobación C_m para el *slice* c, donde $c \in \{1,\ldots,p\}$. Gracias a las características de la matriz de paridad de los códigos estructurados, los subconjuntos $N_m[c]$ cumplen las propiedades descritas en (5.10) y (5.11).

$$N_m[0] \cup N_m[1] \cup N_m[2] \cup \ldots \cup N_m[p-1] = N_m \tag{5.10}$$

$$N_m[0] \cap N_m[1] \cap N_m[2] \cap \ldots \cap N_m[p-1] = \emptyset \tag{5.11}$$

Por otra parte, se define υ como el argumento de la función $\Psi^{-1}(x)$ en la actualización de los nodos de comprobación del algoritmo SP (ecuación 3.1), de manera que:

$$\upsilon_{m,n}^{(i-1)} = \sum_{n' \in N_{m \setminus n}} \Psi(|\lambda_{n',m}^{(i-1)}|) \tag{5.12}$$

El sumatorio sobre el conjunto N_m excluyendo el elemento n de (5.12) se puede escribir como el sumatorio sobre todos los elementos del conjunto N_m menos $\Psi(|\lambda_{n',m}^{(i-1)}|)$, evaluado para el elemento $n' = n$.

$$\upsilon_{m,n}^{(i-1)} = \sum_{n' \in N_m} \Psi(|\lambda_{n',m}^{(i-1)}|) - \Psi(|\lambda_{n,m}^{(i-1)}|) \tag{5.13}$$

Gracias a las propiedades de los subconjuntos $N_m[c]$, descritas en (5.10) y (5.11), el sumatorio se puede dividir en una suma de sumatorios para cada subconjunto. Por lo tanto, (5.13) se puede expresar en términos de $N_m[c]$ como se muestra a continuación.

$$\begin{aligned}
\upsilon_{m,n}^{(i-1)} &= \sum_{n' \in N_m[1]} \Psi(|\lambda_{n',m}^{(i-1)}|) + \cdots + \sum_{n' \in N_m[p]} \Psi(|\lambda_{n',m}^{(i-1)}|) - \Psi(|\lambda_{n,m}^{(i-1)}|) \\
\upsilon_{m,n}^{(i-1)} &= \sum_{c \in \{1..p\}} \sum_{n' \in N_m[c]} \Psi(|\lambda_{n',m}^{(i-1)}|) - \Psi(|\lambda_{n,m}^{(i-1)}|) \\
\upsilon_{m,n}^{(i-1)} &= \sum_{c \in \{1..p\}} \Upsilon_c - \Psi(|\lambda_{n,m}^{(i-1)}|) \\
\text{donde } \Upsilon_c &= \sum_{n' \in N_m[c]} \Psi(|\lambda_{n',m}^{(i-1)}|)
\end{aligned} \tag{5.14}$$

Se puede ver en (5.14) que el cálculo de $\upsilon_{m,n}^{(i-1)}$ se hace sumando los resultados parciales de cada *slice*, representados por la función Υ_c, y restando $\Psi(|\lambda_{n,m}^{(i-1)}|)$. En las arquitecturas SMP e ISMP esta suma se realiza de manera recursiva, acumulando los resultados parciales de cada *slice* y realizando al final la resta. El análisis anterior es también válido para el cálculo de los signos (actualización de paridad $\Gamma_{m,n}^{(i)}$). Por consiguiente, la actualización de paridad también se realiza de manera recursiva, de forma que los resultados parciales (por *slice*) se acumulan y en el último ciclo $(p+1)$ se obtiene la actualización de paridad total. El proceso de decodificación iterativo del algoritmo SP adaptado a las arquitecturas SMP e ISMP se muestra en el pseudocódigo 5.1. Cabe destacar que esta adaptación no modifica de ninguna manera al algoritmo original, por lo tanto, sus prestaciones son las mismas.

La estructura de la unidad CNP de la arquitectura SMP para el código RS(3,6,3) se muestra en la Figura 5.13(a). La magnitud de las entradas $\lambda_{n,m}^{(c)}$, correspondientes al *slice c*, se usan para direccionar las tablas PLUT, computando de esta manera la función $\Psi(x)$. Las salidas de las tablas se suman y el resultado es acumulado en el registro AREG. En el ciclo 1 de cada iteración el valor de este registro es iniciado a cero. El resultado total se obtiene en el ciclo $p+1$ y corresponde al valor de Υ'_m en la línea 15 del pseudocódigo 5.1. Las salidas de las tablas PLUT también son almacenadas en los registros SREG con el fin de evaluar la resta en el argumento de la función $\Psi^{-1}(x)$ (línea 19 del pseudocódigo). Nótese que esta operación y las siguientes se realizan en paralelo para los mensajes de todos los *slices*. Por lo tanto, en el ciclo $p+1$ se obtienen todos los mensajes $\mu_{m,n}$, que serán utilizados por las unidades VNP en la siguiente iteración. El hecho de realizar el cálculo en paralelo para

Pseudocódigo 5.1 Algoritmo SP adaptado a las arquitecturas SMP e ISMP.

1: – *inicialización* –
2: para $m \in \{1,\ldots,M\}$ y $n \in N_m$
3: $\quad \mu_{m,n}^{(0)} = 0$
4: – *iteraciones* –
5: para $i \in \{1,\ldots,I_{max}\}$
6: $\quad$ – *slices* –
7: $\quad \Upsilon'_m = 0$ y $\Gamma'_m = 1$, $\forall m \in \{1,\ldots,M\}$
8: $\quad$ para $c \in \{1,\ldots,p\}$
9: $\quad\quad$ – *actualización de los nodos de bit (slice)*–
10: $\quad\quad$ para $n \in \{(c-1)\cdot N/p+1,\ldots,c\cdot N/p\}$ y $m \in M_n$
11: $\quad\quad\quad \lambda_n^{(i)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(i-1)}$
12: $\quad\quad\quad \lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \mu_{m,n}^{(i-1)}$
13: $\quad\quad$ – *actualización parcial de los nodos de comprobación* –
14: $\quad\quad$ para $m \in \{1,\ldots,M\}$ y $n \in N_m[c]$
15: $\quad\quad\quad \Upsilon'_m = \Upsilon'_m + \Upsilon_c$, donde $\Upsilon_c = \sum_{n' \in N_m[c]} \Psi(|\lambda_{n',m}^{(i)}|)$
16: $\quad\quad\quad \Gamma'_m = \Gamma'_m \cdot \Gamma_c$, donde $\Gamma_c = \prod_{n' \in N_m[c]} \text{sign}(\lambda_{n',m}^{(i)})$
17: $\quad$ – *actualización de los nodos de comprobación* –
18: $\quad$ para $m \in \{1,\ldots,M\}$ y $n \in N_m$
19: $\quad\quad \mu_{m,n}^{(i)} = \Gamma'_m \cdot \text{sign}(\lambda_{n,m}^{(i)}) \cdot \Psi^{-1}\left(\Upsilon'_m - \Psi(|\lambda_{n,m}^{(i)}|)\right)$

todos los mensajes $\mu_{m,n}$, implica que son necesarios $\rho_m = 6$ sumas, 6 bloques de saturación y 6 tablas RLUT. Estas últimas se usan para evaluar la función $\Psi^{-1}(x)$ de la línea 19 del pseudocódigo. Al igual que en la arquitectura completamente paralela, los datos crecen de forma natural y en el bloque SAT se aplica saturación y truncado para limitar el tamaño de las tablas RLUT. En la figura se puede observar que el cálculo de los signos es similar al de las magnitudes, donde, en el ciclo 1 se inicia el registro AREG a cero y en los ciclos 2 a $p+1$ acumula el resultado de la XOR. En el ciclo $p+1$ se realiza una XOR del resultado total de la acumulación con los valores de los signos de cada entrada, almacenados en los registros SREG.

En la actualización de los nodos de bit del pseudocódigo 5.1 se observa que el cálculo de los mensajes λ_n y $\lambda_{n,m}$ se realiza para el conjunto $n \in \{(c-1)\cdot N/p+1,\ldots,c\cdot N/p\}$. Esto implica que para realizar la actualización de los nodos de bit son necesarios únicamente los valores de $\mu_{m,n}$ correspondientes al *slice c* de la iteración anterior, $i-1$. Esto se aprovecha en la arquitectura propuesta ISMP, donde el número de salidas de las unidades CNP se reducen en un factor p y la generación de los mensajes se hace de forma secuencial por *slices*. Esta reducción implica un decremento en la complejidad computacional de las unidades CNP, ya que también se reducen las operaciones de resta y saturación, así como el número de tablas RLUT requeridas. La estructura de las unidades CNP de la arquitectura ISMP para el código RS(3,6,3) se muestra en la Figura 5.13(b). Al igual que en la arquitectura SMP, las salidas de las tablas PLUT son acumuladas durante los ciclos 2 a $p+1$ en el registro AREG y a la vez, almacenadas en los registros SREG. Con el fin de mantener el valor total de la acumulación para la siguiente iteración se utiliza el registro HREG y el multiplexor HMUX. Para esto, el multiplexor selecciona el valor del registro en los ciclos 1 a p y la salida del acumulador en el ciclo $p+1$, actualizando así el valor del registro. Con este valor y los valores almacenados en los registros SREG se genera de forma secuencial por *slices*

los mensajes $\mu_{m,n}$. El cálculo de los signos se hace de forma similar, el resultado total del acumulador de XOR se mantiene constante durante los ciclos 1 a p de la iteración siguiente, utilizando el registro HREG y el multiplexor HMUX. Por último, se calculan los signos de salida utilizando los signos de las señales de entrada, almacenados en los registros SREG y el valor almacenado en el registro HREG.

Comparando las Figuras 5.13(a) y 5.13(b) se aprecia que la reducción de la complejidad *hardware* de la unidad CNP de la arquitectura ISMP es considerable, frente a la unidad CNP de la arquitectura SMP. Para el código RS(3,6,3), el número total de sumadores es un 50 % menor, mientras que la reducción de los bloques de saturación y de las tablas RLUT es del 77 %. Cabe señalar que esta reducción puede ser mayor para códigos de mayor longitud y cuya matriz de paridad es de mayor dimensión. Los únicos elementos adicionales en la unidad CNP de la arquitectura ISMP son el multiplexor HMUX y el registro HREG, cuya complejidad *hardware* es despreciable respecto a la del resto de la unidad. Por otro lado, en esta arquitecturas las unidades CNP representan un alto porcentaje de la complejidad *hardware* total (superior al 50 %), por lo que una reducción en la complejidad de éstas se refleja directamente en la complejidad del decodificador.

La tasa de decodificación para las arquitecturas SMP e ISMP depende del número de *slices* p, como se muestra en (5.15). El factor N corresponde con la longitud del código, f con la frecuencia de reloj e It con el número de iteraciones.

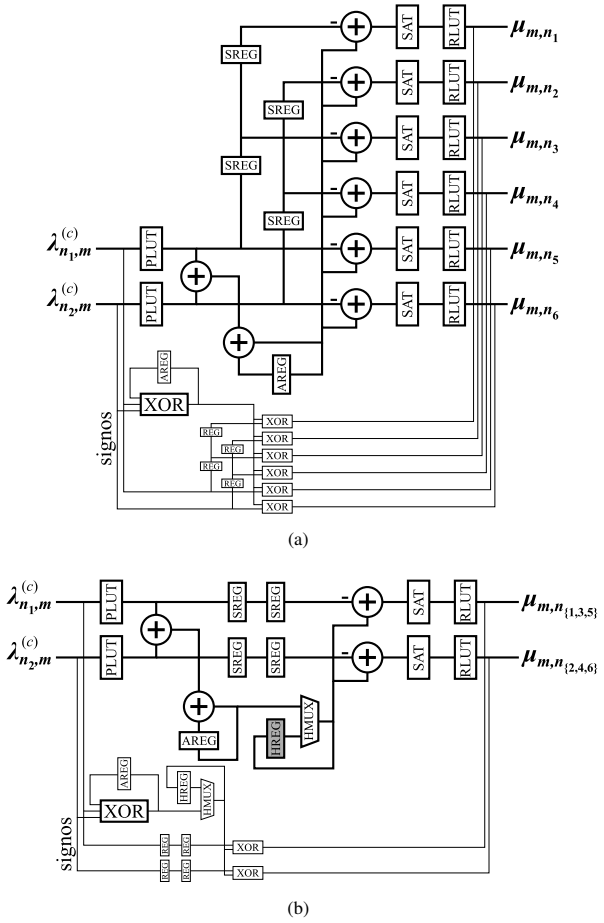
$$T_{SMP} = \frac{N \cdot f}{It \cdot (p+1)} \tag{5.15}$$

Una característica importante de estas arquitecturas es la alta HUE de sus implementaciones, ya que los ciclos de espera de las unidades son bajos respecto a los ciclos de procesado. Las unidades VNP procesan durante los ciclos 1 a p y permanecen en espera en el ciclo $p+1$, mientras que las unidades CNP procesan en los ciclos 2 a $p+1$ y están en estado de espera en el ciclo 1. Por lo tanto, la HUE con esta arquitectura es igual a $p/(p+1)$. Para un número de *slices* $p = 4$ la HUE es del 80 %, un 30 % mayor que la conseguida con las arquitecturas completamente paralela o parcialmente paralela basada en memorias.

5.4.1. Resultados de implementación

La arquitectura ISMP se ha implementado en un ASIC para el mismo código y número de *slices* que la arquitectura SMP presentada en [38]. Además, se ha implementado la arquitectura SMP con el fin de realizar una comparación justa, utilizando la misma librería ASIC y las mismas herramientas de síntesis y rutado. En ambas arquitecturas, SMP e ISMP, se utiliza el algoritmo SP y un número de *slices* igual a 4. El código utilizado es el RS(6,32,6) del estándar IEEE 802.3an, cuya matriz de paridad **H** está compuesta por 6 filas y 32 columnas de submatrices, cuyo tamaño es 64×64. Por otro lado, las implementaciones cuentan con *hardware* adicional, registros y multiplexores, para mantener los datos de entrada l_n estables durante todas las iteraciones del proceso de decodificación. En éstas también se incluye el *hardware* de control, encargado de generar las señales de habilitación para las unidades CNP y VNP en los ciclos correspondientes y las señales para iniciar los registros usados en la recursión. Además, el control se encarga de la gestión de la finalización anticipada.

Los mensajes l_n, $\lambda_{n,m}$ y $\mu_{m,n}$ se cuantifican con un esquema $[6:2]$, para así conseguir una buena relación entre prestaciones y número de bits, como se ha visto en el análisis



(a)

(b)

Figura 5.13: Arquitectura de la unidad CNP para el código RS(3,6,3) y número de *slices* $p = 3$. La unidad calcula de forma recursiva los mensajes $\mu_{m,n}$ acumulando los resultados parciales de cada *slice*. (a) Unidad CNP de la arquitectura SMP. (b) Unidad CNP de la arquitectura ISMP.

Tabla 5.7: Resultados de implementación en un ASIC de las arquitecturas SMP e ISMP con el algoritmo SP para el código RS(6,32,6) del estándar IEEE 802.3an, usando la librería Faraday de 90nm. El número de *slices* es 4 e It es igual a 15.

Arquitectura	**Área** [mm^2]	**Camino crítico** [ns]	**Tasa de decodificación** [Gbps]
SMP	14.85	4.96	5.50
ISMP	9.95	4.82	5.66

realizado en la la Sección 3.4. Para la función no lineal $\Psi(x)$ se utiliza un esquema de cuantificación $[9:7]$. Con el crecimiento natural de los datos (suma de 32 elementos) el acumulador requiere un esquema $[15:7]$. Este mismo esquema se usa en las restas, cuyos resultados se saturan y trucan a un esquema $[6:4]$. El mismo esquema de los mensajes $\mu_{m,n}$ es utilizado para la función $\Psi(x)^{-1}$.

