UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN EN FPGA DE UN DECODIFICADOR DE CÓDIGOS LDPC CON SECUENCIACIÓN POR CAPAS

TRABAJO FINAL DE GRADO

AUTOR:
Ferran Mascarell Català

DIRECTORES:
Javier Valls Coquillat
María José Canet Subiela

Gandía, 27 de junio de 2013.

RESUMEN

Los códigos de comprobación de paridad de baja densidad (LDPC, Low-density Parity-check) están siendo incluidos en multitud de estándares de comunicaciones debido a su gran capacidad de corrección y a su facilidad para paralelizar el proceso de decodificación, lo que permite su uso en sistemas que requieran altas velocidades de transmisión. Este trabajo se centra en el desarrollo de la arquitectura con secuenciación por capas para decodificar códigos LDPC regulares a velocidades de Gbps utilizando un algoritmo de decodificación basado en el volteo de bits con pesos (WBF Weighted Bit Fliping). Concretamente se ha partido de la versión mejorada del algoritmo WBF (IWBF, Improved WBF) y se le ha adaptado la actualización por capas horizontales (*layered*) aplicados a los algoritmos basados en suma-producto (SP) para el código LDPC del estándar 10GBase-T. Se ha diseñado la arquitectura layered, y se ha codificado en VHDL e implementado en dispositivos FPGA alcanzando una velocidad de 2,048 Gbps con 10 iteraciones.

Palabras clave: LDPC, FPGA, Decodificador

ABSTRACT

The Low-Density Parity-Check codes (LDPC) have been included in most of the comunicaction standars mainly due to two facts. First, their error correction capability is very high, making the data transmision rate close to the limit established by Shannon; second, the decoding algorithms are highly parallel, so high throughput decoders (Gbps) can be implemented. This work is focused on developing a layered decoder architecture for LDPC for the Weighted Bit-Flipping algorithms (WBF), specifically the Improved weighted Bit-Flipping (IWBF). The decoder was implemented for the LDPC code of the 10GBase-T standard, which is suitable for high throughput applications. The proposed architecture was coded in VHDL and iplemented in different FPGA devices, achieving a throughput up to 2,048 Gbps with 10 decoding iterations.

Keywords: LDPC, FPGA, Decoder

ÍNDICE GENERAL

Capítulo 1

INTRODUCCIÓN

Los códigos de comprobación de paridad de baja densidad (LPDC, Low-Density Parity- Check) [1] son códigos de bloque lineales definidos por una matriz de paridad dispersa (baja densidad de unos). Fueron inventados por Robert Gallager en 1962, pero debido a la alta complejidad computacional que suponía su decodificación con la tecnología de entonces, fueron olvidados hasta que David MacKay los redescubrió a mitad de los años 90. Hoy en día, el uso de estos códigos está en pleno auge ya que sus prestaciones pueden aproximarse bastante al límite de capacidad de Shannon.

En contraste con códigos propuestos posteriormente a su invención, los códigos LDPC ofrecen mejores prestaciones y altas posibilades de paralelización, por lo que posibilita su implementación a altas velocidades de decodificación y por ello han sido incluidos en muchos estándares de comunicaciones: WiMAX (IEEE 802.16e), DVB-S2 (Digital Video Broadcasting) y 10GBase-T Ethernet (IEEE 802.3) [2].

Existen diversos algoritmos de decodificación de códigos LDPCs como el Suma-Producto (SP) [3] y su simplificación Min-Sum (MS) [3]. Estos algoritmos consiguen buenas prestaciones pero requieren una complejidad de diseño importante. Sin embargo, en sistemas de comunicación que ofrecen un canal confiable y una alta tasa de datos y de decodificación, se permite el uso de algoritmos de bajo coste que requieren únicamente operaciones lógicas simples y ofrecen un excelente compromiso entre complejidad, tasa de error y velocidad de decodificación. Algoritmos basados en Bit Flipping (BF) [1], [4] como Bit Flipping con pesos (WBF) [4], [5], Bit Flipping con pesos Modificado (MWBF) [6] y Bit Flipping con pesos Modificado Mejorado (IMWBF) [7] se adecuan a estas condiciones.

1.1. Objetivos

El objetivo general de esta proyecto final de carrera es la de implementación de un decodificador basado en el algoritmo IWBF basada en la arquitectura layered (TIWBF) en una FPGA. Para alcanzar dicho objetivo se proponen las siguientes tareas:

- Estudio de los algoritmos basados en BF.
- Modelado en matlab del algoritmo de decodificación.
- Optimización de los parámetros del algoritmo de decodificación utilizando el modelo en matlab.
- Diseño e implementación de la arquitectura hardware del decodificador con la arquitectura layered.
- Test del decodificador con la arquitectura layered.

1.2. Organización del trabajo

Este trabajo está organizado de la siguiente manera:

En el Capítulo 2 se introducen los códigos LDPC y se resumen sus principales características. También se presenta los códigos LDPC estructurados y el código concreto utilizado para dicho trabajo.

En el Capítulo 3 se describen los algorítmos de decodificación LDPC más relevantes y los métodos de actualización más importantes.

En el Capítulo 4 se explica el entorno de simulación y modelado del algoritmo implementado. Así como, el proceso seguido para el ajuste de los diferentes parámetros de éste algoritmo.

En el Capítulo 5 se trata la implementación *hardware* del algoritmo diseñado para la arquitectura *layered* para el código LDPC del estándar IEEE 802.3an.

Capítulo 2

CÓDIGOS LDPC

La idea básica de la codificación para la corrección de errores es la de aumentar el número de bits del mensaje introduciendo redundancia para crear una palabra código para éste. Estas palabras código son lo suficientemente diferentes entre sí para permitir que el mensaje transmitido llegue correctamente al receptor, incluso cuando algunos bits de esta palabra código se hayan dañado durante la transmisión por el canal.

Un sencillo sistema de codificación son los códigos de paridad simple (*single parity − check code*). Este código añade un bit extra, llamado bit de paridad, al mensaje. Este bit depende del contenido del mensaje. Un código de paridad par indica si el número de unos es par o impar. Si es par el bit tiene valor 0 y si es impar 1.

Mientras que el cambio de un solo bit se puede detectar fácilmente por un código de paridad simple, este código no es lo suficientemente potente para indicar que bits fueron dañados, ni detectar errores cuando el número de errores es par. Para corregir esto, se requieren códigos más sofisticados que contengan múltiples ecuaciones de comprobación de paridad, donde cada una de las cuales satisfaga la palabra código. Esto se obtiene de la codificación por medio de matrices de paridad ya que pueden ser usadas para verificar si un vector c es una palabra código válida.

Sea un código de comprobación de paridad c de longitud N y K el número de bits del mensaje, que debe satisfacer un conjunto de M restricciones de comprobación de paridad. Este código está definido exclusivamente por una matriz de paridad H de tamaño MxN, donde cada una de las M filas especifica cada una de las restricciones. Este código c es el conjunto de vectores binarios que satisfacen todas las restricciones, es decir, $c \cdot H^T = 0$, donde H^T es la matriz transpuesta. En general, el rango de una matriz de paridad es $rango(H) = N - K$, el cual indica el número de filas linealmente independientes. Cuando éste es igual a M, número de filas de la matriz, cada restricción es linealmente independiente. En cuanto al número de palabras código posibles viene dado por $n^o c = 2^{N-rango(H)} = 2^K$. Mientras que su tasa será $r = K/N$.

2.1. Grafo de Tanner

La matriz de paridad H se puede representar gráficamente mediante el denominado $grafo\ de\ Tanner$ que tiene N nodos de bit (*bit nodes*), uno por cada bit, y M nodos de comprobación (*check nodes*), uno para cada comprobación de paridad. Los nodos de comprobación están conectados a los bits que deben comprobar. Específicamente, una rama conecta el nodo de comprobación m con el nodo de bit n si y sólo si la m-ésima comprobación de paridad involucra el n-ésimo bit. Esto significa que H es la matriz adyacente del grafo. Un ciclo es un trayecto del grafo que empieza y termina en el mismo nodo de bit y su longitud será el número de nodos que atraviesa. La desventaja de tener ciclos en un grafo es que al pasar los mensajes se pueden quedar *."trapados"* en el ciclo, realimentado siempre la misma infromación.