Los resultados de implementación con la librería Faraday de 90nm se muestran en la Tabla 5.7. Aunque la congestión debida a las conexiones es menor en estas arquitecturas, comparada con la arquitectura completamente paralela, ésta sigue siendo alta. La mejor relación entre área y camino crítico se consigue con un porcentaje de utilización del 65 % en ambos diseños. Los resultados de área y camino crítico mostrados son los obtenidos después del emplazado y rutado. La tasa de decodificación se calcula usando la ecuación 5.15, siendo f la inversa del camino crítico. Al igual que para las arquitecturas anteriores, para el cálculo de la tasa de decodificación se ha usado un número de iteraciones, It, igual a 15.

En los resultados de la Tabla 5.7 se observa que el camino crítico y la tasa de decodificación de ambas implementaciones son similares, sin embargo, con la arquitectura SMP se consigue una reducción en el área de un 33 %. Por otro lado, la tasa de decodificación que alcanza la arquitectura ISMP es solamente un 3 % menor que la conseguida con la arquitectura completamente paralela, pero el ahorro en área es superior al 48 %.

5.5. Arquitecturas con actualización vertical *Shuffled* y *x-Shuffled*

En esta tesis se han desarrollado dos arquitecturas para la implementación de los métodos de actualización *shuffled* y *x-shuffled*, explicados en las secciones 3.3.3 y 4.3, respectivamente. Estas arquitecturas son válidas para códigos estructurados y se han enfocado a la alta velocidad, por lo que el nivel de paralelismo que se tiene es alto. Sin embargo, su complejidad *hardware* es mucho menor que la obtenida con la arquitectura completamente paralela. Las arquitecturas *shuffled* y *x-shuffled* son óptimas cuando son implementadas con el algoritmo SP [56] y consiguen mejores resultados, en términos de complejidad *hardware*, que con el algoritmo MS. En [46] y [47] se proponen dos aproximaciones para la implementación del algoritmo MS con arquitecturas similares, las cuales consiguen reducir la complejidad *hardware* a costa de pequeñas pérdidas en las prestaciones.

Las arquitecturas *shuffled* y *x-shuffled* tienen la misma estructura y lo único que las diferencia es el orden en el que se ejecutan las actualizaciones. En ambas, el número de unidades CNP es igual al número de filas de la matriz de paridad y el número de unidades VNP depende del número de grupos, de la misma manera que depende del número de *sli-*

ces en la arquitectura ISMP. Por lo tanto, para una matriz de paridad de tamaño $M \times N$ y un número de grupos G se requieren N/G unidades VNP y M unidades CNP. En la arquitectura *shuffled* cada grupo, que corresponde a una subiteración g, se computa en 2 ciclos de reloj. Por consiguiente, son necesarios $2 \cdot G$ ciclos para completar una iteración. En el primer ciclo de cada subiteración, las unidades VNP procesan en paralelo las columnas del grupo correspondiente (grupo g), utilizando los mensajes generados por las unidades CNP en la subiteración anterior (grupo $g-1$). En la primera subiteración, se utilizan los mensajes generados por las unidades CNP en la última subiteración (G) de la iteración anterior. En el segundo ciclo, las unidades CNP realizan en paralelo la actualización horizontal correspondiente a las M filas de la matriz de paridad, utilizando los mensajes generados por las unidades VNP en las G subiteraciones anteriores. Este cómputo se hace de forma recursiva, de manera similar al que se hace en las unidades CNP de la arquitectura ISMP.

Por otro lado, en la arquitectura *x-shuffled* las actualizaciones de unidades VNP y CNP se realizan al mismo tiempo, siendo necesario un solo ciclo de reloj para computar cada subiteración y G ciclos para completar una iteración. En la subiteración g, las unidades VNP realizan la actualización vertical del grupo correspondiente, utilizando los mensajes generados por las unidades CNP en la subiteración anterior ($g-1$). Al mismo tiempo, las unidades CNP calculan de forma recursiva la actualización horizontal de las M filas de $\mathbf{H}$, utilizando los mensajes generados por las unidades VNP de la subiteración anterior.

El cálculo de las ecuaciones de paridad en ambas arquitecturas se hace de forma recursiva. En cada subiteración g, se calculan parcialmente las M ecuaciones de paridad utilizando los bits decodificados z_n del grupo correspondiente (grupo g). Los M resultados parciales se acumulan y en la última subiteración se realiza una operación AND con éstos. Si el resultados de la AND es 1 la paridad es válida.

En las subiteraciones de ambas arquitecturas el procesado de las M unidades CNP y de las N/G unidades VNP se realiza en paralelo. Es necesario leer y escribir una gran cantidad de información de forma concurrente, lo que implica el uso de registros para almacenar los mensajes generados por las unidades. Las conexiones entre unidades se realizan a través de multiplexores, de manera que los mensajes intercambiados entre unidades tengan el mismo origen y destino que las conexiones en el grafo de Tanner. El número de conexiones de las unidades VNP es igual que en la arquitectura completamente paralela, γ_n. Sin embargo, el número de conexiones de las unidades CNP es ρ_m/G, es decir, G veces menor.

La estructura común de las arquitecturas *shuffled* y *x-shuffled* para el código RS(3,6,3) y $G = 3$ se muestra en la Figura 5.14. El número de columnas de la matriz de paridad (N) es igual a 48, por tanto, el número de unidades VNP es $48/3 = 16$. El número de unidades CNP es 24, al igual que el número de filas de la matriz (M). Los mensajes generados por las unidades VNP se almacenan en los registros LR y los generados por las unidades CNP en los registros MR. La actualización de los registros LR y MR en la arquitectura *x-shuffled* se realiza en cada ciclo de reloj, mientras que en la arquitectura *shuffled* la actualización es alternada, en un ciclo se actualizan solamente los registros LR y en el siguiente solamente los registros MR. Los multiplexores asociados a las unidades VNP ($\mathrm{MUX_L}$) se encargan de las conexiones (bloque "Matriz de Interconexión") con las unidades CNP en cada subiteración. La selección se hace de acuerdo a la posición de los unos en las columnas de la matriz $\mathbf{H}$. Por ejemplo, en la primera subiteración el MUX asociado a la $\mathrm{VNP_1}$ selecciona las conexiones con las unidades $\mathrm{CNP_1}$, $\mathrm{CNP_{10}}$ y $\mathrm{CNP_{19}}$. En la segunda subiteración selecciona las conexiones con las unidades $\mathrm{CNP_1}$, $\mathrm{CNP_{14}}$ y $\mathrm{CNP_{23}}$. Por último, en la tercera subiteración selecciona las conexiones con las unidades $\mathrm{CNP_1}$, $\mathrm{CNP_{16}}$ y $\mathrm{CNP_{19}}$. La selección de los

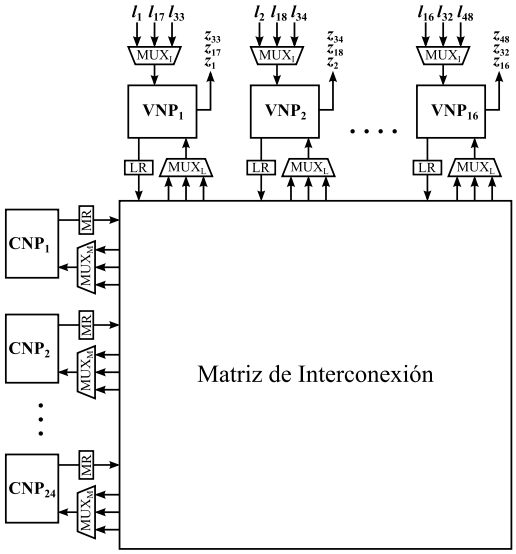


Figura 5.14: Arquitectura común *shuffled* y *x-shuffled* para el código RS(3,6,3) y un número de grupos $G = 3$. Consta de 16 unidades VNP y 24 unidades CNP que se conectan entre si a través de multiplexores (MUX_L y MUX_M). Los multiplexores MUX_I seleccionan la información *a priori* de canal y en los registros MR y LR corresponden se almacenan los mensajes.

multiplexores asociados a las unidades CNP (MUX_M) se realizan de acuerdo a la posición relativa de los unos en las filas de los grupos. Por ejemplo, el MUX asociado a la unidad CNP_2 selecciona las unidades VNP_5 y VNP_{10} en el grupo 1 (primera subiteración), las unidades VNP_5 y VNP_{11} en el grupo 2 (segunda subiteración) y las unidades VNP_3 y VNP_{10} en el grupo 3 (tercera subiteración). La información *a priori* del canal, l_n, correspondiente a cada bloque, se selecciona a través de los multiplexores MUX_I. Los bits decodificados, z_n, salen en bloques de tamaño N/G bits en cada subiteración.

Las unidades VNP, tanto para la arquitectura *shuffled* como para la *x-shuffled*, tienen la misma estructura que en la arquitectura completamente paralela (Figura 5.9). Sin embargo, la estructura de las unidades CNP difiere debido al procesado recursivo que éstas realizan. Para una subiteración g, la actualización de los nodos de comprobación (operaciones que realizan las unidades CNP) con el método de ordenación *shuffled* y el algoritmo SP están definidas por (5.16) y (5.17), donde $m \in \{1,\ldots,M\}$, $n \in N_m$ y $\lambda_{n,m}^{(0)} = 0$.

$$\mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \Psi^{-1}\Big(\sum_{\substack{n' \in N_{m\setminus n} \\ n' > g \cdot N_G}} \Psi(|\lambda_{n',m}^{(i-1)}|) + \sum_{\substack{n' \in N_{m\setminus n} \\ n' \leq g \cdot N_G}} \Psi(|\lambda_{n',m}^{(i)}|) \Big) \tag{5.16}$$

$$\Gamma_{m,n}^{(i)} = \prod_{\substack{n' \in N_{m\setminus n} \\ n' > g \cdot N_G}} \operatorname{sign}(\lambda_{n',m}^{(i-1)}) \cdot \prod_{\substack{n' \in N_{m\setminus n} \\ n' \leq g \cdot N_G}} \operatorname{sign}(\lambda_{n',m}^{(i)}) \tag{5.17}$$

Por otro lado, se define $\upsilon_{m,n}^{(i,g)}$ como el argumento de la función $\Psi^{-1}(x)$ en la ecuación 5.16, de manera que:

$$\upsilon_{m,n}^{(i,g)} = \sum_{\substack{n' \in N_{m\setminus n} \\ n' > g \cdot N_G}} \Psi(|\lambda_{n',m}^{(i-1)}|) + \sum_{\substack{n' \in N_{m\setminus n} \\ n' \leq g \cdot N_G}} \Psi(|\lambda_{n',m}^{(i)}|)$$

El sumatorio sobre el conjunto N_m excluyendo el elemento n se puede escribir como el sumatorio sobre todos los elementos del conjunto N_m menos el argumento del sumatorio evaluado en n, por consiguiente:

$$\upsilon_{m,n}^{(i,g)} = \sum_{\substack{n' \in N_m \\ n' > g \cdot N_G}} \Psi(|\lambda_{n',m}^{(i-1)}|) + \sum_{\substack{n' \in N_m \\ n' \leq g \cdot N_G}} \Psi(|\lambda_{n',m}^{(i)}|) - \begin{cases} \Psi(|\lambda_{n,m}^{(i-1)}|) & n > g \cdot N_G \\ \Psi(|\lambda_{n,m}^{(i)}|) & n \leq g \cdot N_G \end{cases}$$

En los códigos estructurados los grupos son independientes, por lo que los sumatorios se pueden expresar como una suma de sumatorios para cada grupo, tal y como se muestra en (5.18).

$$\upsilon_{m,n}^{(i,g)} = \sum_{c \in \{g+1..G\}} \Upsilon_c^{(i-1)} + \sum_{c \in \{1..g\}} \Upsilon_c^{(i)} - \begin{cases} \Psi(|\lambda_{n,m}^{(i-1)}|) & n > g \cdot N_G \\ \Psi(|\lambda_{n,m}^{(i)}|) & n \leq g \cdot N_G \end{cases} \tag{5.18}$$

$$\text{donde } \Upsilon_c^{(i)} = \sum_{\substack{n' \in N_m \\ n' > (c-1) \cdot N_G \\ n' \leq c \cdot N_G}} \Psi(|\lambda_{n',m}^{(i)}|)$$

Se puede ver en (5.18) que para el cálculo de $\upsilon_{m,n}^{(i,g)}$ se realiza la suma de los resultados parciales (función Υ_c) de G grupos consecutivos. En la arquitectura *shuffled* esta suma se hace de manera recursiva, acumulando en cada subiteración g el resultado parcial $\Upsilon_g^{(i)}$ y restando $\Upsilon_g^{(i-1)}$. El análisis anterior es también válido para la actualización de paridad $\Gamma_{m,n}^{(i)}$ (ecuación 5.17), de manera que ésta también se realiza de forma recursiva. El proceso de decodificación iterativo del algoritmo SP adaptado a la arquitectura *shuffled* se muestra en el pseudocódigo 5.2.

En la Figura 5.15 se muestra la estructura de la unidad CNP para el código RS(3,6,3) y número de grupos $G = 3$. Aunque la explicación de la operación recursiva se ha desarrollado para la arquitectura *shuffled*, la estructura de la unidad CNP planteada es válida también para la arquitectura *x-shuffled*. La magnitud de los mensajes $\lambda_{n,m}^{(g)}$, correspondientes al grupo o subiteración g, se usan para direccionar las tablas PLUT, computando de esta manera la función $\Psi(x)$. Las salidas de las tablas se suman y el resultado se almacena en el registro de desplazamiento ASRL y al mismo tiempo se acumula utilizando el registro AREG. La estructura de ventana deslizante, conformada por el registro de desplazamiento ASRL, el registro AREG y los dos sumadores, realiza las operaciones descritas en la línea 15 del pseudocódigo 5.2. Las salidas de las tablas PLUT también son almacenadas en los registros SREG con el fin de evaluar la resta en el argumento de la función $\Psi^{-1}(x)$. Por último, el resultado de la resta se satura y trunca (bloques SAT) y se calculan las salidas $\mu_{m,n}$ mediante las tablas RLUT. La saturación y el truncado se aplican para limitar el tamaño

Pseudocódigo 5.2 Algoritmo SP adaptado a la arquitectura *shuffled*.