Para el caso de un código LDPC regular, cada bit está involucrado en j comprobaciones de paridad. De ahí que el número de ramas que salen de un nodo de bit siempre es j. De igual manera, ya que cada comprobación de paridad involucra k bits, el número de ramas que salen de cada nodo de comprobación siempre es k.

El conjunto de bits que participan en el nodo de comprobación (*check node*) m se denotan como $N(m) = \{n : Hmn = 1\}$. Y el conjunto de comprobaciones en las cuales el bit n, también llamado nodo variable (*bit node*), participa se denota como $M(n) = \{m : Hmn = 1\}$. Finalmente $N(m)/n$ denota que el bit n está excluido del conjunto $N(m)$.

El grafo de Tanner asociado para la matriz de comprobación de paridad de la figura 2.2 es el mostrado en la figura 2.1. En este grafo hay tantos nodos de bit como columnas en la matriz y tantos nodos de comprobación de paridad como filas. Así que el grafo para dicha matriz tiene 12 nodos de comprobación de paridad y 16 nodos de bit. Mientras que la interconexión entre los nodos viene dada por la posición de los unos en la matriz de comprobación de paridad. Por ejemplo, el nodo de comprobación c_1 se conectan a los nodos de bit v_1, v_6, v_12 y v_13 ya que en la matriz $N(1) = \{1, 6, 12, 13\}$. Mientras que la conexión del nodo de bit v1 se conectan a los nodos de comprobación $c1$, $c8$ y $c10$ dado que en la matriz $M(1) = \{1, 8, 10\}$.

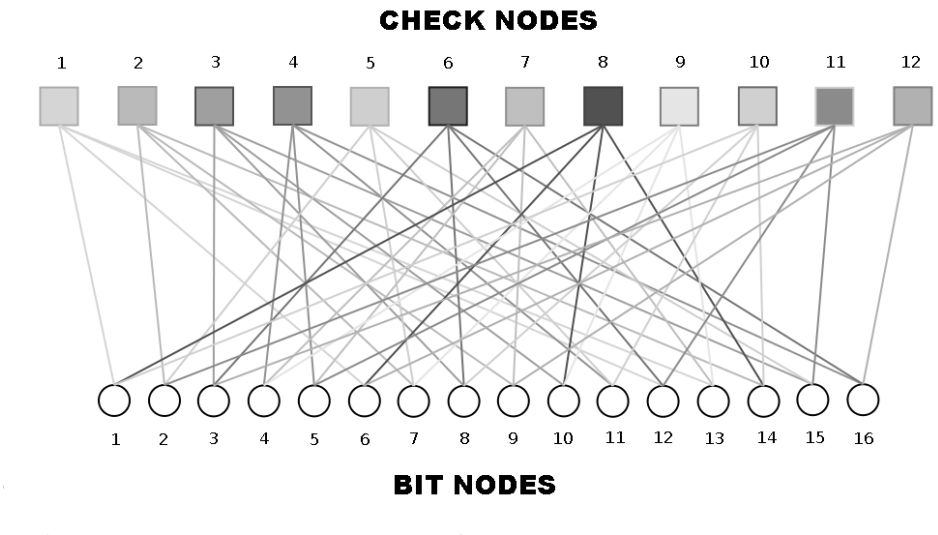


Figura 2.1: Grafo de Tanner para un código LDPC (3,4) de orden 12x16.

2.2. Tipos de código

Por definición un código de comprobación de paridad de baja densidad es un código de bloque lineal que tiene una matriz de paridad dispersa. Con esto se quiere decir que la cantidad de unos existentes en esta matriz es mucho menor comparada con la cantidad de ceros.

Una matriz LDPC regular (j,k) es una matriz binaria de orden MxN que tiene j unos en cada columna y k unos en cada fila. Donde $j < k$ y además ambos son más pequeños comparados con N. Esta limitación es necesaria para asegurar que todas las palabras código nulas satisfagan todas las restricciones. Los índices j y k indican el peso de las columnas y filas respectivamente (w_c, w_r). El cual se refiere al número de elementos distintos de cero que existen en la matriz. Así que, el número de unos en la matriz H es $M \cdot k = N \cdot j$ y la tasa del código es $r = K/N = 1 - M/N$. En la figura 2.2 se muestra un ejemplo de una matriz LDPC regular $(3,4)$ de orden $12x16$.

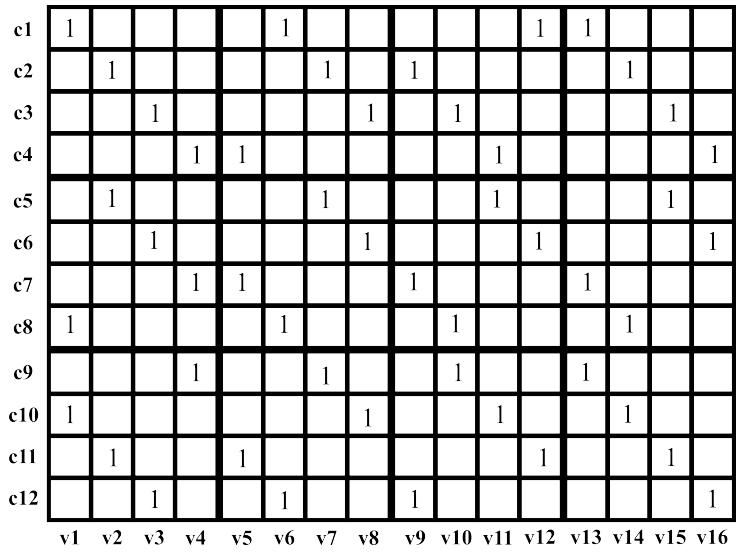


Figura 2.2: Matriz LDPC regular (3,4) de orden 12x16.

Por otra parte, una matriz LDPC irregular es aquella en que no todas las columnas y filas tienen el mismo número de unos y por lo tanto distintos j y k, sin embargo esta matriz aún sigue siendo dispersa.

2.3. Código LDPC estructurado del estándar 10GBase-T

La matriz utilizada para el decodificador es la del estándar IEEE 802.3an (10Gbase-T) [2], también llamada RsBased (2048,1723). Se genera a partir de códigos Reed-Solomon, y son estructuradas y regulares. Una matriz estructurada es aquella que se puede dividir en sub-matrices cuadradas y regulares. Para el caso de la matriz del estándar IEEE 802.3an la matriz de paridad, cuyo tamaño es 384x2048, se puede dividir en sub-matrices de tamaño 64x64 de peso unitario, por tanto el número de sub-matrices es 32 por fila y 6 por columna. Resumiendo, las principales características de esta matriz son: matriz LDPC regular binaria de orden 384x2048 que tiene 6 unos en cada columna y 32 unos en cada fila. Y la su tasa de código es $r = K/N = 1723/2048 = 5/6$.

En la figura 2.3 se muestra la representación de la matriz RsBased (2048,1723), donde se representan los unos como puntos, mientras que los ceros se representan en blanco. Tal y como se aprecia en la figura, es una matriz dispersa, ya que el número de unos de la matriz es bajo respecto al número de ceros.

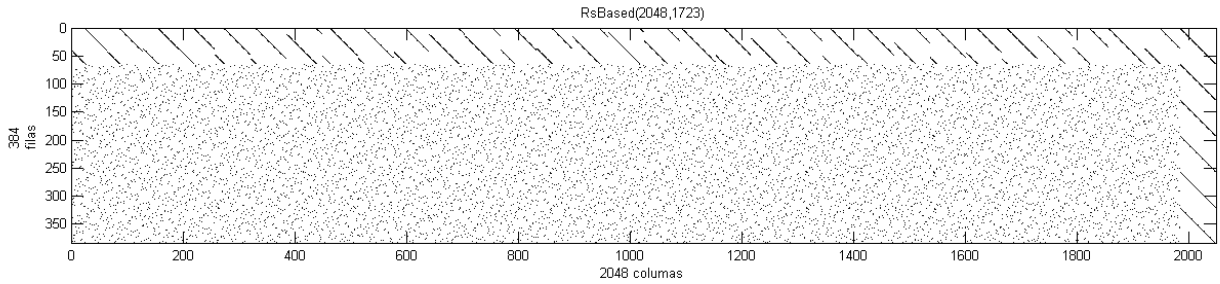


Figura 2.3: Representación de la matriz RsBased(20148,1723).