1: – *inicialización* –
2: para $m \in \{1,\ldots,M\}$ y $n \in N_m$
3: $\quad \mu_{m,n}^{(0)} = 0$, $\Upsilon'_m = 0$ y $\Gamma'_m = 1$
4: – *iteraciones* –
5: para $i \in \{1,\ldots,I_{max}\}$
6: $\quad$ – *grupos* –
7: $\quad$ para $g \in \{1,\ldots,G\}$
8: $\quad\quad$ – *actualización de los nodos de bit (grupo)*–
9: $\quad\quad$ para $n \in \{(g-1)\cdot N_G+1,\ldots,g\cdot N_G\}$ y $m \in M_n$
10: $\quad\quad\quad \lambda_n^{(i)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(i-1)}$
11: $\quad\quad\quad \lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \mu_{m,n}^{(i-1)}$
12: $\quad\quad$ – *actualización de los nodos de comprobación* –
13: $\quad\quad$ para $m \in \{1,\ldots,M\}$ y $n \in N_m$
14: $\quad\quad\quad \Upsilon'_m = \Upsilon'_m + \Upsilon_g^{(i)} - \Upsilon_g^{(i-1)}$, donde $\Upsilon_g^{(0)} = 0$
15: $\quad\quad\quad \Gamma'_m = \Gamma'_m \cdot \Gamma_g^{(i)} \cdot \Gamma_g^{(i-1)}$, donde $\Gamma_g^{(0)} = 1$
16: $\quad\quad\quad \mu_{m,n}^{(i)} = \begin{cases} \Gamma'_m \cdot \text{sign}(\lambda_{n,m}^{(i-1)}) \cdot \Psi^{-1}\left(\Upsilon'_m - \Psi(|\lambda_{n,m}^{(i-1)}|)\right) & n > g\cdot N_G \\ \Gamma'_m \cdot \text{sign}(\lambda_{n,m}^{(i)}) \cdot \Psi^{-1}\left(\Upsilon'_m - \Psi(|\lambda_{n,m}^{(i)}|)\right) & n \leq g\cdot N_G \end{cases}$

de las tablas RLUT. Debido a que en cada subiteración la actualización de los nodos de bit (pseudocódigo 5.2) se realiza para un grupo, solamente es necesario generar los mensajes $\mu_{m,n}$ correspondientes a ese grupo. Por lo tanto, son necesarias únicamente ρ_m/G salidas que se actualizan secuencialmente en cada subiteración. El cálculo de los signos se realiza de forma similar al de las magnitudes. En este caso la estructura de ventana deslizante usa puertas XOR.

En estas arquitecturas las unidades CNP representan una parte importante en la complejidad *hardware* del decodificador. Por lo tanto, cualquier optimización de éstas se refleja en los resultados totales. Con el fin de reducir el número de operadores se ha modificado ligeramente la estructura de la unidad CNP, tal y como se ve en la Figura 5.16. La modificación consiste en realizar la saturación a la salida del acumulador aprovechando que la salida de las tablas PLUT es siempre positiva y por consiguiente, la salida del acumulador es positiva y mayor o igual que las salidas de las tablas. Este cambio reduce el número de bloques SAT y además reduce el tamaño de palabra para las restas. Además, se ha comprobado por simulación que el cambio propuesto no afecta a las prestaciones del algoritmo.

La tasa de decodificación para las arquitecturas *shuffled* y *x-shuffled* se muestra en (5.19) y (5.20), respectivamente. El factor N corresponde con la longitud del código, f con la frecuencia de reloj, It con el número de iteraciones y G con el número de grupos. Es importante resaltar que la tasa de decodificación alcanzada con la arquitectura *shuffled* es la mitad que la alcanzada con la arquitectura *x-shuffled*.

$$T_{GSH} = \frac{N \cdot f}{It \cdot 2 \cdot G} \tag{5.19}$$

$$T_{GSX} = \frac{N \cdot f}{It \cdot G} \tag{5.20}$$

La HUE de las implementaciones con la arquitectura *x-shuffled* es del 100 % debido

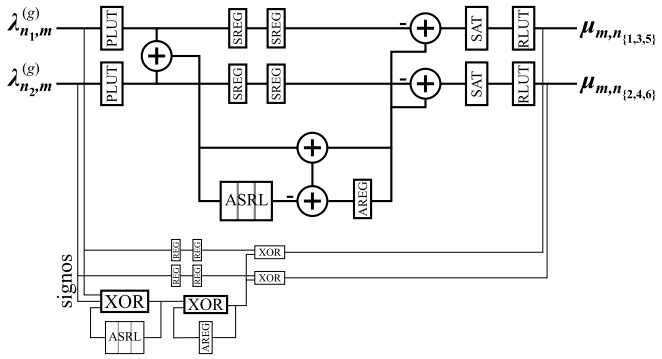


Figura 5.15: Arquitectura de la unidad CNP para el código RS(3,6,3) y número de grupos $G = 3$. La unidad calcula de forma recursiva los mensajes $\mu_{m,n}$ acumulando los resultados parciales de cada grupo g.

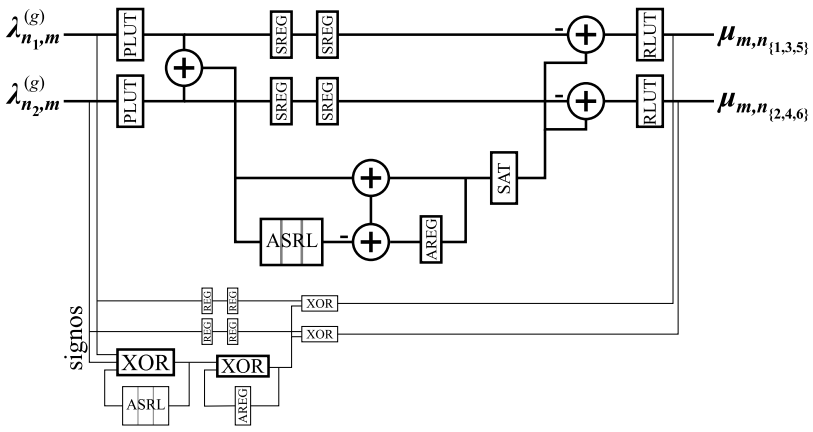


Figura 5.16: Arquitectura de la unidad CNP mejorada para el código RS(3,6,3) y número de grupos $G = 3$. La unidad calcula de forma recursiva los mensajes $\mu_{m,n}$ acumulando los resultados parciales de cada grupo g.

a que tanto las unidades CNP como las unidades VNP procesan en todos los ciclos. Sin embargo, en la arquitectura *shuffled* la actualización es alternada entre unidades, por lo tanto, mientras procesan las unidades CNP las VNP están en estado de espera y *viceversa*. Esto implica que la HUE de las implementaciones con la arquitectura *shuffled* es de apenas un 50 %. Cabe destacar que la HUE con estas arquitecturas es independiente del número de grupos.

5.5.1. Resultados de implementación

Las arquitecturas *shuffled* y *x-shuffled* se han implementado en un ASIC para el código RS(6,32,6) del estándar IEEE 802.3an con el algoritmo SP. La matriz de paridad **H** está compuesta por 6 filas y 32 columnas de submatrices, cuyo tamaño es 64×64. Las arquitecturas se han implementado para un número de grupos G igual a 2, 4, 8, 16 y 32. Además, las implementaciones cuentan con *hardware* adicional, registros y multiplexores, para mantener los datos de entrada l_n estables durante todas las iteraciones del proceso de decodificación. En éstas también se incluye el *hardware* de control, encargado de generar las señales de habilitación para las unidades CNP y VNP en los ciclos correspondientes y gestionar la finalización anticipada.

La cuantificación se ha hecho basada en los resultados del análisis de precisión finita de la Sección 3.4. Por consiguiente, los mensajes l_n,$\lambda_{n,m}$ y $\mu_{m,n}$ se cuantifican con un esquema $[6:2]$. En las unidades VNP se dejan crecer los datos de forma natural, por tanto, el esquema antes de la saturación es $[9:2]$. Las funciones $\Psi(x)$ y $\Psi(x)^{-1}$ de las unidades CNP se cuantifican con los esquemas $[9:7]$ y $[6:2]$, respectivamente. El acumulador se cuantifica con un esquema $[15:7]$ para evitar desbordamientos y para obtener la máxima precisión. En la unidad optimizada, la salida del acumulador se satura y trunca para adaptarla al esquema $[6:4]$. Este valor se resta a las salidas de las tablas PLUT, cuyos palabras están cuantificadas con un esquema $[9:7]$. Por lo tanto, es necesario truncar estos valores a $[6:4]$ para realizar la resta con la salida saturada y truncada del acumulador.

Los resultados de implementación con la librería Faraday de 90nm se muestran en la Tabla 5.8. El porcentaje de utilización en todas las implementaciones es del 65 %. Los resultados de área y camino crítico mostrados son los obtenidos después del emplazado y rutado. La tasa de decodificación se calcula usando las ecuaciones 5.19 y 5.20, siendo f la inversa del camino crítico. El número de iteraciones es diferente para cada implementación debido a la dependencia que existe entre el número de grupos y la convergencia, como se muestra en las Figuras 3.16 y 4.13. El número de iteraciones en cada caso es tal que las prestaciones (relación señal a ruido) para un PER de 10^{-5} sean las mismas que las del algoritmo SP con el método de actualización por inundación y 15 iteraciones.

En la Tabla 5.8 se puede ver que los resultados de área y camino crítico son iguales para las implementaciones con las arquitecturas *shuffled* y *x-shuffled*. Aunque el número de iteraciones requerido por la arquitectura *shuffled* es generalmente menor, las tasas de decodificación alcanzadas por las implementaciones con la arquitectura *x-shuffled* son mucho mayores. En estas últimas, la tasa de decodificación más alta la consigue la implementación con $G = 2$, aunque también es la que más área requiere. En general, el área y la tasa de decodificación disminuyen a medida que aumenta el número de grupos. Sin embargo, la implementación con $G = 32$ necesita más área que la implementación con $G = 16$ y el decremento de área de la implementación con $G = 16$ respecto a la implementación con $G = 8$ es bajo. Esto se debe a que la reducción en la complejidad *hardware* de las unidades

Tabla 5.8: Resultados de implementación en un ASIC de las arquitecturas *shuffled* y *x-shuffled* con el algoritmo SP para el código RS(6,32,6) del estándar IEEE 802.3an, usando la librería Faraday de 90nm.

Arquitectura	***G***	***It***	**Área** [mm^2]	**Camino crítico** [ns]	**Tasa de decodificación** [Gbps]
Shuffled	2	11	11.86	4.65	10.01
	4	10	8.19	4.63	5.53
	8	9	6.65	4.58	3.10
	16	8	6.47	4.27	1.87
	32	8	6.49	4.07	0.98
x-Shuffled	2	14	11.86	4.65	15.73
	4	12	8.19	4.63	9.21
	8	10	6.65	4.58	5.59
	16	9	6.47	4.27	3.33
	32	8	6.49	4.07	1.96

es menor que el aumento generado por los multiplexores. Además, las tasas de decodificación para $G = 32$ son un 47 % menores que las tasas para $G = 16$ y las tasas para $G = 8$ son un 40 % superiores a las conseguidas con $G = 16$, por lo tanto, las implementaciones con $G = 16$ y $G = 32$ no son eficientes.

Comparando los resultados con los de la implementación con la arquitectura ISMP se obtiene que para un mismo número de bloques, *slices* en un caso y grupos en el otro, la implementación con la arquitectura *x-shuffled* consigue una mayor tasa de decodificación, superior al 38 %, con un área 17 % menor. Aunque las arquitecturas sean similares, la aceleración de la convergencia del método *x-shuffled* junto con la optimización de las unidades CNP llevan a estas mejoras.

5.6. Arquitectura con actualización horizontal o *Layered*

Esta arquitectura implementa de manera eficiente el método de actualización por capas horizontales - *layered*, descrito en la Sección 3.3.2. En éste, la matriz de paridad se divide en bloques de filas, denominados capas, que se procesan se forma secuencial. En la arquitectura *layered* original, propuesta en [48], cada capa corresponde a una fila de la matriz de paridad y el almacenamiento de los mensajes se realiza en memorias. El procesado secuencial por capas reduce considerablemente la complejidad de las unidades VNP, lo que permite diseñar una única unidad CNP/VNP que computa a la vez las dos fases de la decodificación.

La arquitectura *layered* basada en memorias es ampliamente utilizada debido a que sus implementaciones alcanzan tasas de decodificación del orden de cientos de Mbps con un área reducida [29, 48–50]. Estas tasas de decodificación son suficientes para aplicaciones como las redes inalámbricas de área local Wi-Fi o las redes inalámbricas de acceso metropolitano de banda ancha WiMAX. Sin embargo, para aplicaciones que requieren tasas de decodificación de decodificación muy altas (Gbps), como por ejemplo 10-Gigabit Ethernet sobre par trenzado no blindado (IEEE 802.3an), esta arquitectura no es válida. Para conseguir tasas de decodificación muy altas es necesario paralelizar (replicar *hardware*), y para esto se aprovechan las características de los códigos estructuras (por cada fila de submatrices solamente hay un "1" por columna). De esta manera, es posible procesar z filas

consecutivas de la matriz **H** al mismo tiempo, correspondientes a una fila de submatrices, sin alterar las prestaciones del algoritmo. Al paralelizar el procesado es necesario el uso de registros para el almacenamiento de los mensajes, debido a la concurrencia en la escritura y lectura de éstos. En [51] y [53] se implementa la arquitectura *layered* basada en registros para el algoritmo βMS, alcanzando tasas de decodificación de varios Gbps. En esta tesis, la arquitectura *layered* basada en registros se ha implementado con los algoritmos propuestos αiMS y vwMS, con el fin de alcanzar altas tasas de decodificación con un bajo coste *hardware*.

Para un código estructurado, cuya matriz de paridad está compuesta por $M_b \times N_b$ submatrices de tamaño $z \times z$, el número de capas (G) corresponde con la cantidad de filas de submatrices M_b. Por consiguiente, cada capa está compuesta por z filas consecutivas de la matriz **H**. En la arquitectura *layered* basada en registros se requieren z unidades CNP que procesan en paralelo las filas de las capas y $z \cdot N_b$ unidades VNP que procesan las columnas. El procesado de las unidades VNP se hace de forma recursiva por capas, lo que reduce su complejidad considerablemente. Esta reducción permite la integración de las unidades VNP dentro de las unidades CNP, de manera que las dos fases de decodificación se puedan ejecutar en un solo ciclo de reloj. Por lo tanto, para completar una iteración son necesarios G ciclos. En cada ciclo, correspondiente a una subiteración, primero se realiza la actualización horizontal del grupo utilizando los mensajes $\lambda_{n,m}$ de la subiteración anterior. Posteriormente, se realiza de manera recursiva la actualización vertical, utilizando los mensajes de las G subiteraciones anteriores. En la subiteración 1 de la primera iteración, las actualizaciones (horizontal y vertical) se realizan utilizando la información *a priori* del canal (l_n). En cada subiteración se obtiene la información *a posteriori* de bit λ_n y se realiza la decisión *hard* de ésta. Con el fin de ahorrar recursos *hardware*, el cálculo de la paridad se realiza por capas. En cada subiteración se verifican las z ecuaciones de paridad correspondientes utilizando los bits decodificados en la subiteración anterior. La paridad se considera válida, y por tanto se puede finalizar anticipadamente, cuando se cumple la paridad en G capas consecutivas.

En la Figura 5.17 se muestra la estructura de la arquitectura *layered* basada en registros para el código RS(3,6,3). La matriz de paridad de este código, mostrada en la Figura 5.1, se compone de 3×6 submatrices de tamaño 8×8. Por lo tanto, el número de capas G es 3, el número de unidades CNP es igual a 8 y el número de unidades VNP es 48. Cada unidad CNP está conectada con 6 unidades VNP (peso de fila ρ_m), todas ellas (unidad CNP y unidades VNP) se integran en una única unidad denominada ICNP. Cada una de estas unidades genera 6 mensajes λ_n por capa, donde n depende de las posiciones de los unos en las filas correspondientes. Los mensajes se almacenan en los registros LR, los cuales se actualizan en cada una de las $G = 3$ subiteraciones. Por ejemplo, en la primera subiteración (capa 1) la unidad ICNP_8 genera los mensajes λ_4, λ_{16}, λ_{20}, λ_{26}, λ_{34} y λ_{48}. En la segunda subiteración (capa 2) genera los mensajes λ_7, λ_{15}, λ_{24}, λ_{27}, λ_{33} y λ_{48} y en la tercera subiteración (capa 3), los mensajes λ_3, λ_{10}, λ_{22}, λ_{31}, λ_{37} y λ_{48}. Los multiplexores MUX se encargan de seleccionar los mensajes de entrada a las unidades ICNP en cada subiteración. La selección se hace de acuerdo a la posición de los unos en las filas de cada capa, por lo tanto, los mensajes λ_n de entrada tienen los mismos índices que los de salida, pero corresponden a la subiteración anterior. La información *a priori* l_n que entra en la unidad ICNP es utilizada únicamente en la primera subiteración de la iteración 1, por lo que los índices de éstos corresponden con los índice de los unos en la fila correspondiente a la capa 1. Por otro lado, en cada subiteración se generan los bits decodificados z_n, los cuales se obtienen realizando la decisión *hard* de los mensajes λ_n.

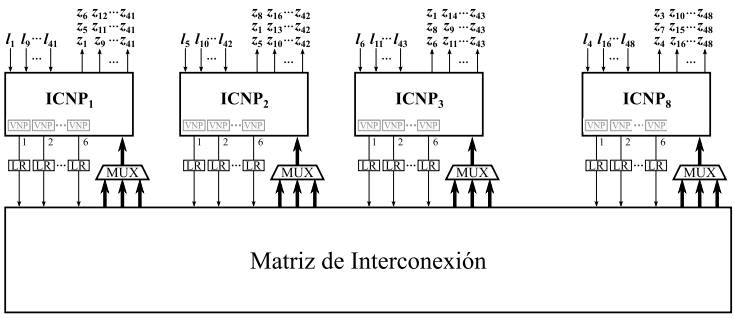


Figura 5.17: Arquitectura *layered* basada en registros para el código RS(3,6,3). Consta de 8 unidades ICNP que se conectan entre si a través de multiplexores (MUX). Los mensajes λ_n se almacenan en los registros LR.

En las unidades ICNP, la actualización horizontal se realiza en paralelo para todo los mensajes $\mu_{m,n}$, mientras que la actualización vertical (nodos de bit) se hace de forma recursiva. Para una subiteración g, la actualización de los nodos de bit con el método de ordenación *layered* y el algoritmo MS están definidas por (5.21) y (5.22), donde $n \in \{1,\ldots,M\}$ y $m \in M_n$.

$$\lambda_n^{(i)} = l_n + \sum_{\substack{m' \in M_n \\ m' > g \cdot M_G}} \mu_{m',n}^{(i-1)} + \sum_{\substack{m' \in M_n \\ m' \leq g \cdot M_G}} \mu_{m',n}^{(i)} \tag{5.21}$$

$$\lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \begin{cases} \mu_{m,n}^{(i-1)} & m > g \cdot M_G \\ \mu_{m,n}^{(i)} & m \leq g \cdot M_G \end{cases} \tag{5.22}$$

Sea $M_n[g]$ el subconjunto de nodos de comprobación adyacentes a un nodo de bit V_n para la capa g, donde $g \in \{1,\ldots,G\}$. En la arquitectura *layered*, el número de capas (G) corresponde con el número de filas de submatrices M_b, y debido a que en cada submatriz solamente hay un uno por fila y columna (característica de los códigos estructurados), los subconjuntos $M_n[g]$ contienen como máximo un elemento. Por consiguiente, los sumatorios en (5.21) se pueden expresar en término de los subconjuntos como se muestra en (5.23).

$$\lambda_n^{(i,g)} = l_n + \sum_{c \in \{g+1..G\}} \Lambda_c^{(i-1)} + \sum_{c \in \{1..g\}} \Lambda_c^{(i)} \tag{5.23}$$

$$\text{donde } \Lambda_c^{(i)} = \mu_{m',n}^{(i)}\Big|_{m' \in M_n[c]}$$

En (5.23) se puede ver que el cálculo de $\lambda_n^{(i,g)}$ se realiza sumando los resultados de G capas consecutivas. Esta operación se puede hacer de manera recursiva, acumulando en cada subiteración g el resultado parcial $\Lambda_{c=g}^{(i)}$ (iteración actual) y restando $\Lambda_{c=g}^{(i-1)}$ (resultado parcial de la iteración anterior). El proceso de decodificación iterativo del algoritmo MS adaptado a la arquitectura *layered* se muestra en el pseudocódigo 5.3.

Pseudocódigo 5.3 Algoritmo MS adaptado a la arquitectura *layered*.

1: – *inicialización* –
2: para $n \in \{1,\ldots,N\}$ y $m \in M_n$
3: $\quad \lambda_{n,m}^{(0)} = l_n$ y $M'_n = l_n$
4: – *iteraciones* –
5: para $i \in \{1,\ldots,I_{max}\}$
6: $\quad$ – *capas* –
7: $\quad$ para $g \in \{1,\ldots,G\}$
8: $\quad\quad$ – *actualización de los nodos de comprobación (capa)* –
9: $\quad\quad$ para $m \in \{(g-1)\cdot M_G+1,\ldots,g\cdot M_G\}$ y $n \in N_m$
10: $\quad\quad\quad \mu_{m,n}^{(i)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)$
11: $\quad\quad$ – *actualización de los nodos de bit* –
12: $\quad\quad$ para $n \in \{1,\ldots,N\}$ y $m \in M_n$
13: $\quad\quad\quad \Lambda'_n = \Lambda'_n + \Lambda_{c=g}^{(i)} - \Lambda_{c=g}^{(i-1)}$, donde $\Lambda_c^{(0)} = 0$
14: $\quad\quad\quad \lambda_{n,m}^{(i)} = \Lambda'_n - \begin{cases} \mu_{m,n}^{(i-1)} & m > g\cdot M_G \\ \mu_{m,n}^{(i)} & m \leq g\cdot M_G \end{cases}$

La estructura de la unidad ICNP de la arquitectura *layered* basada en registros con el algoritmo MS y para el código RS(3,6,3) se muestra en la Figura 5.18. Los multiplexores $\mathrm{MUX_I}$ se utilizan para la inicialización, de forma que en la subiteración 1 de la primera iteración seleccionan las entradas l_n y en caso contrario los mensajes λ_n de la subiteración anterior. Las restas de la entrada, junto con las sumas a la salida y los registros de desplazamiento (SRL) se utilizan para el cálculo recursivo, descrito en la línea 13 del pseudocódigo 5.3. La salida de las restas se satura (bloques SAT) para obtener los mensajes $\lambda_{n,m}$. El formato de éstos es complemento a 2, por lo que es necesario transformarlos a signo-magnitud (bloque 2C-SM) para realizar la actualización de los nodos de comprobación (línea 10). Para esto, primero se calculan los dos mínimos (bloque 2MIN) de las magnitudes de las 6 señales $\lambda_{n,m}$. A continuación se realiza la exclusión comparando el primer mínimo (min1) con las magnitudes de las señales $\lambda_{n,m}$. Si hay coincidencia, los multiplexores ($\mathrm{MUX_M}$) seleccionan el segundo mínimo, en caso contrario seleccionan el primero (ecuación 3.6). Las salidas de los bloques $\mathrm{MUX_M}$ corresponden con las magnitudes de las señales $\mu_{m,n}$. Los signos de $\mu_{m,n}$ se calculan en dos pasos. Primero se realiza una operación XOR con todos los signos de $\lambda_{n,m}$ y luego, una XOR del resultado total con el signo de cada una de las entradas.

Con el fin de reducir la complejidad *hardware* de la arquitectura *layered* implementada con los algoritmos basados en MS, los registros de desplazamiento SRL se usan para almacenar únicamente dos bits por cada señal, el bit de signo y el bit de decisión. Este último indica si la magnitud de la señal corresponde al primer o segundo mínimo. Esta modificación requiere el uso de dos registros de desplazamiento adicionales para los valores min1 y min2, así como un *hardware* adicional antes de las restas, el cual selecciona el mínimo correspondiente. En otras palabras, se replican los multiplexores $\mathrm{MUX_M}$. En el caso del algoritmo vwMS, solamente se requiere un registro de desplazamiento adicional para almacenar el valor mínimo y antes de la resta, se aplica el factor de corrección de acuerdo al bit de decisión.

La tasa de decodificación para la arquitectura *layered* basada en registros en términos de la longitud del código N, la frecuencia de reloj f, el número de iteraciones It y el número de capas G se muestra en (5.24). Cabe recordar que con el método de actualización *layered*

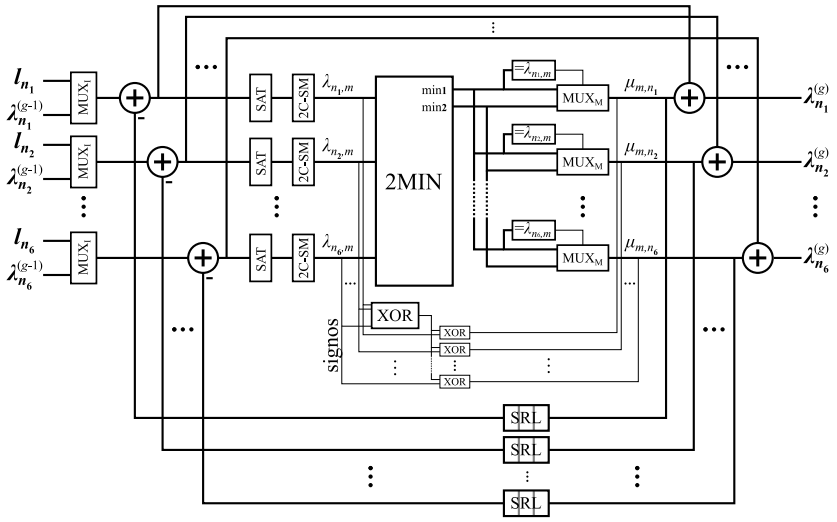


Figura 5.18: Arquitectura de la unidad ICNP para el código RS(3,6,3) con el algoritmo MS. La unidad calcula de forma recursiva los mensajes λ_n acumulando los resultados parciales de cada capa.

la convergencia de los algoritmos se acelera, por lo que el número de iteraciones se reduce, comparado con la actualización por inundación (Figura 3.15).

$$T_{LYR} = \frac{N \cdot f}{It \cdot G} \tag{5.24}$$

Debido a la integración de las unidades VNP y CNP, en cada ciclo de reloj se realizan las dos fases de actualización, horizontal y vertical. Por lo tanto, las unidades ICNP procesan en todos los ciclos de reloj, obteniendo una HUE del 100 %.

5.6.1. Resultados de implementación

La arquitectura *layered* basada en registros se ha implementado en un ASIC para el código RS(6,32,6) del estándar IEEE 802.3an con los algoritmos propuestos αiMS y vwMS. Como referencia y a manera de comparación también se ha implementado con el algoritmo αMS. El código utilizado es estructurado y su matriz de paridad está compuesta por 6 filas y 32 columnas de submatrices, cuyo tamaño es 64×64. Por lo tanto, el número de capas es $G = 6$ y el número de unidades ICNP es igual a 64. En las implementaciones se ha añadido el *hardware* necesario para realizar el control y para realizar la finalización anticipada.

Los factores de corrección y escalado utilizados en las implementaciones son los que mejores prestaciones obtienen para el código utilizado (secciones 3.4, 4.1 y 4.2). El factor α utilizando en todos los algoritmos es 0.75. En el algoritmo vwMS el vector de factores de corrección w es $[1.25; 1.75; 2.25; 2.75]$ y el de umbrales u es $[5; 10; 15]$. De la misma manera, los esquemas de cuantificación son aquellos con los que se obtiene el mejor compromiso entre prestaciones y tamaño de palabra. La información *a priori* del canal l_n se ha cuantifi-

Tabla 5.9: Resultados de implementación en un ASIC de la arquitectura *layered* basada en registros con los algoritmos αMS, αiMS y vMS para el código RS(6,32,6) del estándar IEEE 802.3an, usando la librería Faraday de 90nm. El número de iteraciones *It* es 6.

Algoritmo	**Área** [mm^2]	**Camino crítico** [ns]	**Tasa de decodificación** [Gbps]
αMS	4.12	4.98	11.42
αiMS	3.12	4.12	13.80
vwMS	3.84	4.43	12.84

cado con un esquema $[6:2]$. Este mismo esquema es usado para las señales $\lambda_{n,m}$ y $\mu_{m,n}$ en los algoritmo αMS y vwMS. En el algoritmo αiMS se cuantifican las señales $\lambda_{n,m}$ con un esquema $[6:2]$ y las señales $\mu_{m,n}$ con un esquema $[4:0]$. Para evitar desbordamientos, las señales λ_n se cuantifican con el esquema $[10:2]$.

Los resultados de implementación con la librería Faraday de 90nm se muestran en la Tabla 5.9. El porcentaje de utilización en todas las implementaciones es del 75 %. Los resultados de área y camino crítico mostrados son los obtenidos después del emplazado y rutado. La tasa de decodificación se calcula usando la ecuación 5.24, siendo f la inversa del camino crítico. El número de iteraciones It es igual a 6, de forma que para un PER de 10^{-5}, las prestaciones (relación señal a ruido) son las mismas que con el método de actualización por inundación y 15 iteraciones (Figura 3.15).

En los resultados mostrados en la Tabla 5.9 se puede ver que la implementación con el algoritmo αiMS es la que menos área requiere y mayor tasa de decodificación alcanza. La implementación con el algoritmo vwMS requiere un 18 % más de área y su tasa de decodificación es un 7 % menor. Sin embargo, los resultados de implementación con esta última son mejores que con el algoritmo αMS.

Al comparar los resultados con los de las implementaciones de la arquitectura parcialmente paralela basada en memorias, se aprecia que el área que requieren las implementaciones de la arquitectura *layered* basada en registros es menor (aproximadamente un 13 %). Por lo tanto, la complejidad *hardware* de las implementaciones con la arquitectura *layered* se puede considerar baja.

Finalmente, destacar las altas tasas de decodificación que alcanzan las implementaciones de la arquitectura *layered* basada en registros. Éstas son superiores a las conseguidas con la arquitectura completamente paralela, alcanzando los 13.8 Gbps para el caso del algoritmo αiMS.

5.7. Comparación con implementaciones existentes

Tras el análisis de las arquitecturas expuestas en las secciones anteriores se concluye que las que mejores resultados de implementación obtienen, y por tanto las más óptimas, son las *x-shuffled* y *layered*. Por lo tanto, en esta sección se realiza la comparación de éstas con diferentes implementaciones (arquitecturas) encontrados en la literatura. Concretamente se han seleccionado las implementaciones más relevantes para el código RS(6,32,6) del estándar IEEE 802.3an.

En [46] se presenta un decodificador basado en el método de actualización *shuffled* para

Tabla 5.10: Resultados de implementación en un ASIC de diferentes arquitecturas y algoritmos para el código RS(6,32,6) del estándar IEEE 802.3an.