Capítulo 3

ALGORITMOS DE DECODIFICACIÓN LDPC

En este capítulo se presenta una breve introducción del sistema de comunicaciones utilizado y su nomenclatura. A continuación se hace un repaso de los algoritmos de decodificación LDPC, empezando por los algoritmos basados en SP, ya que son considerados como referencia en la literatura, y siguiendo con los algoritmos basados en Bit Flipping, explicando sus pertinentes modificaciones y mejoras de cada uno, hasta llegar al algoritmo elegido.

Sea $c = (c_1, c_2, ..., c_N)$ una palabra código binaria de un código LDPC regular de longitud N definida como el espacio nulo de una matriz de paridad dispersa H que tiene M columnas y N filas. Se supone el uso de una modulación BPSK transmitida sobre un canal $AWGN$. Al código c se le asigna una secuencia bipolar $s = (s_1, s_2, ..., s_N)$, donde $s_n = 2 \cdot c_n - 1$ y el vector de valores reales recibido es $y = (y_1, y_2, ..., y_N)$, en el cual $y_n = s_n + n_n$, donde n es el ruido gaussiano aditivo de media cero y varianza $\sigma = No/2$. En la figura 3.1 se puede ver el sistema de comunicación descrito anteriormente.

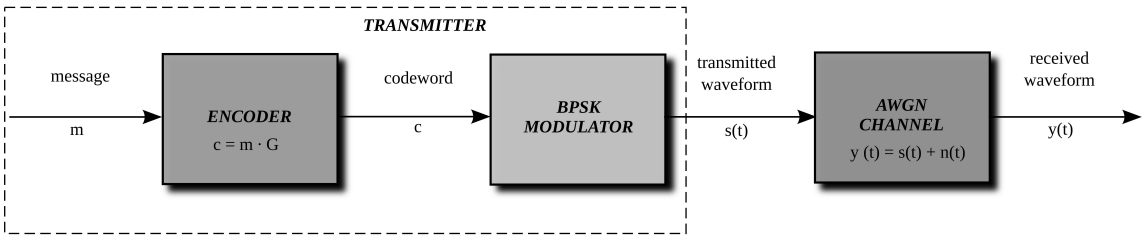


Figura 3.1: Diagrama de un sistema básico de comunicaciónes.

3.1. Intercambio de mensajes

Los algoritmos de decodificación de códigos LDPCs se basan en el intercambio de mensajes, en esta técnica de decodificación la actualización de los mensajes se dividen en dos fases: actualización de los nodos de comprobación (*check nodes*) y actualización de los nodos de bit (*bit nodes*).

En la actualización de los nodos de comprobación o actualización horizontal, los nodos de comprobación generan sus mensajes (s_m) a partir de los mensajes recibidos de los nodos de bits (z_n). La actualización de un nodo concreto se representa mediante una línea horizontal sobre la fila correspondiente a dicho nodo en la matriz de comprobación de paridad.

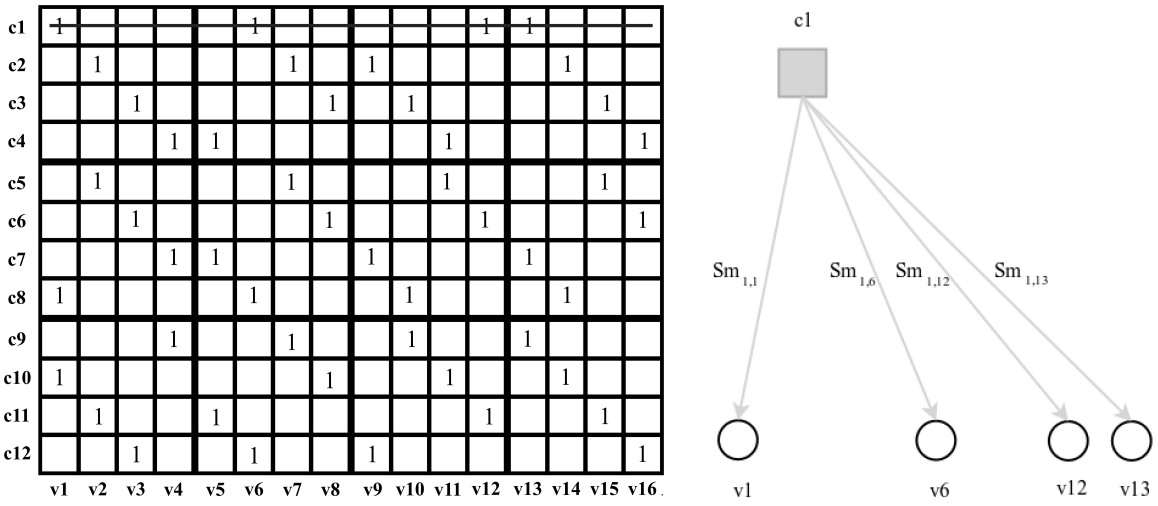


Figura 3.2: Representación de la actulización del nodo c_1 y su grafo de Tanner asociado.

En la actualización de los nodos variables o actualización vertical, los nodos variables generan sus mensajes (z_n) a partir de los mensajes recibidos de los nodos de comprobación (s_m). La actualización de un nodo concreto se representa mediante una línea vertical sobre la columna correspondiente a dicho nodo.

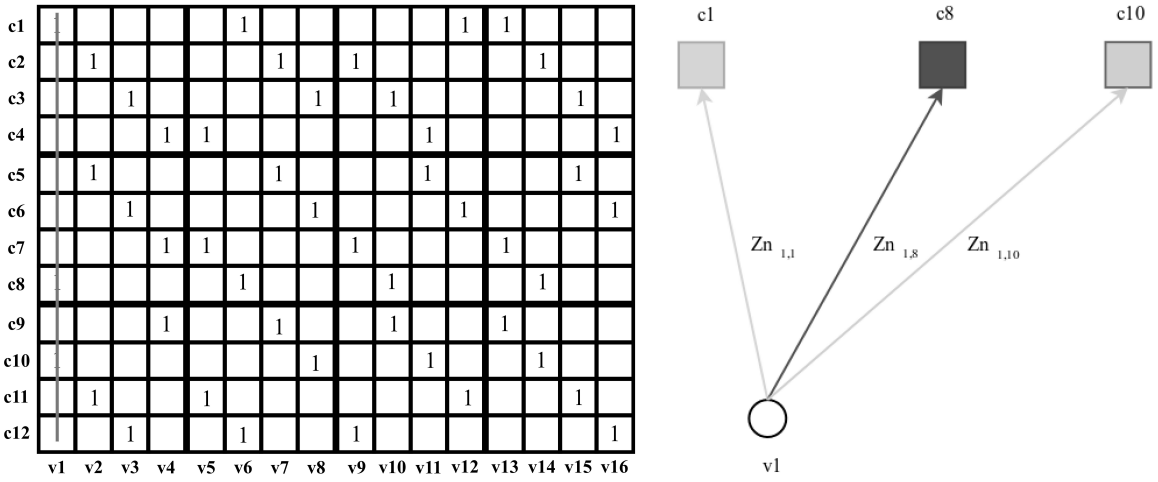


Figura 3.3: Representación de la actulización del nodo v_1 y su grafo de Tanner asociado.

3.2. Suma Producto (SP)

Este es el algoritmo de decodificación de códigos LDPC más popular, ya que sus prestaciones son óptimas. Aunque existen varias simplificaciones para el algoritmo, continúa siendo considerado una referencia en la literatura.

El algoritmo SP [3] es un algoritmo iterativo de decodificación soft basado en la propagación de la creencia siendo extremadamente eficiente para decodificar códigos LDPC. El algoritmo está basado en el intercambio de mensajes, en esta técnica de decodificación los mensajes son actualizados por cada nodo de comprobación y enviados a sus respectivos nodos de bit y viceversa. También se puede decir que es un algoritmo recursivo que siempre converge en una relación logarítmica de la probabilidad (LLR) a posteriori después de que un número

finito de mensajes hayan sido enviados.