Arquitectura (Algoritmo)	***It***	**SNR@ PER=10^{-5}** [dB]	**Área (A)** [mm^2]	***f*** [MHz]	**Th** [Gbps]	**Ratio Th/A**
x-Shuffled $G = 2$ (SP)	14	4.34	11.86	215	15.73	1.32
x-Shuffled $G = 4$ (SP)	12	4.34	8.19	216	9.21	1.12
Layered (αiMS)	6	4.34	3.12	243	13.8	4.42
Layered (vwMS)	6	4.32	3.84	226	12.8	3.34
Shuffled modificada (αMS) [46]	8	4.34	4.41	303	4.85	1.09
Parallel bit-serial (smMS) [33]	16	4.75	9.8	250	8	0.81
Layered (MS post-procesado) [52]	8	4.25	10.7[1]	560[1]	11.92[1]	1.11
Split-Row Thresh. (Split-MS) [37]	11	4.55	9.68[1]	156[1]	29.04[1]	3
Full-duplex layered (βMS) [53]	6	4.30	5.35	137	7.79	1.45
Pipelined layered (αMS) [54]	5	4.34	9.2[1]	348[1]	11.85[1]	1.28

[1]Resultados escalados a la tecnología de 90nm.

el algoritmo αMS. En éste se utilizan permutadores para optimizar las conexiones entre unidades y además, se utiliza un método de ordenación para calcular los mínimos recursivamente de forma eficiente. En [33] se reduce el consumo de potencia mediante la serialización de los mensajes en una arquitectura completamente paralela con el algoritmo smMS. Con la serialización también se consigue reducir significativamente la complejidad de las conexiones y de las unidades. El decodificador implementado en [52] utiliza la arquitectura *layered* basada en registros con el algoritmo MS e incorpora un post-procesado para mejorar el suelo de error, el cual aparece a bajas tasas de error y se manifiesta como un cambio de pendiente en las curvas de error. Además, en esta implementación se realiza una división de las conexiones en locales y globales, con el fin de optimizar la congestión. En [37] se propone un algoritmo basado en MS de baja complejidad, llamado *Split-row threshold*. En éste, se divide la matriz de paridad en bloques de columnas, los cuales se procesan en paralelo de forma casi independiente. Este algoritmo incurre pérdidas en las prestaciones, sin embargo, permite reducir la congestión debida a las conexiones y la complejidad en general. Las implementaciones en [53] y [54] se basan en el método de actualización *layered*. En ambas se proponen diferentes esquemas de comunicación entre unidades con la finalidad de reducir la congestión en las conexiones. En [54] ademas se utiliza segmentación para disminuir el camino crítico.

En la Tabla 5.10 se comparan los resultados de las diferentes implementaciones. El número de iteraciones (*It*) es aquel con el que se consigue la SNR de la tercera columna, para una PER de 10^{-5}. Para poder hacer una comparación de las implementaciones con diferentes tecnologías CMOS, los resultados de área (A), frecuencia (f) y tasa de decodificación (Th) se han escalado a la tecnología de 90nm, usando los factores de conversión presentados en [61]. Para convertir los resultados de 65nm se usa un factor de 2 para el área y de 0.8 para la frecuencia y tasa de decodificación. Los resultados de área para 130nm se multiplican por 0.5 y los de frecuencia y tasa de decodificación por 1.25. Como medida de comparación se utiliza el ratio entre tasa de decodificación y área (Th/A), el cual define la cantidad de Gbps por mm^2 que alcanza el decodificador.

Se observa en la Tabla 5.10 que la arquitectura *Split-Row Threshold* es la que mayor tasa de decodificación consigue, siendo casi el doble que la segunda (arquitectura *x-shuffled*

con 2 grupos). Sin embargo, el ratio Th/A conseguido por las arquitecturas *layered* con los algoritmos αiMS y vwMS es mayor. Éstas son las que menos área requieren y alcanzan tasas de decodificación superiores a los 12 Gbps. La arquitectura menos eficiente, en términos de ratio, es la *parallel bit-serial* y además es la que peores prestaciones consigue. Las arquitecturas propuestas *x-shuffled* consiguen tasas de decodificación altas, pero tienen un coste *hardware* elevado y por tanto, el ratio no es muy alto. Sin embargo, destaca el hecho de que son las únicas que implementan el algoritmo SP, cuya complejidad es mayor a las de los algoritmos basados en MS.

5.8. Conclusiones

En este trabajo se presenta una arquitectura para el cálculo de los dos mínimos, requerido en la mayoría de algoritmos basados en MS, que consigue una reducción en la complejidad hardware de entre el 1.6 % y el 17.4 %, comparado con las arquitecturas existentes. Para el caso de interés, 32 entradas, la reducción es de al menos el 6.4 %.

Los algoritmos expuestos en capítulos anteriores se han implementado en un ASIC con las arquitecturas completamente paralela y parcialmente paralela basada en memorias, con el fin de evaluar el impacto de los algoritmos propuestos, iMS, αiMS y vwMS. Con la arquitectura completamente paralela, las implementaciones con los algoritmos iMS y αiMS requiren un 32 % menos de área que con el algoritmo MS y alcanzan unas tasas de decodificación 20 % superiores. Con el algoritmo vwMS la tasa de decodificación es similar a la de los algoritmos MS, αMS y βMS, sin embargo, el área es un 16 % menor. En las implementaciones con la arquitectura parcialmente paralela basada en memorias también se consigue reducir el área con los algoritmos propuesto, aunque en este caso las tasas de decodificación son similares (sobre los 340 Mbps). Con los algoritmos iMS y αiMS se consigue una reducción en el área del 25 %, mientras que con el algoritmo vwMS es del 8 %. Esta arquitectura se utilizará en el simulador *hardware* descrito en el capítulo 6, por lo que también se ha implementado en un dispositivos FPGA, alcanzando tasas de decodificación superiores a los 170 Mbps con todos los algoritmos estudiados.

Otra aportación hecha en este trabajo es la mejora de la arquitectura *Sliced-Message Passing* (SMP) [38], la cual se ha denominado ISMP (*improved*-SMP). La arquitectura propuesta consigue una reducción en el área de la arquitectura SMP de un 33 % cuando es implementada con el algoritmo SP. Además, la arquitectura ISMP ha servido como base para proponer las arquitecturas *shuffled* y *x-shuffled*, las cuales implementan de manera eficiente los métodos de actualización con el mismo nombre. La arquitectura *shuffled* consigue una tasa de decodificación de 10 Gbps con un área de 11.86 mm^2, pero su HUE es de tan solo el 50 %. En cambio, la arquitectura *x-shuffled* consigue un 100 % y alcanza una tasa de decodificación de 15.73 Gbps, con la misma área.

Los algoritmos αiMS y vwMS también se han implementado con la arquitectura *layered* basada en registros. Con esta arquitectura se consiguen un buen compromiso entre área y tasa de decodificación y una HUE alta (100 %). La implementación con el algoritmos αiMS alcanza una tasa de decodificación de 13.8 Gbps con un área de 3.12 mm^2, mientras que la implementación con el algoritmo vwMS alcanza 12.8 Gbps y necesita 3.84 mm^2. Comparado con otras implementaciones existentes de la literatura, para el mismo código LDPC, éstas son las que menos área requieren, además de ser las que mayor ratio entre tasa de decodificación y área obtienen.

Capítulo 6

SIMULADOR HARDWARE

Las prestaciones de los códigos LDPC a bajas tasas de error (PER $< 10^{-8}$) se suelen ver afectadas por un fenómeno conocido como suelo de error, que se manifiesta como un cambio en la pendiente de la curva de error. Este efecto se muestra en la Figura 6.1, en la que se presentan las curvas de BER y PER del algoritmo αMS para el código LDPC del estándar IEEE 802.3an. La degradación producida por el suelo de error puede hacer inservible el código en aplicaciones que requieren tasas de error bajas, como es el caso de los sistemas de comunicaciones ópticas o sistemas magnéticos. Por ello en estos sistemas es necesario evaluar su comportamiento a estas tasas de error, lo que supone un reto computacional en sí mismo. Para obtener las prestaciones a tasas de error bajas es necesario simular una gran cantidad de paquetes, del orden de 10^{10}, para garantizar un resultado estadísticamente fiable. Al realizar estas simulaciones por *software* se requieren varios meses para obtener las prestaciones, ya que la máxima tasa de decodificación que se alcanza es del orden de cientos de kbps.

Los simuladores *hardware* basados en FPGA permiten reducir considerablemente los tiempos de simulación, pues alcanzan tasas de decodificación del orden de Gbps [62–65]. En [62] se presenta un simulador *hardware* que utiliza un decodificador LDPC serie-paralelo, el cual consigue una máxima tasa de decodificación de 240 Mbps. En [63] se propone una estructura de múltiples núcleos (*cores*), usando tarjetas BEE2 con 5 dispositivos FPGA cada una, para el análisis de prestaciones de los códigos LDPC en canales de almacenamiento magnético. Este simulador está compuesto por 27 *cores* que trabajan en paralelo. Cada *core* alcanza una tasa de decodificación de 175 Mbps, por lo que el sistema completo consigue una tasa de decodificación de 4.7 Gbps. En [64] se paralelizan varios decodificadores en una mismo dispositivo FPGA utilizando vectores (agrupaciones de señales de los diferentes decodificadores), alcanzando una tasa de decodificación de 1.23 Gbps con 4 *cores* (332 Mbps con un *core*). En éste se implementa una modificación de la arquitectura parcialmente paralela basada en memorias que permite la decodificación de los vectores y en la que se aprovecha al máximo los recursos de almacenamiento (memorias embebidas) disponibles en los dispositivos FPGA. El simulador expuesto en [65] alcanza una tasa de decodificación de 158 Mbps con un sólo *core* y de 950 Mbps con 6 *cores* implementados en

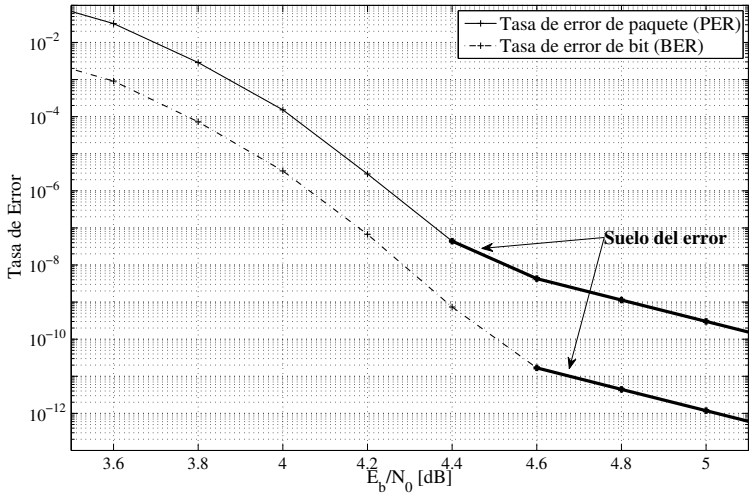


Figura 6.1: Suelo de error con el código LDPC del estándar IEEE 802.3an y el algoritmo de decodificación αMS.

una placa BEE3. En esta tesis se ha diseñado e implementado un simulador *hardware* que utiliza un decodificador LDPC con arquitectura parcialmente paralela basada en memorias y alcanza una tasa de decodificación de 664 Mbps y 1.35 Gbps con un sólo *core*, en dispositivos FPGA Altera Cylcone IV-E y Xilinx Virtex-6, respectivamente. El simulador puede ser implementado con una estructura de múltiples *cores*, aumentando linealmente la tasa de decodificación en función del número de *cores* implementados.

6.1. Arquitectura

El diagrama de bloques detallado del simulador *hardware* propuesto se muestra en la Figura 6.2. El simulador se compone de cuatro bloques y, en general, su funcionamiento es el mismo que los simuladores *software*. 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. El simulador utiliza un decodificador con arquitectura parcialmente paralela basada en memorias, por lo que es válido únicamente para códigos estructurados y algoritmos de decodificación del tipo de intercambio de mensajes.

Algunos parámetros de la simulación se configuran a través de una interfaz *software*, como la varianza del ruido σ^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 F_{err}. Otros parámetros, como el código LDPC, el algoritmo de decodificación y los esquemas de cuantificación se configuran antes de la síntesis. Para facilitar la configuración pre-síntesis se ha desarrollado un *script* en Matlab que genera un paquete VHDL, con el formato requerido por el simulador, el cual contiene el código LDPC y los esquemas de cuantificación. Para configurar el algoritmo es necesario codificar en VHDL las unidades CNP y VNP siguiendo algunas

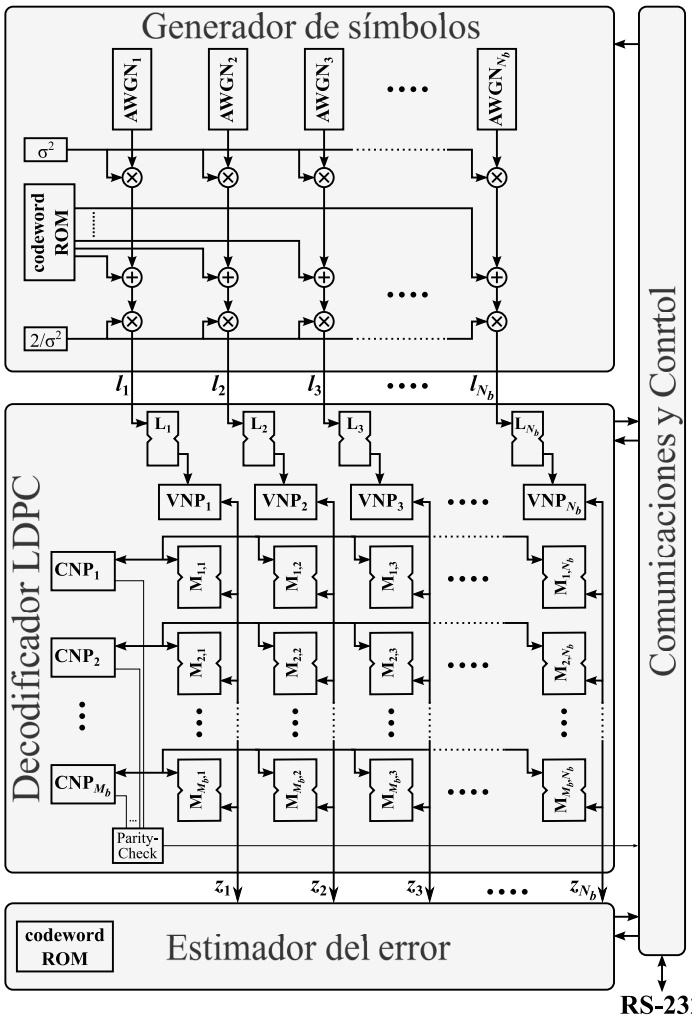


Figura 6.2: Diagrama de bloques detallado del simulador *hardware*.

pautas para la compatibilidad con el resto de componentes del simulador.

6.1.1. Generador de símbolos

La generación de los símbolos recibidos se hace a partir de una palabra código (*codeword*) almacenada en una memoria ROM, la cual ha sido generada codificando una trama de datos aleatoria. En el simulador hardware no se ha implementado el codificador debido al alto coste que tiene por lo que se ha usado una palabra código fija para todas las simulaciones, aprovechando la linealidad existente en los códigos LDPC. Además, mediante simulación *software* se ha comprobado que los resultados obtenidos con una palabra código fija son similares a los obtenidos con palabras código aleatorias. Los bits de la palabra código se modulan en BPSK y se les suma ruido blanco Gaussiano con el fin de emular el canal AWGN.

El ruido blanco Gaussiano se genera utilizando el método de la inversión de la función de distribución acumulada (ICDF) propuesto en [66]. Primero se generan números aleatorios uniformemente distribuidos a partir de una semilla y luego se aplica la inversa de la función de distribución Gaussiana. La inversa se aproxima mediante interpolación polinómica utilizando partición no uniforme. El ruido generado tiene varianza unitaria por lo que es necesario escalarlo por σ^2 para obtener la SNR requerida. La señal recibida (palabra código modulada más ruido blanco Gaussiano) 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 [67] con una longitud de 64 bits, el cual alcanza una periodicidad de 2^{175}. Gracias a esto, el generador AWGN consigue un valor máximo de la desviación estándar de la distribución Gaussiana de $\pm 9.4\sigma$.