El funcionamiento del SP es el siguiente: en la fase de inicialización, el algoritmo determina los $\mu_{m,n}^{(0)}$ y $\lambda_{m,n}^{(0)}$ iniciales para la correcta puesta en marcha del algoritmo mediante (3.1) y (3.2) respectivamente, donde en esta última $\sigma = No/2$ no es más que la varianza de ruido gaussiano del canal. En el proceso iterativo, en la l-ésima iteración los nodos de comprobación envían sus mensajes ($\mu_{m,n}^{(l)}$) (3.3) a sus nodos variables correspondiente. Mientras que, los nodos variables envían sus mensajes ($\lambda_{m,n}^{(l)}$) (3.5) a sus nodos de comprobación correspondientes. Estos últimos mensajes, no son más que una estimación del LLR. El proceso iterativo continúa hasta que satisfaga una condición de salida o se llegue al número de iteraciones máximo (It_{max}). Finalmente, basándose en las medidas de confianza calculadas de cada símbolo, se realiza una decisión hard para obtener la palabra código z_n mediante (3.6) y (3.7).

La decodificación con el algoritmo SP consiste en los siguientes pasos:

Fase de inicialización:

$$\mu_{m,n}^{(0)} = 0, \forall m \in \{1,...,M\} \cap n \in N(m) \tag{3.1}$$

$$\lambda_{m,n}^{(0)} = l_n = \frac{2}{\sigma^2} \cdot r_n, \forall n \in \{1,...,N\} \tag{3.2}$$

Parte iterativa ($l = 1,2,...,It_{max}$):

1. Actualización del nodo de comprobación:

$$\mu_{m,n}^{(l)} = \Gamma_{m,n} \cdot \Psi \left(\sum_{n' \in N_{m \setminus n}} \Psi \left(\left| \lambda_{n',m}^{(i-1)} \right| \right) \right), \forall m \in \{1,...,M\} \cap n \in N_m \tag{3.3}$$

dónde:

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

$$\Psi = \log \left(\tanh \left(\frac{|x|}{2} \right) \right) \tag{3.4}$$

2. Actualización del nodo de bit:

$$\lambda_{m,n}^{(l)} = l_n + \sum_{m' \in M_{n \setminus m}} \mu_{m',n}^{(l)}, \forall n \in \{1,...,N\} \cap m \in M_n \tag{3.5}$$

3. Repetir los pasos 1 y 2 hasta que $s = 0$ o se llegue al número máximo de iternaciones(It_{max}).

Decisión hard, para la el cálculo de la salida decodificada:

$$\lambda_{m,n}^{(l)} = l_n + \sum_{m' \in M_n} \mu_{m',n}^{(l_{end})} \tag{3.6}$$

$$z_n = sign(\lambda_n) \tag{3.7}$$

3.3. Min sum (MS)

El algoritmo Min-Sum (MS) [3] es una simplificación de la computación realizada en el nodo de comprobación (3.3). Esta modificación es simplemente una aproximación del cálculo de $\Psi(x)$ (3.4). De modo, que ahora el cálculo del nodo de comprobación únicamente tiene en cuenta las entradas de λ que tengan la mínima magnitud. Es decir, se requiere determinar los dos LLR entrantes mínimos (3.8).

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

El algoritmo MS disminuye la complejidad del cálculo de los LLR pero a costa de la degradación de las prestaciones de la decodificación. Así que, se introducen dos mejoras o variantes del MS que proporcionan prestaciones cercanas a una decodificación óptima sin aumentar la complejidad. Estas variantes son MS normalizado y MS con offset. La primera variante introduce un factor de corrección β que se resta al valor mínimo obtenido (3.9). La segunda es la introducción de un factor α multiplicando a los términos de la ecuación tal y como se observa en (3.10).

$$\mu_{m,n}^{(l)} = \Gamma_{m,n} \cdot max\left(\min_{n' \in N_{m \setminus n}} (|\lambda_{n',m}^{(i-1)}|) - \beta, 0 \right) \tag{3.9}$$

$$\mu_{m,n}^{(l)} = \Gamma_{m,n} \cdot \alpha \cdot \min_{n' \in N_{m \setminus n}} (|\lambda_{n',m}^{(i-1)}|) \tag{3.10}$$

3.4. Bit Flipping (BF)

Este algoritmo es bastante sencillo ya que sólo requiere operaciones lógicas simples. Lo primero, en la fase de inicialización, se toma una decisión hard (3.11) con los datos recibidos del canal. A continuación, en la parte iterativa, primero se realiza el cálculo de los síndromes mediante la ecuación (3.12), el paso siguiente es hacer el flip del bit z_n que pertenezca a más de un determinado umbral δ de comprobaciones de paridad insatisfechas. δ depende del SNR, del peso de columna y fila. Y por eso debe escogerse de tal manera que minimice la tasa de error y el número de cálculos de comprobaciones de paridad. Este tipo de decodificación se mejorará posteriormente en otros algoritmos usando umbrales adaptativos a cambio de más cálculos por iteración.

Fase de inicialización, decisión hard:

$$z_n = \begin{cases} 1, & si \quad y_n \leq 0 \\ 0 & si \quad y_n > 0 \end{cases} \tag{3.11}$$

Cálculo de síndromes:

$$s = s_1, s_2, ..., s_M = z_n \cdot H^T \quad donde \quad s_m = \sum_{n=1}^{N} z_n \cdot h_{mn} \tag{3.12}$$

definición de Fn:

$$F_n = \{s_m : s_m = 1; \quad m \in M(n)\}$$

Los pasos de decodifcación de algoritmo BF son los siguientes:

1. Cálculo de síndromes.
2. Flip z_n cuando $|F_n| \geq \delta \; \forall \; n = 1, 2, ..., N$.
3. Repetir los pasos 1 y 2 hasta que $s = 0$ o se llegue al número máximo de iternaciones (It_{max}).

3.5. Bit Flipping con pesos (WBF)

El rendimiento del algoritmo BF estándar puede ser mejorado mediante el uso de una medida de fiabilidad de valores soft de los símbolos recibidos. Esta medida de fiabilidad o confianzas se determina con $|y|_{min-m}$ (3.13). Cuando más grande sea $|y_n|$ mayor será la confianza del bit z_n. El valor de $|y|_{min-m}$ se obtiene con:

$$|y|_{min-m} = \sum_{n:n\in N(m)} |y|_n \ \forall\ m = 1,2,...,M \tag{3.13}$$

donde $|y|_{min-m}$ es la confianza de la m-ésima ecuación de paridad. Este valor es utilizado para hacer una ponderación de las comprobaciones de paridad E_n (3.14) de cada bit. Si este valor es alto, indicará que la confianza de este bit es baja. Así que, se tomará la decisión de flip si supera el umbral δ. Este valor de E_n, decrecerá a medida que la decodificación converja.

$$E_n = \sum 2 \cdot (s_m - 1) \cdot |y_n|_{min-m} \tag{3.14}$$

Este algoritmo requiere operaciones con números reales para el cálculo de la ponderación de comprobaciones de paridad E_n en la toma de decisiones. Esto supone una dificultad y una mayor complejidad ya que se tiene que almacenar un número real para cada comprobación.

Los pasos de decodificación del algoritmo WBF son:

1. Cálculo de síndromes.
2. Cálculo de E_n.
3. Flip z_n para $n = arg\ max_{1\leq n\leq N}$.
4. Repetir los pasos anteriores hasta que $s = 0$ o se llegue al número máximo de iternaciones (It_{max}).

3.6. Bit Flipping con pesos cuantificado (QWBF)

Desde el punto de vista práctico, la implementación del decodificador WBF es problemática, o prácticamente inviable, ya que se tiene que almacenar un número real para cada comprobación y luego ser utilizados para el cálculo de los valores E_n. Para solucionar este problema, en el QWBF, se le asigna a cada bit una fiabilidad *alta* o *baja* dependiendo de un umbral Δ_1.