Las bajas tasas de error se consiguen con relaciones señal a ruido (SNR) altas, donde la mayoría de paquetes transmitidos se pueden decodificar con una sola iteración. En este caso, el decodificador alcanza la máxima tasa de decodificación, determinada por (6.1), donde N es la longitud del código, f la frecuencia de reloj, z el tamaño de las submatrices del código estructurado y c el grado de segmentación de las unidades del decodificador (suma de latencias de las unidades VNP y CNP).

$$T_{max} = \frac{N \cdot f}{2 \cdot z + c} \tag{6.1}$$

Para que la generación de símbolos no sea el cuello de botella, ésta tiene que ser capaz de generar los datos a la máxima tasa de decodificación. Para esto se han implementado N_b generadores en paralelo, aprovechado de esta manera la arquitectura del decodificador, en la que los datos de entrada se introducen en bloques de N_b símbolos. Cada uno de los N_b generadores produce z símbolos consecutivos, para así completar los $N = z \cdot N_b$ símbolos correspondientes a un paquete. Al tener varios generadores de símbolos en paralelo se debe garantizar que las secuencias de éstos sean independientes. Para esto se han generado N_b semillas de forma aleatoria y se ha comprobado que los números aleatorios uniformemente distribuidos generados (generador Tausworthe) 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 cada uno de los N_b generadores AWGN produce por lo menos 10^{14} datos independientes. En el caso de las implementaciones con múltiples *cores* es necesario garantizar la independencia de los generadores de todos los *cores*, por lo que es necesario buscar $C_s \cdot N_b$ semillas

que generen datos independientes, donde C_s se refiere al número de *cores*.

6.1.2. Decodificador de códigos LDPC

Como se ha dicho anteriormente, el decodificador se implementa con la arquitectura parcialmente paralela basada en memorias, descrita en la Sección 5.3. Con esta arquitectura se consigue una buena relación entre complejidad *hardware* y tasa de decodificación, alcanzando valores altos (cientos de Mbps o incluso Gbps). Como se puede ver en la Figura 6.2, el decodificador se componen de M_b unidades CNP_m, N_b unidades VNP_n, $2 \cdot M_b \cdot N_b$ memorias ($\text{M}_{m,n}$) que almacenan los mensajes intercambiados $\lambda_{n,m}$ y $\mu_{m,n}$, y N_b memorias (L_n) que almacenan los mensajes de entrada l_n. Estos últimos provienen del generador de símbolos.

El simulador implementa la finalización anticipada con el fin de reducir el número de iteraciones medio en relaciones señal a ruido altas (baja tasa de error). Para esto, el decodificador verifica la paridad (bloque Parity-Check) con los bits decodificados por iteración y si ésta se cumple, se indica mediante un *flag*. En el caso de que la paridad sea válida, el bloque de control se encarga de detener el proceso de decodificación. En cada iteración, las salidas del decodificador son la palabra decodificada (z_n), el *flag* de paridad y el valor de la iteración.

6.1.3. 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 (F_{err}) en la máxima iteración (I_{max}). Por lo 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.

6.1.4. Comunicaciones y control

Este bloque se encarga del control interno del simulador *hardware* y de la comunicación con la interfaz software. Desde la interfaz *software* se envían los parámetros de configuración del simulador que se almacenan en registros. Además, la interfaz *software* es la encargada de indicar al simulador cuándo comienza la simulación a través de una palabra de control. Cuando la interfaz *software* detecta que la simulación ha concluido envía una palabra de control que indica al simulador 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 simulador se encuentra en funcionamiento, así mismo, desde la interfaz *software* se pueden leer los registros de configuración para

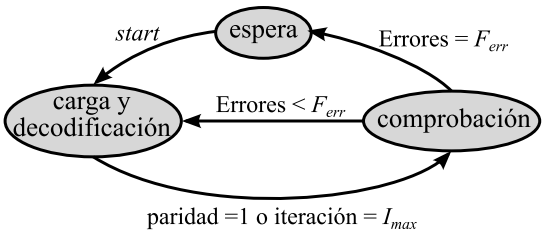


Figura 6.3: Diagrama de estados de la maquina de control del simulador *hardware*.

poder comprobar que lo que se ha enviado ha sido recibido correctamente. La comunicación entre la interfaz *software* y el simulador *hardware* se realiza por medio del puerto serie usando el estándar RS-232.

El diagrama de estados del bloque de control se muestra en la Figura 6.3. La simulación comienza cuando se recibe la palabra de inicio (*start*) desde la interfaz *software*. En el estado "carga y decodificación" se generan las señales que habilitan los bloques generador y decodificador. La señal de habilitación del bloque generador tiene una duración de z ciclos de reloj (genera un paquete de $N = z \cdot N_b$ símbolos), mientras que la del decodificador permanece en alto hasta que termina el proceso de decodificación del paquete recibido. En el estado "comprobación" primero se verifica la paridad y el número de iteraciones: si se cumple la paridad el paquete no contiene errores y por tanto, el contador de errores no se actualiza y si por contra, se llega al número máximo de iteraciones y la paridad no se cumple, el contador de errores se incrementa en uno. Luego se comprueba si el número de errores es inferior al número máximo de errores F_{err}, configurado desde la interfaz *software*. Cuando el número de errores alcanza el máximo se pasa al estado "espera", activando el *flag* de simulación terminada. En caso contrario se vuelve al estado "carga y decodificación" para simular de esta manera el siguiente paquete.

El control también se encarga de almacenar el número de ciclos de reloj que dura la simulación. Con este valor se puede determinar el tiempo que tarda la simulación y el número medio de iteraciones.

6.1.5. Interfaz *software*

Desde la interfaz *software*, desarrollada en Matlab, se configura el simulador *hardware* y se leen los resultados de la simulación. La comunicación con el simulador se realiza en dos vías: primero se envía una palabra de control para indicar al simulador la tarea a realizar y éste devuelve un resultado o confirmación. Se utiliza el puerto de comunicaciones RS-232 configurado a una velocidad de 57.6 kbps y los datos se transmiten en bloques de 8 bits.

Antes de empezar la simulación es necesario enviar al simulador *hardware* los parámetros de configuración (σ^2, $2/\sigma^2$, I_{max} y F_{err}). Una vez cargada la configuración se envía la palabra para iniciar la simulación. Periódicamente se envía la palabra de control para verificar el estado del simulador. Si se detecta que la simulación ha terminado se procede a leer los resultados. Por último, en la interfaz *software* los resultados se almacenan en ficheros para su posterior análisis. Para identificar las simulaciones (algoritmo y código) se usa un código de 4 dígitos hexadecimales, el cual se especifica en el simulador *hardware* antes de

la síntesis. Este código es leído por la interfaz *software* y es usado para nombrar los ficheros en los que se almacenan los resultados.

En el caso de configuraciones de múltiples *cores* el código de identificación es el mismo para todos los *cores*. El número de errores F_{err} para cada *core* se calcula dividiendo el valor F_{err} entre el número de *cores*. Por otra parte, cuando se utiliza una configuración multi-*core* con varios dispositivos FPGA (un *core* por dispositivo), la interfaz *software* realiza una operación de consulta constante (*polling*) a cada *core* para verificar el estado de cada simulación. Cuando todas las simulaciones han terminado se procede a la lectura de los resultados, simulador por simulador. Los resultados de bits erróneos, paquetes erróneos y paquetes transmitidos se suman, mientras que el número de ciclos de reloj resultante se determina seleccionando el máximo. Cuando los múltiples *cores* están implementados en un mismo dispositivo FPGA el funcionamiento es diferente. En este caso la configuración multi-*core* es transparente para la interfaz *software*, a excepción del número máximo de errores F_{err}. En *hardware* se determina cuando han finalizado todos los *cores* mediante un único estado. También se realizan las sumas de los resultados de bits y paquetes erróneos, así como la de paquetes transmitidos. El contador de ciclos de reloj es conjunto a todos los *cores*, por lo tanto, la selección del máximo es implícita.

6.2. Implementación del simulador

El simulador *hardware* se ha implementado para el código LDPC del estándar IEEE 802.3an con los algoritmos MS, αMS y βMS, presentados en el Capítulo 3, y con los algoritmos propuestos iMS, αiMS y vwMS, mostrados en el Capítulo 4. Además, estos algoritmos se han evaluado a bajas tasas de error utilizando diferentes esquemas de cuantificación. Para las implementaciones se han usado los dispositivos FPGA Cyclone IV-E EP4CE115F29C7 y Virtex-6 XC6VLX240T de las compañias Altera y Xilinx, respectivamente.

En el dispositivo Cyclone IV-E es posible implementar un único *core* de los casos a evaluar, por lo que la configuración multi-*core* requiere el uso de varios dispositivos. Sin embargo, en el dispositivo Virtex-6 es posible incluir 4 *cores* en el mismo dispositivo. Además, la frecuencia de reloj conseguida con el dispositivo Virtex-6 es el doble que en el dispositivo Cyclone IV-E. Por lo tanto, la implementación del simulador en el dispositivo Virtex-6 alcanza una tasa de decodificación 8 veces superior que la conseguida con la implementación en el dispositivo Cyclone IV-E. Por ejemplo, la implementación de un solo *core* con el algoritmo αMS en el dispositivo Altera Cyclone IV-E requiere 53330 elementos lógicos (LEs), 388 memorias embebidas M9K y 256 multiplicadores embebidos P9, y alcanza una frecuencia de reloj de 150 MHz. El porcentaje de utilización del dispositivo es del 46 %, 90 % y 48 % del total de LE's, memorias M9K y multiplicadore P9, respectivamente.

La implementación de un único *core* del simulador en el dispositivo Xilinx Virtex-6 con el algoritmo αMS requiere 7986 slices, 76 Block RAMs y 128 multiplicadores embebidos y alcanza una frecuencia de reloj de 300 MHz (3.3ns de camino crítico). En este mismo dispositivo es posible implementar 4 *cores* que alcanzan la misma frecuencia de trabajo (300 MHz). En este caso la implementación requiere 32324 slices, 304 Block RAMs y 512 multiplicadores embebidos. El porcentaje de uso del dispositivo para esta implementación es del 85 %, 36 % y 67 % del total de slices, Block RAMs y multiplicadores embebidos, respectivamente.

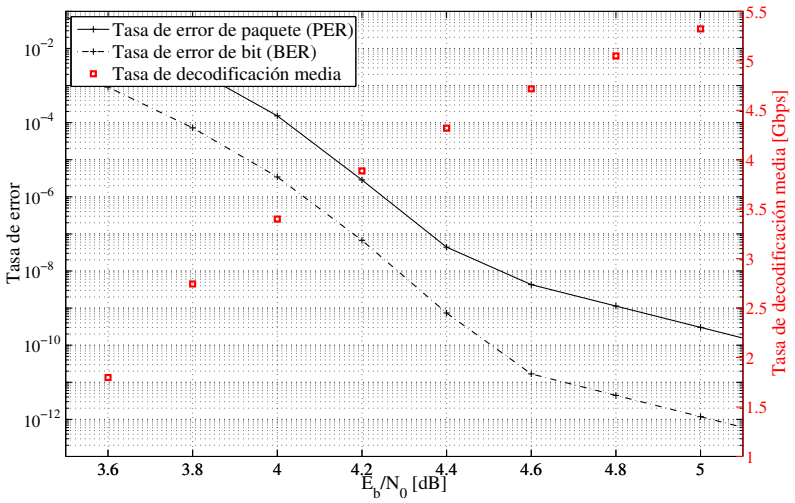


Figura 6.4: Curva de error y tasa de decodificación media para el código LDPC del estándar IEEE 802.3an con el algoritmo αMS. Simulador implementado con 4 *cores* en un dispositivo Virtex-6 ($f = 300$ MHz).

En la Figura 6.4 se muestra la tasa de decodificación media en función de la tasa de error para la implementación del simulador con 4 *cores* en el dispositivo Virtex-6, para el código del estándar IEEE 802.3an y el algoritmo αMS. En este caso el máximo número de errores (F_{err}) es igual a 100 (cada *core* busca 25 errores) y el máximo número de iteraciones (I_{max}) es igual a 50. La tasa de decodificación media se calcula dividiendo el número de paquetes simulado sobre el tiempo de simulación (número de ciclos de reloj/f) y multiplicando el resultado por el número de bits por paquete (N). Se observa que la tasa de decodificación media aumenta a medida que la tasa de error (PER o BER) disminuye, alcanzando 5.4 Gbps para un BER de 10^{-12}. Este punto requiere la simulación de 3.3×10^{11} paquetes de 2048 bits, por lo que son necesarias 35 horas para obtener el resultado.

En la Tabla 6.1 se comparan las tasas de decodificación alcanzadas por otros simuladores *hardware* encontrados en la literatura con la conseguida por la implementación del simulador propuesto. Se puede ver que la tasa de decodificación alcanzada por el simulador de este trabajo con un solo *core* es al menos 4 veces mayor que las conseguidas en otros trabajos. Además, utilizando solamente 4 *cores* se consigue una tasa de decodificación superior al resto de los simuladores *hardware*.

6.3. Resultados de simulación

En esta sección se muestran las curvas de error obtenidas con el simulador *hardware*. Aunque se han simulado varios factores y pesos para los algoritmos αMS, βMS, αiMS y vwMS, solamente se muestran los casos que mejores prestaciones obtienen. El objetivo de estas simulaciones es corroborar los datos obtenidos mediante simulación *software* para tasas de error de bit superiores a 10^{-8} y analizar el comportamiento de los algoritmos en la

Tabla 6.1: Comparación de las tasas de decodificación medias de varios simuladores *hardware* de la literatura con la conseguida con simulador propuesto.

	[65]	[63]	[62]	[64]	Este trabajo
Longitud del código (N)	2304	4923	2048	3369	2048
Tasa de decodificación (un *core*) [Mbps]	158	175	240	332	1350
Tasa de decodificación (multi-*core*) [Gbps]	0.95	4.70	–	1.23	5.40
Número de *cores*	6	27	–	4	4

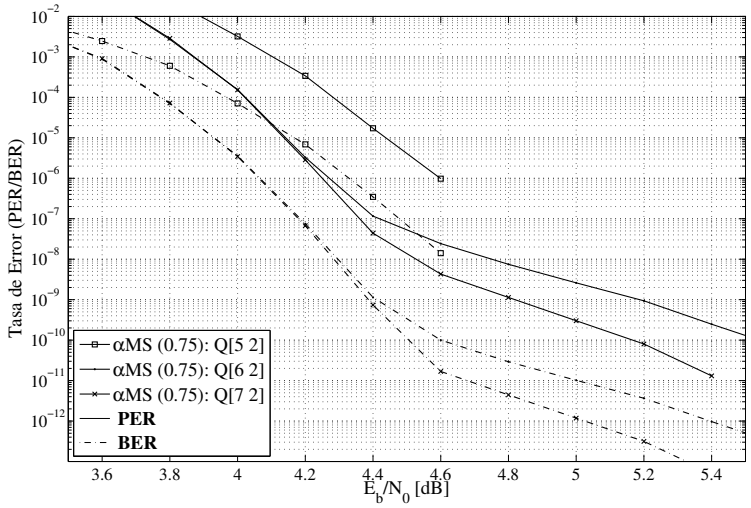


Figura 6.5: Tasa de error del algoritmo αMS para el código del estándar IEEE 802.3an con varios esquemas de cuantificación.

zona baja de BER (menor a 10^{-8}).