Los pasos de decodificación del algoritmo QWBF son:

1. Cálculo de síndromes.
2. Cálculo de E_n.
3. Flip z_n para $z_n \forall \quad E_n > \Delta_2, \quad n = 1,2,...,N$.
4. Repetir los pasos anteriores hasta que $s = 0$ o se llegue al número máximo de iternaciones (It_{max}).

Se ha de tener en cuenta que en la decodificación QWBF, los valores de los umbrales Δ_1 y Δ_2 se obtienen mediante simulaciones, y son elegidos empíricamente.

Comparado con el decodificador WBF, el QWBF tiene la ventaja de solo necesitar un bit para el almacenamiento de la fiabilidad. Por estas razones, el QWBF es más viable para la implementación que el algoritmo de decodificación anteriormente citado.

3.7. Bit Flipping con pesos mejorado (IWBF)

Para obtener aún un mayor incremento de la capacidad de decodificación, el algoritmo IWBF toma en la fase de inicialización la ecuación (3.15) para el cálculo de las confianzas.

$$W_{nm} = \min_{n:n\in N_{(n)\backslash n}} |y_n|, \quad m[1,M], \quad n \in N(m) \tag{3.15}$$

En esta ecuación se busca que, en el paso de mensajes, la confianza transmitida excluya la información del símbolo en cuestión, por lo que para cada bit n, se excluye ese mismo bit, y la confianza de la m-ésima ecuación de paridad debe tomar su valor sin tener en cuenta ese bit. Mientras que ahora el cálculo de E_n viene dado por:

$$E_n = \sum 2 \cdot (s_m - 1) \cdot W_{nm}, \quad n \in N(m)$$

Algoritmo IWBF:

1. Cálculo de síndromes.
2. Cálculo de E_n.
3. Flip z_n para $z_n \forall \quad E_n > \Delta_2, \quad n = 1,2,...,N$.
4. Repetir los pasos anteriores hasta que $s = 0$ o se llegue al número máximo de iternaciones (It_{max}).

3.8. Métodos de 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 [3], [4] y el método por capas horizontales o layered [8], [9], [10]. La actualización por inundación es la forma básica mientras la actualización layered es una mejora que acelera la convergencia de los algoritmos.

3.8.1. Actualización por inundación - Flooding

En el método por inundación ($flooding$) las actualizaciones de los nodos de comprobación y de bit se realiza en dos fases. En la primera fase todos los nodos de comprobación Cm calculan los mensajes (S_m) que enviarán a los nodos de bit y en la segunda todos los nodos de bit Vn calculan los mensajes (z_n) que van desde los nodos de bit a los nodos de comprobación.

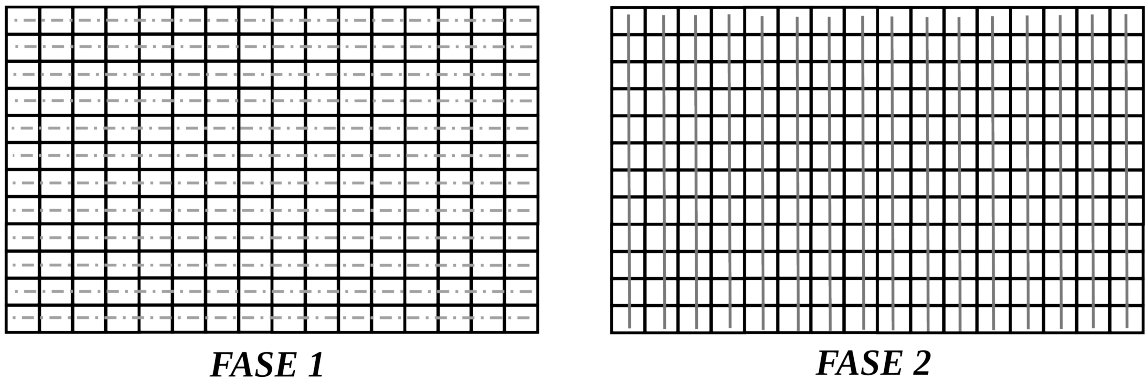


Figura 3.4: Método de actualización por inundación para la matriz LDPC regular (3,4) de orden 12x16.

La representación gráfica del método de actualización por inundación para una matriz de paridad de tamaño 12x16 se muestra en la Figura 3.4. Las lineas horizontales representan la actualización de los nodos de comprobación y las lineas verticales representan la actualización de los nodos bit.

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

Pseudocódigo 3.1 Algoritmo BF con actualización por *inundación*.

1: – *inicialización* –
2: para $n \in \{1,\ldots,N\}$ y $m \in M_n$
3: $z_n = \begin{cases} 1, & si \quad y_n \leq 0 \\ 0 & si \quad y_n > 0 \end{cases}$
4: – *iteraciones* –
5: para $i \in \{1,\ldots,I_{max}\}$
6: – *actualización de los nodos de comprobación* –
7: para $m \in \{1,\ldots,M\}$ y $n \in N_m$
8: $s_m = \sum_{n=1}^{N} z_n \cdot h_{mn}$
9: – *actualización de los nodos de bit* –
10: para $n \in \{1,\ldots,N\}$ y $m \in N_m$
11: $E_n = \sum 2 \cdot (s_m - 1) \cdot |y_n|_{min-m}$
12: – *inverertir* z_n –
13: para $n \in \{1,\ldots,N\}$
14: $z_n =$ not z_n si $E_n \geq \delta^i$

3.8.2. 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 j de la matriz de paridad $\boldsymbol{H}$. Además, este valor debe ser tal que el resultado de la división $M_G = M/G$ sea cero, siendo M_G el número de nodos de comprabación de cada grupo g. Cada iteración se compone de G subiteraciones, que se realizan en dos fases, primero se actualiza los g nodos de comprobación correspondientes y después se actulizan todos los nodos de bit. El intercambio de mensajes parciales entre los grupos de nodos de comprobación y los nodos de bit provoca que la convergencia del algoritmo de decodificación se acelere. Esto se traduce en que un algoritmo con actualización layered alcanza con menos iteraciones las mismas prestaciones que con la actualización por inundación.

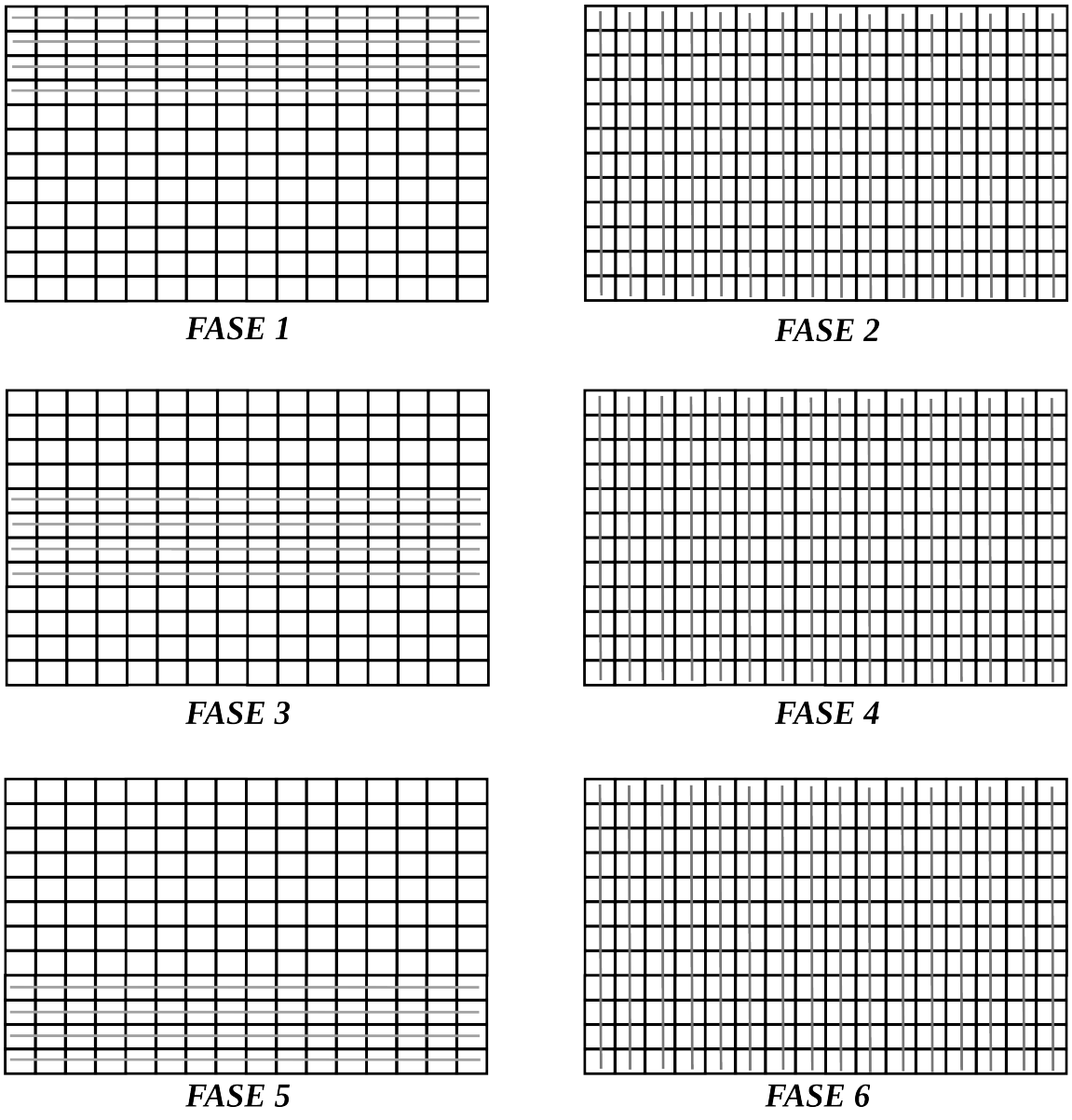


Figura 3.5: Método de actualización *layered* para la matriz LDPC regular (3,4) de orden 12x16.

En la figura 3.5 se muestra la representación gráfica del método de actualización *layered* para un número de grupos $G = 3$ y una matriz H de tamaño $12x16$.

En el pseudocódigo 3.2 se observa el funcionamiento del algoritmo *layered*. Igual como en el pseudocódigo de la actualización por inundación tampoco contempla la finalización anticipada. En este caso el algoritmo utilizado para la actualización *layered* es el algoritmo MS.

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: $\quad\quad \mu_{m,n}^{(i,g)} = \Gamma_{m,n}^{(i)} \cdot \min_{n' \in N_{m\setminus n}} (|\lambda_{n',m}^{(i-1)}|)$
11: $\quad\quad$ – *actualización de los nodos de bit* –
12: $\quad\quad$ para $n \in \{1,\dots,N\}$ y $m \in M_n$
13: $\quad\quad \lambda_{m,n}^{(l)} = l_n + \sum_{\substack{m' \in M_n \\ m' > g\cdot M_G}} \mu_{m',n}^{(i-1)} + \sum_{\substack{m' \in M_n \\ m' \leq g\cdot M_G}} \mu_{m',n}^{(i)}$
14: $\quad\quad \lambda_{n,m}^{(i)} = \lambda_n^{(i)} - \begin{cases} \mu_{m,n}^{(i-1)} & m > g\cdot M_G \\ \mu_{m,n}^{(i)} & m \leq g\cdot M_G \end{cases}$
15: – *decodificación hard* –
16: para $n \in \{1,\dots,N\}$
17: $\quad z_n = \begin{cases} 1 & \lambda_n^{(I_{max})} \geq 0 \\ 0 & \lambda_n^{(I_{max})} < 0 \end{cases}$

3.9. Bit Flipping con pesos con actualización layered modificada

En este apartado se le adapata la actualización *layered* de los algoritmos basados en SP al algoritmo IWBF. Al portar el modelo de actualización *layered* a este algoritmo, se ha modificado un poco el funcionamiento del mismo para obener una mejor decodificación. La primera modificación es la eliminación de la resta de contribución de uno mismo de E_n del estado anterior, ya que era un valor casi despreciable. Y la otra es quitar peso en el cálculo de E_n de la energía calculado en el estado anterior, así que en el algoritmo se introducira un factor C_{E_n} que se llamará factor de compensación de energía.

A continuación se observa el pseudocódigo 3.3 resultante para el algoritmo IWBF con la actualización *layered* modificaca. Como en los otros pseudocódigos se ha eliminado la finalización anticipada por simplicidad.

Pseudocódigo 3.3 Algoritmo IWBF con actualización *layered*.

– *inicialización* –
para $n \in \{1,\dots,N\}$ y $m \in M_n$
  $z_n = \begin{cases} 1, & si \quad y_n \leq 0 \\ 0 & si \quad y_n > 0 \end{cases}$
  $W_{nm} = \min_{n:n\in N_{(n)\setminus n}} |y_n|, \quad m[1,M], \quad n \in N(m)$
– *iteraciones* –
para $i \in \{1,\dots,I_{max}\}$
  – *subiteraciones* –
  para $g \in \{1,\dots,G\}$
    – *actualización de los nodos de comprobación* –
    para $m \in \{1,\dots,M\}$ y $n \in N_m$
      $s_m = \sum_{n=1}^{N} z_n \cdot h_{mn}$
    – *actualización de los nodos de bit (grupo)* –
    para $n \in \{(g-1)\cdot N_G + 1,\dots,g\cdot N_G\}$ y $m \in M_n$
      $E_{n_{it}} = \sum 2\cdot(s_m - 1)\cdot W_{nm} + \beta_2$
      $E_n = E_{n_{it}} + C_{E_n}\cdot E_n$
– *inverertir* z_n –
para $n \in \{1,\dots,N\}$
  $z_n =$ not z_n si $E_n \geq \delta^i$

Capítulo 4

SIMULACIÓN Y MODELADO DEL SISTEMA

El entorno de simulación y los algoritmos de decodificación se han programado en Matlab. En éste se puede definir tanto el decodificador a utilizar como los parámetros mecesarios para realizar una simulación. Entre estos parámetros se puede destacar, el número de canales, el rango de la realción señal a ruido en dB, el número máximo de iteraciones, la matriz a utilizar, el número máximo de paquetes erróneos por canal, etc.

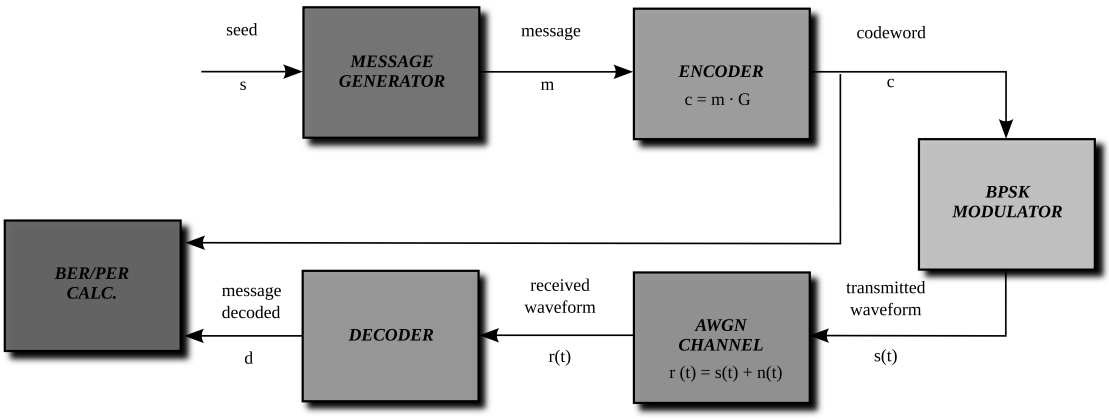


Figura 4.1: Bloques del simulador software.

Una vez citados los principales parámetros del entorno de simulación, se proseguira con el funcionamiento del simulador software (4.1). Primero se genera un mensaje aleatorio, a continuación en el siguiente bloque, se genera la palabra código utilizando la palabra mensaje y la matriz generadora G. Luego se le aplica una modulación $BPSK$ para posteriormente ser transmitido por un canal $AWGN$. Y por último la onda recibida del canal llega al bloque decodificador para la obtención del mensaje decodificado original. Este proceso es repetido hasta que se simulen todos los canales y se llegue al máximo de paquetes erróneos trasmitidos que no se ha podido decodificar.