En la Figura 6.5 se presentan las curvas de error para el algoritmo αMS con el código del estándar IEEE 802.3an y varios esquemas de cuantificación. Se aprecia que las pérdidas de prestaciones con el esquema $[5:2]$ son altas, mayores a 0.35 dB para BER$= 10^{-8}$. Las prestaciones de los esquemas $[6:2]$ y $[7:2]$ son similares hasta los 4.4 dB de E_b/N_0, sin embargo, con el esquema $[6:2]$ el suelo de error aparece casi dos órdenes de magnitud antes que con el esquema $[7:2]$.

Las curvas de error de los algoritmos de referencia MS, αMS y βMS se muestran en la Figura 6.6. Se aprecia que las prestaciones de los algoritmos αMS y βMS son similares para todo el rango de E_b/N_0 simulado. Es importante destacar que aunque la convergencia de los algoritmos αMS y βMS es más rápida que la del algoritmo MS, el suelo de error en todos los algoritmos tiende al mismo punto (BER$= 10^{-11}$ para el esquema $[6:2]$).

Un comportamiento similar al obtenido con los algoritmos de referencia se aprecia con los algoritmos propuestos iMS y αiMS. El suelo de error de éstos aparece en el mismo

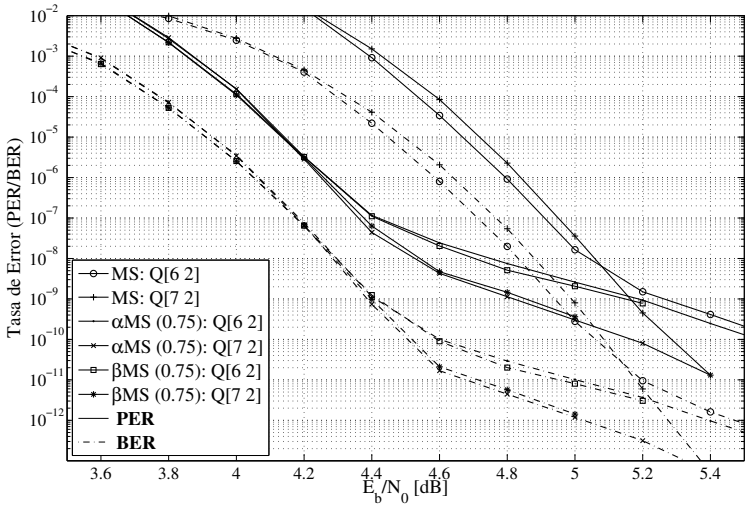


Figura 6.6: Tasa de error de los algoritmos MS, αMS y βMS para el código del estándar IEEE 802.3an.

punto (BER= 10^{-9} para el esquema $[6:2]$ y BER= 10^{-11} para el esquema $[7:2]$) que en el algoritmo αMS y converge al mismo punto.

En las Figuras 6.8 y 6.9 se muestran las curvas de error del algoritmo vwMS con factores de escalado α iguales a 0.5 y 0.75, respectivamente. Como referencia se incluyen las curvas correspondientes al algoritmo αMS con esquemas de cuantificación $[6:2]$ y $[7:2]$. Para el factor de escalado 0.5 los vectores de factores de correción y umbrales son $u = [5;10;15]$ y $w = [1.25;1.75;2.25;2.75]$, respectivamente. Mientras que para el factor de escalado 0.75 los vectores son $u = [5;10;15]$ y $w = [0.5;1;1.5;2]$. En la Figura 6.8 se puede ver que para BER superiores a 10^{-9} las prestaciones del algoritmo vwMS con factor de escalado $\alpha = 0.5$ son similares a las obtenidas con el algoritmo MS, utilizando ambos esquemas de cuantificación. Sin embargo, el suelo de error del algoritmo vwMS aparece un orden de magnitud más abajo. Destaca el hecho de que el suelo de error del algoritmo vwMS con el esquema de 6 bits coincide con el suelo de error del algoritmo MS con un esquema de 7 bits.

Las prestaciones del algoritmo vwMS con factor de escalado $\alpha = 0.75$ son peores que las conseguidas con el algoritmo αMS, para tasas de error de bit mayores a 10^{-9}. Sin embargo, en la zona baja de BER (menor a 10^{-9}) el comportamiento del algoritmo vwMS es mucho mejor. Con el esquema $[6:2]$ el suelo de error de éste aparece tres órdenes de magnitud por debajo del suelo de error del algoritmo MS con esquema $[6:2]$ y casi un orden de magnitud menos que con el esquema $[7:2]$. Cabe destacar las buenas prestaciones a bajas tasas de error del algoritmo vwMS con esquema $[7:2]$, en las que no se aprecia suelo de error por encima de un BER de 10^{-13}.

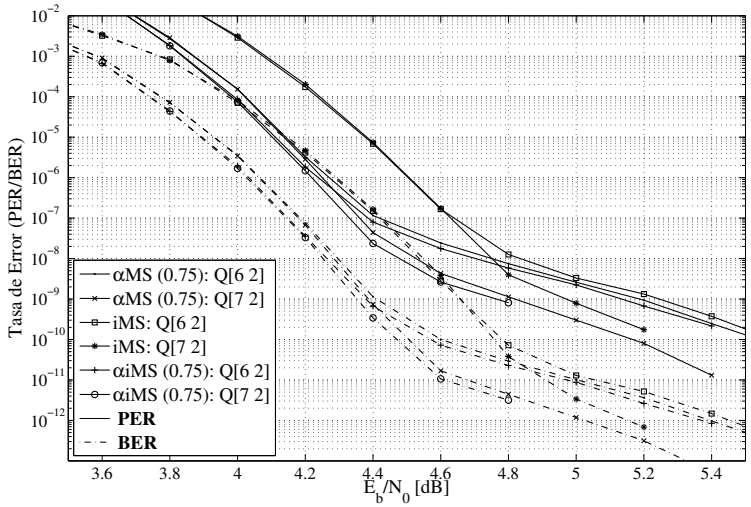


Figura 6.7: Tasa de error de los algoritmos αMS, iMS y αiMS para el código del estándar IEEE 802.3an.

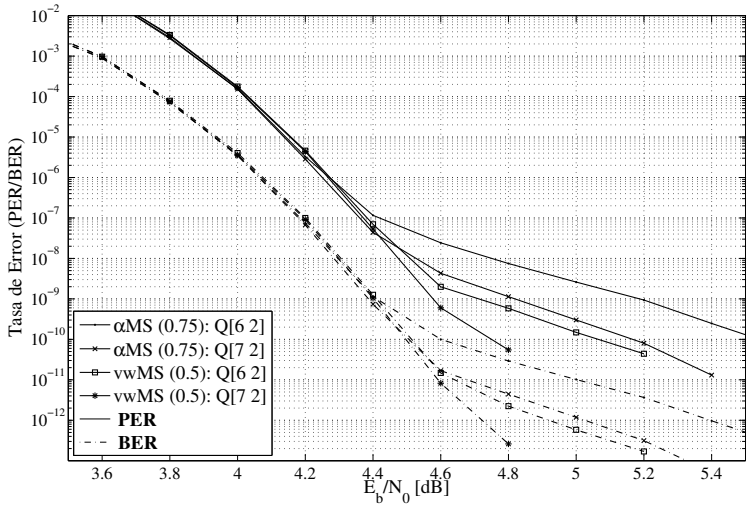


Figura 6.8: Tasa de error de los algoritmos αMS y vwMS ($\alpha = 0.5$) para el código del estándar IEEE 802.3an.

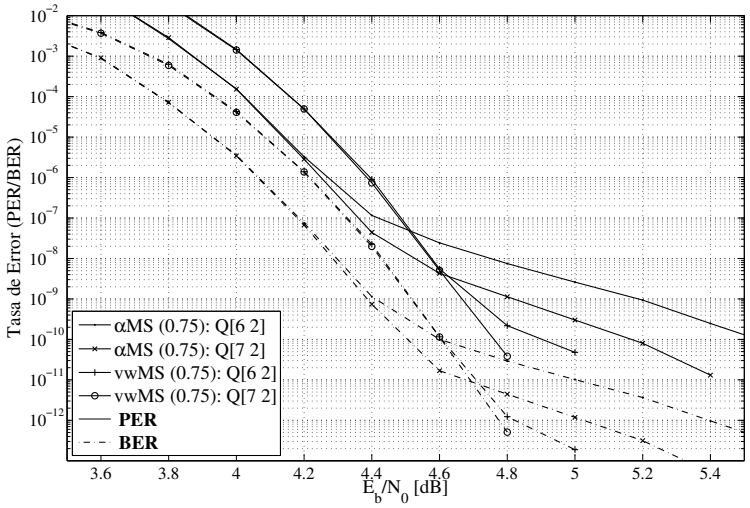


Figura 6.9: Tasa de error de los algoritmos αMS y vwMS ($\alpha = 0.75$) para el código del estándar IEEE 802.3an. En el algoritmo vwMS.

6.4. Conclusiones

En esta tesis de ha desarrollado un simulador *hardware* para poder analizar el comportamiento de los códigos LDPC y los algoritmos de decodificación en la zona de baja tasa de error (BER$< 10^{-8}$). El simulador propuesto, implementado en un dispositivo Virtex-6, alcanza una tasa de decodificación de 1.35 Gbps con un sólo *core*. En este mismo dispositivo es posible implementar hasta 4 *cores*, en cuyo caso se alcanza una tasa de decodificación de 5.4 Gbps. Estas tasas de decodificación son superiores a las conseguidas por otros simuladores *hardware* de la literatura.

Mediante el simulador *hardware* se ha analizado el comportamiento de los diferentes algoritmos estudiados en esta tesis para tasas de error muy bajas (BER$< 10^{-8}$). El estudio se ha realizado con el código LDPC del estándar IEEE 802.3an y se han probado varios esquemas de cuantificación. Los resultados obtenidos muestran que con un esquema de cuantificación $[6:2]$ las prestaciones de los algoritmos de referencia MS, αMS y βMS, y de los algoritmos propuestos iMS y αiMS son buenas para BER menores a 10^{-8}, sin embargo, por debajo de esta tasa de error aparece el suelo de error. Usando un esquema de cuantificación $[7:2]$ el suelo de error se reduce casi dos órdenes de magnitud, manteniendo las prestaciones en la zona alta de BER.

Destaca el comportamiento del algoritmo propuesto vwMS para tasas de error bajas. Con un factor de escalado $\alpha = 0.5$ el suelo de error del algoritmo vwMS está dos órdenes de magnitud por debajo del suelo de error del algoritmo αMS, usando el mismo esquema de cuantificación. Además, con este factor de escalado las prestaciones en la zona alta de la tasa de error son similares a las del algoritmo αMS. Las prestaciones del algoritmo vwMS con un factor α igual a 0.75 son peores (0.2 dB) que las del algoritmo αMS, sin embargo, en la zona baja de la tasa de error el suelo de error está tres órdenes de magnitud por debajo,

para un mismo esquema de cuantificación. No se aprecia suelo de error en el algoritmo vwMS con $\alpha = 0.75$ y esquema de cuantificación $[7:2]$ por encima de un BER de 10^{-13}.

Capítulo 7

CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

7.1. Conclusiones

En esta tesis se han investigado los algoritmos de decodificación para códigos de comprobación de paridad de baja densidad (LDPC) y las arquitecturas para la implementación *hardware* de éstos. Concretamente se han estudiado los algoritmos del tipo de intercambio de mensajes para códigos estructurados. Inicialmente se ha evaluado las prestaciones del algoritmo de referencia Sum-Product (SP) y se ha realizado el análisis de precisión finita para varios códigos LDPC (de los estándares IEEE 802.3an, IEEE 802.11n e IEEE 802.16e), concluyendo que con un esquema de cuantificación $[6:2]$ las pérdidas de las prestaciones son menores a 0.2 dB en todos los casos. También se han evaluado los algoritmos Min-Sum (MS) y sus variantes, MS escalado (αMS) y MS con *offset* (βMS). Estas variantes son ampliamente utilizadas debido a que su complejidad es baja, respecto al algoritmo SP, y las prestaciones que consiguen son similares con el mismo esquema de cuantificación ($[6:2]$). Además, la mayoría de trabajos que se encuentran en la literatura son modificaciones o se basan en el algoritmo MS.

Por otra parte se han estudiado diferentes métodos de actualización de los mensajes para los algoritmos de decodificación del tipo de paso de mensajes. Con el método de actualización básico, denominado por inundación, se requieren en general 30 iteraciones para conseguir la convergencia de los algoritmos para una tasa de error de paquete PER$= 10^{-5}$. Con los métodos de actualización por capas horizontales (*layered*) o verticales (*shuffled*) se consigue mejorar la velocidad de convergencia, necesitando aproximadamente la mitad de iteraciones para alcanzar las mismas prestaciones que con la actualización por inundación.

Tras estudiar los algoritmos y métodos de actualización de referencia existentes, se han propuesto dos nuevos algoritmos basados en MS (iMS y vwMS) con el fin de reducir el coste *hardware* de sus implementaciones sin pérdidas en las prestaciones (tasas de error), y un método de actualización (*x-shuffled*) que consigue aumentar la tasa de decodificación. Con el algoritmo iMS y su versión escalada αiMS se consigue reducir el ancho de palabra de la ruta de datos en las unidades CNP, llegando a mejorar las prestaciones de los algoritmos

αMS y βMS con precisión finita, en 0.02 dB (para un PER$= 10^{-5}$). El algoritmo vwMS requiere el cálculo de un sólo mínimo con la reducción *hardware* que esto supone. Las prestaciones de este algoritmo en precisión finita, para el código regular del estándar IEEE 802.3an, son mejores en 0.03 dB a las de los algoritmos SP y αMS. Sin embargo, para los otros códigos evaluados (todos ellos irregulares) las prestaciones de éste son peores, obteniendo unas pérdidas de aproximadamente 0.12 dB respecto al algoritmo αMS. En el método *x-shuffled* la actualización se realiza por capas verticales al igual que en el método *shuffled*, sin embargo, la actualización de los nodos de comprobación y los nodos de bit se realizan al mismo tiempo. Con este método se alcanza una velocidad de convergencia similar a la de los métodos *layered* y *shuffled* (son necesarias 2 iteraciones más), y además se consigue que la eficiencia de utilización *hardware* de las implementaciones sea del 100 %, duplicando la tasa de decodificación del método de actualización *shuffled*.

Además de realizar el estudio algorítmico y de precisión finita, se han implementado en un ASIC diferentes arquitecturas para los algoritmos de referencia estudiados y los algoritmos propuestos, así como para los diferentes métodos de actualización. En general, la finalidad de las implementaciones realizadas es conseguir tasas de decodificación del orden de Gbps, por lo que se ha utilizado el código LDPC del estándar IEEE 802.3an (especificado para 10 Gbps). En primer lugar se ha propuesto una arquitectura para el cálculo de los dos mínimos, necesario en gran parte de los algoritmos de decodificación basados en MS, con la que se reduce la complejidad *hardware* de esta operación entre el 1.6 % y el 17.4 %, comparado con otras arquitecturas de la literatura. Con la arquitectura completamente paralela, las implementaciones de los algoritmos propuestos iMS, αiMS alcanzan una tasa de decodificación de 9.4 Gbps con un área de 11.47 mm^2 y 11.50 mm^2, respectivamente. El área de éstas es un 32 % menor que la requerida por las implementaciones con los algoritmos MS, αMS y βMS y la tasa de decodificación alcanzada es un 20 % superior. Para el caso del algoritmo vwMS, la tasa de decodificación es superior un 3 % y 5 %, comparada con la conseguida con los algoritmos αMS y βMS, respectivamente, mientras que el área requerida es aproximadamente un 17 % menor. Los resultados obtenidos con la arquitectura parcialmente paralela basada en memorias muestran que con los algoritmos iMS y αiMS se requiere un 25 % menos de área comparado con las implementaciones de los algoritmos MS, αMS y βMS, y que con el algoritmo vwMS la reducción en el área es del 8 %. Con esta arquitectura la tasa de decodificación es similar para todos los algoritmos basados en MS, sobre los 340 Mbps.