Además el entorno de simuácion también proporciona la tasa de error de bit (BER) y la tasa de error por paquete (PER) para cada uno de los canales simulados, alacenando los resultados para poder analizarlos posteriomente. Estos dos resultados son utilizados para el análisis de las prestaciones de los distintos algoritmos.

4.1. Método seguido para la obtención del valor de compensación de E_n

Se parte del algoritmo IWBF, el cual sólo cambia el bit Z_n que tiene mayor energía, así no influirá la eleción de los umbrales para este cálculo. Así que partiendo de este algoritmo se simulará el algoritmo IWBF sin ningúna modificación para posteriormente compararlo con el algoritmo IWBF con actualización layered con distintos valores de la compensación de la E_n (C_{E_n}). Como se puede observar en la gráfica 4.2, se ha simulado con unos valores de 0.25 a 0.5, el valor final elegido para el algoritmo es el de 0.25, aunque no sea el que mejor prestaciones da, es el que a la hora de implementar en hardware es el mas sencillo de implementar. Y también que la diferencia entre el de mejor prestaciones (0.3) y este es mínima, así que por coste hardware se elegige este, ya que sólo implicará un desplazamiento al ser el equivalente a una división por 4 en potencia de 2.

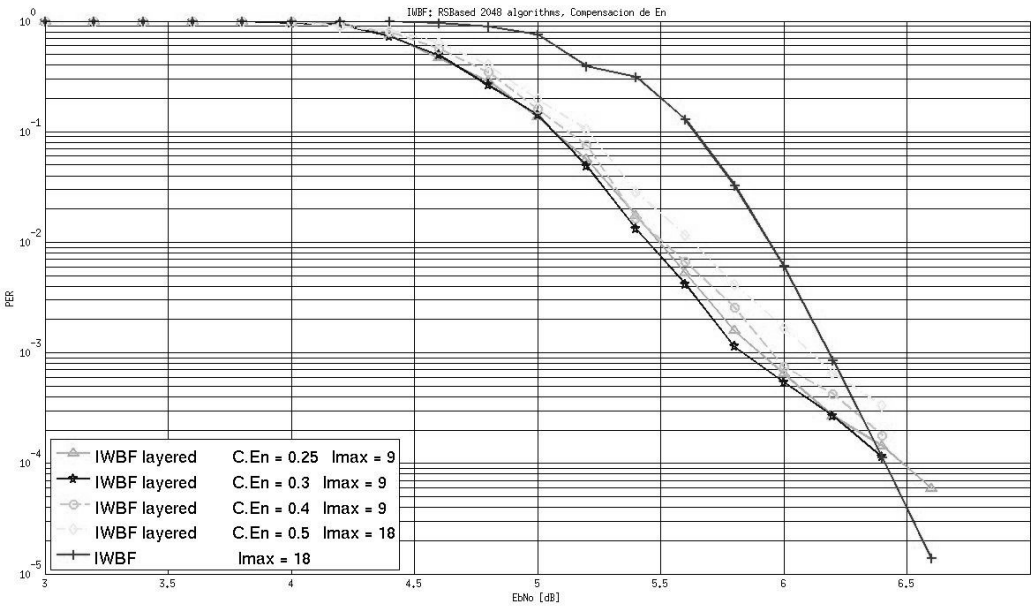


Figura 4.2: Tasa de error por paquete de el algoritmo IWBF para $C_{E_n} = 0.25, 0.3, 04, 05$ con 10 iteraciones.

4.2. Método seguido para la obtención de los umbrales

Para el ajuste de los umbrales δ por iteración con el simulador en Matlab utilizando el algoritmo *TIWBF* se simula con un *EbN*0 medio y se captura en cada simulación 10000 paquetes erróneos. Primero se empezara con un umbral fijo para todas las iteraciones y se ajusta el valor de la primera iteración. Se va ajustando el valor de los umbrales de la primera iteración a la última ya que el umbral elegido en las anteriores iteraciones influyen en las siguientes.

Para la evaluación de los datos realizados en cada simulación se hará la media de los valores de E_n correctos ($meanMm_{ok}$) y la media de los valores de E_n de los bits erróneos ($meanMm_{er}$). Con esto y la desviación típica σ ($meanMs$) se podrá evaluar y ajustar los valores de δ para cada iteración. En la gráfica 4.3 se puede ver los datos obtenidos de una simulación, en el eje y se representa los valores de E_n, mientras que en el eje x se representa los valores de la iteración. También se observa en esta como la media de E_n correctos ($meanMm_{ok}$) y la media de E_n erróneos ($meanMm_{er}$) se mantienen separadas bastante al final, así se asegura cambiar los bits erróneos. Para la elección de los umbrales se partirá de la referencia dada por dos veces la desviación típica de los bits erróneos ($meanMs_{er}$) siendo los umbrales elegidos mayor a esta. Así se asegura no cambiar un bit correcto por un erróneo.

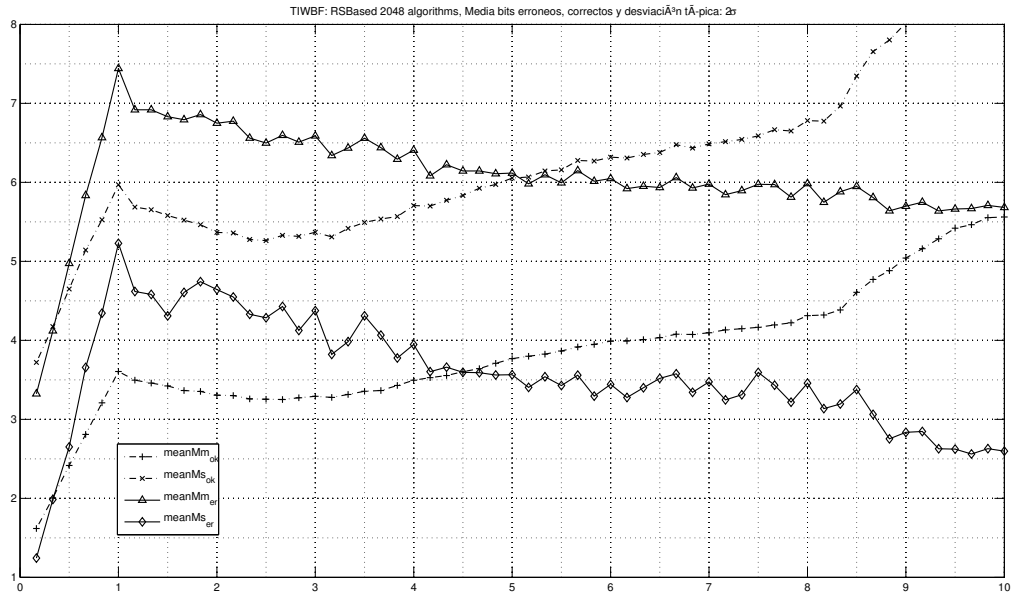


Figura 4.3: Representación de la media de E_n de los bits correctos, desviación típica bits correctos (σ_{ok}), media E_n de los bits erróneos y desviación típica de los erróneos (σ_{er}).