La arquitectura ISMP (*improved*-SMP), propuesta en esta tesis, es una mejora de la arquitectura SMP (*Sliced-Message Passing*). La implementación del algoritmo SP con esta arquitectura alcanza una tasa de decodificación de 5.66 Gbps con un área de 9.95 mm^2, un 33 % menos de área y un 3 % más de tasa de decodificación comparado con la arquitectura original. La eficiencia de utilización *hardware* con estas arquitecturas es superior al 80 %, un 30 % más que la conseguida con la arquitectura completamente paralela. Por otra parte, se han diseñado dos arquitecturas basadas en la ISMP para implementar de forma eficiente los métodos de actualización *shuffled* y *x-shuffled*. El área requerida por las implementaciones de estas arquitecturas con el algoritmo SP es la misma (11.86 mm^2), sin embargo, la tasa conseguida con la arquitectura *x-shuffled* es un 36 % mayor, alcanzando los 15.73 Gbps. Además, la HUE de la implementación con la arquitectura *shuffled* es del 50 % mientras que con la arquitectura *x-shuffled* es de 100 %.

Las implementaciones de los algoritmos αMS, αiMS y vwMS con la arquitectura *layered* alcanzan tasas de decodificación muy altas con poca área. Con el algoritmo αMS la tasa

de decodificación es de 11.42 Gbps con un área de 4.12 mm^2. Los resultados con el algoritmo αiMS muestran que la tasa de decodifiación alcanzada es un 17 % mayor (13.80 Gbps) y que el área requerida es un 24 % menor (3.12 mm^2). Por último, con el algoritmo vwMS la tasa es de 12.84 Gbps y el área es 3.84 mm^2, mejorando los resultados de implementación del algoritmo αMS. Comparado con otras implementaciones encontradas en la literatura para el mismo código del estándar IEEE 802.3an, las implementaciones de los algoritmos αiMS y vwMS con la arquitectura *layered* son las que mejor ratio entre tasa de decodificación y área obtienen, además de ser las que menor área de implementación requieren.

Para poder evaluar el comportamiento de los códigos LDPC y algoritmos de decodificación en la zona de baja tasa de error (BER$< 10^{-8}$) se ha implementado un simulador *hardware*, que utiliza un decodificador con arquitectura parcialmente paralela basada en memorias. La tasa de decodificación alcanzada por el simulador, implementado en un dispositivo Virtex-6 para el código del estándar IEEE 802.3an y con los algoritmos basados en MS expuesto en esta tesis, es de 1.35 Gbps con un sólo *core*. Con una configuración de 4 *cores*, los cuales se implementan en un mismo dispositivos Virtex-6, se consigue una tasa de decodificación de 5.4 Gbps. Estas tasas son superiores a las conseguidas por otros simuladores *hardware* de la literatura.

Los resultados del simulador *hardware* para los algoritmos basados en MS estudiados en esta tesis (de referencia y propuestos) muestran que con el esquema de cuantificación $[6:2]$ las prestaciones de los algoritmos MS, αMS, βMS, iMS y αiMS se deterioran en tasas de error inferiores a 10^{-8}. En otra palabras, para estos algoritmos con el esquema $[6:2]$ aparece un suelo de error. Los resultados obtenidos también muestran que usando un esquema de cuantificación $[7:2]$ el suelo de error baja casi dos ordenes de magnitud (hasta $BER = 10^{-11}$) y que las prestaciones para BER altos, superiores a 10^{-7}, son similares a las obtenidas con el esquema $[6:2]$. Las prestaciones del algoritmo vwMS con un factor de escalado $\alpha = 0.5$ en la zona alta de BER son similares a las del algoritmo αMS, con el mismo esquema de cuantificación. Sin embargo, el suelo de error se encuentra dos órdenes de magnitud más abajo que el del resto de algoritmos. Este mismo algoritmo con un factor de escalado $\alpha = 0.75$ presenta unas pérdidas de 0.2 dB respecto al αMS en la zona alta de BER ($> 10^{-8}$). Sin embargo, para un mismo esquema de cuantificación el suelo de error se encuentra tres órdenes de magnitud más abajo que el del resto de algoritmos. Es más, con un esquema de cuantificación $[7:2]$ no se aprecia suelo de error para tasas de error de bit (BER) mayores a 10^{-13}.

7.2. Líneas futuras de trabajo

La degradación de las prestaciones debida a la aparición del suelo de error es un gran problema en aplicaciones que requieren tasas de error muy bajas, como son los sistemas de comunicaciones ópticas o los sistemas de almacenamiento magnético y magnetico-óptico. En esta tesis se ha desarrollado un algoritmo que desplaza el suelo del error, mejorando a los algoritmos convencionales. Sin embargo, se desconoce el porqué este algoritmo provoca este efecto. Así pues, nuestro trabajo futuro se centrará en el estudio de los fenómenos que producen el suelo de error y en el desarrollo de algoritmos para su eliminación.

Concretamente se trabajara en:

- Estudio de las técnicas existentes para la reducción del suelo de error y la aplicación de éstas en los algoritmos de baja complejidad iMS y αiMS.

- Estudio del funcionamiento del algoritmo vwMS a bajas tasas de error: análisis del comportamiento ante la presencia de "trapping sets". Utilización de este análisis para mejorar las prestaciones de este algoritmo.
- Estudiar más a fondo el comportamiento a bajas tasas de error del algoritmo vwMS: entender el porqué de la mejora en el suelo de error de este algoritmo.
- Optimización del simulador *hardware* con el fin de facilitar el estudio del suelo de error. Como por ejemplo, almacenar los patrones de datos que producen errores en la decodificación con relaciones señal a ruido altas y el estado de las unidades VNP y CNP es este mismo caso.

Otra línea de investigación que se pretende llevar a cabo es el desarrollo de algoritmo de baja complejidad y arquitecturas hardware para decodificar códigos LDPC no binarios.

BIBLIOGRAFÍA

[1] F. T. Corporation. UMC 90nm logic SP (LowK) process standard core cell library.

[2] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, jan. 1962.

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

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

[5] "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.

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

[7] "IEEE standard for local and metropolitan area networks Part 11: wireless lan medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11n-2009*, oct. 2009.

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

[9] "IEEE standard for local and metropolitan area networks Part 15.3: wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPANs)," *IEEE Std 802.15.3c-2009*, oct. 2009.

[10] "IEEE standard for local and metropolitan area networks Part 3: carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications," *IEEE Unapproved Draft Std P802.3/D2.2*, 2008.

[11] D. MacKay, S. Wilson, and M. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Transactions on Communications*, vol. 47, no. 10, pp. 1449–1454, oct. 1999.

[12] M. Luby, M. Amin Shokrolloahi, M. Mizenmacher, and D. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation," p. 117, aug. 1998.

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

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

[15] H. Jin, A. Khandekar, A. Kh, and R. J. McEliece, "Irregular repeat accumulate codes," *Proceedings of the Second International Symposium on Turbo Codes and Related Topics*, sep. 2000.

[16] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols," *IEEE Communications Letters*, vol. 7, no. 7, pp. 317–319, jul. 2003.

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

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

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

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

[21] E. Boutillon, F. Guillou, and J.-L. Danger, "λ-Min decoding algorithm of regular and irregular LDPC codes," in *Proceedings of the 3nd International Symposium on Turbo Codes Related Topics*, sep. 2003.

[22] X. Wu, Y. Song, M. Jiang, and C. Zhao, "Adaptive normalized/offset Min-Sum algorithm," *IEEE Communications Letters*, vol. 14, no. 7, pp. 667–669, jul. 2010.

[23] A. Darabiha, A. Carusone, and F. Kschischang, "A bit-serial approximate Min-Sum LDPC decoder and FPGA implementation," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, may. 2006.

[24] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Lin, "Flexible LDPC decoder design for multigigabit-per-second applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 116–124, jan. 2010.

[25] T. Mohsenin and B. Baas, "Split-Row: a reduced complexity, high throughput LDPC decoder architecture," in *International Conference on Computer Design (ICCD)*, oct. 2006, pp. 320–325.

[26] T. Mohsenin, P. Urard, and B. Baas, "A thresholding algorithm for improved Split-Row decoding of LDPC codes," in *42nd Asilomar Conference on Signals, Systems and Computers*, oct. 2008, pp. 448–451.

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

[28] J. Chen, R. Tanner, C. Jones, and Y. Li, "Improved Min-Sum decoding algorithms for irregular LDPC codes," in *International Symposium on Information Theory*, sep. 2005, pp. 449–453.

[29] J. Jin and C. ying Tsui, "An energy efficient layered decoding architecture for LDPC decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1185–1195, aug. 2010.

[30] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, feb. 2005.

[31] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, mar. 2002.

[32] T. Mohsenin and B. Baas, "High-throughput LDPC decoders using a multiple Split-Row method," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2, apr. 2007, pp. II.13–II.16.

[33] A. Darabiha, A. Chan Carusone, and F. Kschischang, "Power reduction techniques for LDPC decoders," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 8, pp. 1835–1845, aug. 2008.

[34] T. Mohsenin, D. Truong, and B. Baas, "An improved Split-Row threshold decoding algorithm for LDPC codes," in *IEEE International Conference on Communications*, jun. 2009, pp. 1–5.

[35] T. Mohsenin and B. M. Baas, "A split-decoding message passing algorithm for low density parity check decoders," *Journal of Signal Processing Systems*, vol. 61, no. 3, pp. 329–345, feb. 2010.

[36] N. Onizawa, T. Hanyu, and V. Gaudet, "Design of high-throughput fully parallel LDPC decoders based on wire partitioning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 3, pp. 482–489, mar. 2010.

[37] T. Mohsenin, D. N. Truong, and B. M. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 5, pp. 1048–1061, may. 2010.

[38] L. Liu and C.-J. Shi, "Sliced message passing: high throughput overlapped decoding of high-rate low-density parity-check codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3697–3710, dec. 2008.

[39] T. Zhang and K. Parhi, "A 54 Mbps (3,6)-regular FPGA LDPC decoder," in *IEEE Workshop on Signal Processing Systems (SIPS)*, oct. 2002, pp. 127–132.

[40] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for Quasi-Cyclic LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 1, pp. 104–114, jan. 2007.

[41] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 μm CMOS process," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 672–683, mar. 2008.

[42] C.-H. Liu, S.-W. Yen, C.-L. Chen, H.-C. Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, mar. 2008.

[43] M.-A. Chao, J.-Y. Wen, X.-Y. Shih, and A.-Y. Wu, "A triple-mode LDPC decoder design for IEEE 802.11n system," in *IEEE International Symposium on Circuits and Systems, 2009 (ISCAS)*, may. 2009, pp. 2445–2448.

[44] S.-M. Kim, C.-S. Park, and S.-Y. Hwang, "A novel partially parallel architecture for high-throughput LDPC decoder for DVB-S2," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 820–825, may. 2010.

[45] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory system optimization for FPGA-based implementation of Quasi-Cyclic LDPC codes decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 1, pp. 98–111, jan. 2011.

[46] Y.-L. Ueng, C.-J. Yang, K.-C. Wang, and C.-J. Chen, "A multimode shuffled iterative decoder architecture for high-rate RS-LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 10, pp. 2790–2803, oct. 2010.

[47] Z. Cui, Z. Wang, and X. Zhang, "Reduced-complexity column-layered decoding and implementation for LDPC codes," *IET Communications*, vol. 5, no. 15, pp. 2177–2186, sep. 2011.

[48] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976–996, dec. 2003.

[49] K. Gunnam, G. Choi, M. Yeary, and M. Atiquzzaman, "VLSI architectures for layered decoding for irregular LDPC codes of WiMAX," in *IEEE International Conference on Communications (ICC)*, jun. 2007, pp. 4542–4547.

[50] K. Zhang, X. Huang, and Z. Wang, "A high-throughput LDPC decoder architecture with rate compatibility," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 839–847, apr. 2011.

[51] K. Gunnam, G. Choi, and M. Yeary, "A parallel VLSI architecture for layered decoding for array LDPC codes," in *20th International Conference on VLSI Design and 6th International Conference on Embedded Systems*, jan. 2007, pp. 738–743.

[52] Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "An efficient 10Gbase-T ethernet LDPC decoder design with low error floors," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, apr. 2010.

[53] A. Cevrero, Y. Leblebici, P. Ienne, and A. Burg, "A 5.35 mm2 10Gbase-T ethernet LDPC decoder chip in 90nm CMOS," in *IEEE Asian Solid State Circuits Conference (A-SSCC)*, nov. 2010, pp. 1–4.

[54] D. Bao, X. Chen, Y. Huang, C. Wu, Y. Chen, and X. Y. Zeng, "A single-routing layered LDPC decoder for 10Gbase-T ethernet in 130nm CMOS," in *17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, feb. 2012, pp. 565–566.

[55] F. Angarita, T. Sansaloni, M. Canet, and J. Valls, "Improved sliced message passing architecture for high throughput decoding of LDPC codes," *Journal of Signal Processing Systems*, vol. 66, no. 2, pp. 99–104, feb. 2012.

[56] F. Angarita, T. Sansaloni, A. Perez-Pascual, and J. Valls, "Modified shuffled based architecture for high-throughput decoding of LDPC codes," *Journal of Signal Processing Systems*, vol. 68, pp. 139–149, 2012, 10.1007/s11265-011-0592-z.

[57] C.-L. Wey, M.-D. Shieh, and S.-Y. Lin, "Algorithms of finding the first two minimum values and their hardware implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3430–3437, dec. 2008.

[58] Q. Xie, Z. Chen, X. Peng, and S. Goto, "A sorting-based architecture of finding the first two minimum values for LDPC decoding," in *IEEE 7th International Colloquium on Signal Processing and its Applications (CSPA)*, mar. 2011, pp. 95–98.

[59] L. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2342–2346, dec. 2012.

[60] D. E. Knuth, *The art of computer programming, volume 3: (2nd ed.) sorting and searching (section 5.3.3)*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[61] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits- a design perspective*, 2nd ed. Prentice Hall, 2004.

[62] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Investigation of error floors of structured LDPC codes by hardware emulation," in *IEEE Global Telecommunication Conference*, dec. 2006, pp. 1–6.

[63] Y. Cai, S. Jeon, K. Mai, and B. Kumar, "Highly parallel FPGA emulation for LDPC error floor characterization in perpendicular magnetic recording channel," *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3761–3764, oct. 2009.

[64] X. Chen, J. Kang, S. Lin, and V. Akella, "Accelerating FPGA-based emulation of quasi-cyclic LDPC codes with vector processing," in *Design, Automation Test in Europe Conference*, apr. 2009, pp. 1530–1535.

[65] H. Li, Y. S. Park, and Z. Zhang, "Reconfigurable architecture and automated design flow for rapid FPGA-based LDPC code emulation," in *Proceedings of the ACM/SIGDA*, feb. 2012, pp. 167–170.

[66] R. Gutierrez, V. Torres, and J. Valls, "Hardware architecture of a Gaussian noise generator based on inversion method," *IEEE Transactions on Circuits Systems II*, to be published 2012.

[67] P. L'Ecuyer, "Maximally equidistributed combined Tausworthe generators," *Mathematics of Computation*, vol. 65, no. 213, pp. 203–213, 1996.