Para el cálculo de la media ($\overline{x}$), la desviación típica (σ) y la varianza (σ^2) se aplican las siguientes fórmulas:

$$\overline{x} = \mu = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \overline{x})^2}$$

$$\sigma^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \overline{x})^2$$

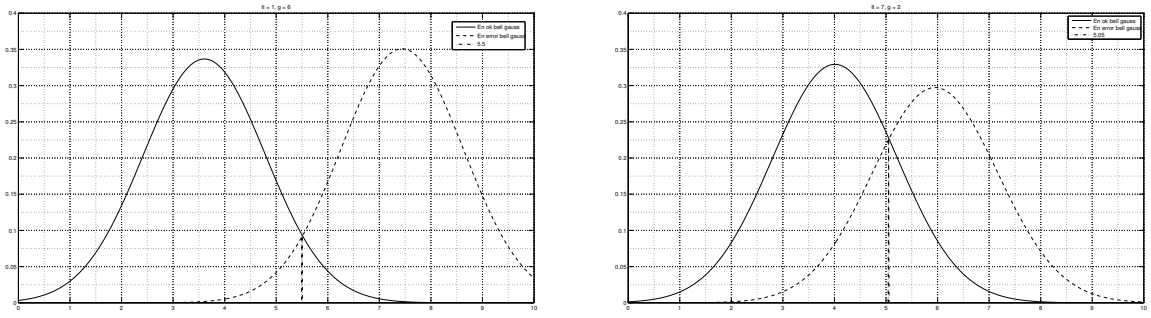
Y para la representación de *meanMs*:

$$meanMs_{ok} = \overline{x} + 2 \cdot \sigma$$

$$meanMs_{er} = \overline{x} - 2 \cdot \sigma$$

Otra manera de representación, tal y como se observa en la gráfica 4.4, sería la representación con distribuciones normales. En esta, se representa la campana de Gauss de los valores de E_n correctos y los valores de E_n erróneos, además del punto de intersección de estas dos. En este caso cada figura representa los datos de la iteración y subiteración concreta. En la figura (a) se representa la iteración 1 y la subiteración 6, y en la figura (b) la iteración 7 y subiteración 2. Para generar la figura 4.4 se ha aplicado la ecuación 4.1.

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}} \tag{4.1}$$



(a) (b)

Figura 4.4: Distribución normal de E_n de los bits correctos y erróneos. (a) Para iteración 1 y subiteración 6. (b) Para iteración 7 y subiteración 2.

Finalmente, en la tabla 4.1 tras muchas simulaciones estos son los umbrales δ escogidos como óptimos para el código LDPC estructurado 10GBase-T para cada iteración del algoritmo TIMWBF.

It	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
δ	8	8	8	7.5	7.5	7.5	7.25	7.25	6.75	6.75

Tabla 4.1: Valores δ para cada iteración *It* del algoritmo TIMWBF

4.3. Prestaciones del algoritmo

En este apartado se muestra las curvas de tasa de error por paquetes de el algoritmo basados en IWBF con la arquitectura por inundación y la layered. Como se observa en la figura 4.5, el algoritmo con actualización *layered* consigue mantener las prestaciones del algoritmo con actualización por inundación sólo con 10 iteraciones respecto a las 18 iteraciones de la actualización por inundación.

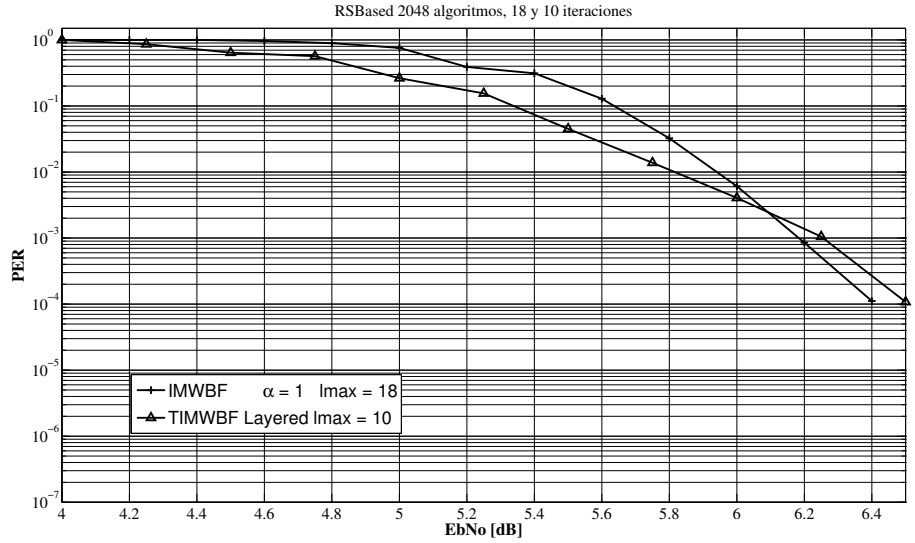


Figura 4.5: Prestaciones de los algoritmos basados en IWBF. Resultados de la tasa de error por paquete (PER).

Capítulo 5

DISEÑO E IMPLEMENTACIÓN

En este apartado se explica la arquitectura *layered* propuesta para un decodificador basado en IWBF, tanto a nivel de funcionamiento como a nivel hardware. En cuanto a la implementación, el decodificador se ha implementado en *VHDL*, y al ser un código parametrizable se puede utilizar para cualquier matriz regular, aunque se tendría que recalcular los umbrales y distintos parámetros del decodificador para la matriz en concreto. Este decodificacor ha sido probado con una matriz de test pequeña y con la matriz RsBased (2048,1723) del estándar IEEE 802.3an (10GBase-T). Finalmente se hace una comparativa de las dos arquitecturas en cuanto a prestaciones, área y velocidad.

5.1. Arquitectura layered

Esta arquitectura implementa el método de actualización por capas horizontales - *layered*. En éste, la matriz de paridad *H* se divide en bloques de filas, denominado capas, que se procesan de forma sequencial. En la arquitectura *layered* propuesta para el algoritmo IWBF agrupa el cálculo de una CNU con sus correspondientes VNUs en una unidad que se llamará *total node unit* (TNU), y por ésto se llamará a esta arquitectura TIWBF.

Cada capa o bloque está formado por 64 TNUs, y se tiene 6 capas o bloques. Por lo tanto, se obtendrá una iteración al cabo de 6 subiteraciones. Para la actualización de las 64 TNUs se actualizan todas de golpe en un ciclo de reloj. Mientras que en la siguiente subiteración se actualizará la siguiente capa, pasando a entrar los datos calculados en la capa anterior en el orden adequado a través de unos multiplexores a las entradas de la TNU. Físicamente se implementan 64 TNUs y a través de los multiplexores situados a la entrada de estas se reordenan y entran los datos en el orden adequado para una de las capas.

En la figura 5.1 se muestra la arquitectura resultante de una TNU. Se puede dividir en tres secciones, la primera sección son los multiplexores de entrada, que como ya se ha dicho son para introducir en el orden adequado las entradas. La segunda la CNU que realiza las operaciones descritas en (3.12) y (3.15). Mientras que la última sección está formada por el resto de componentes que será el equivalente de 64 VNUs y son las operaciones en el pseudocódigo 3.3 línea 14,15 y 18. Estas dos últimas partes se detallarán en los dos apartados siguientes.

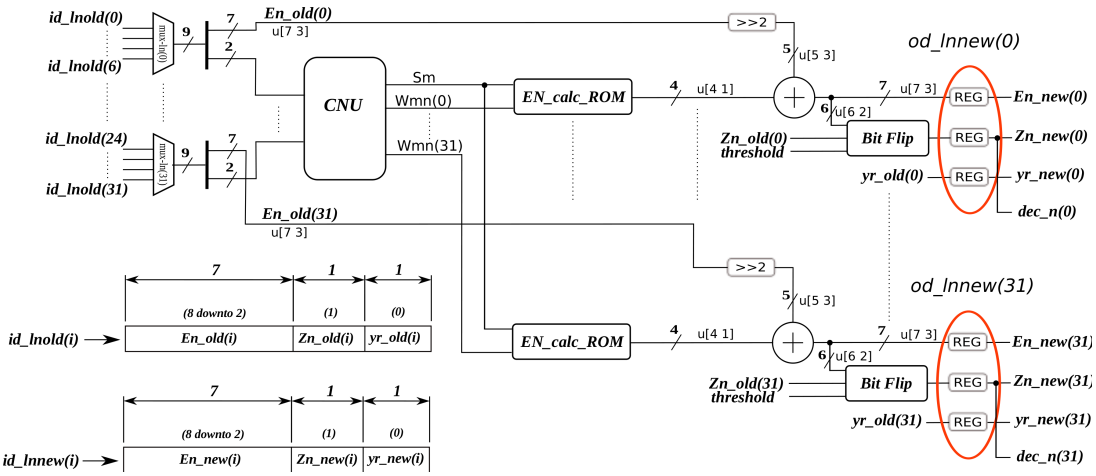


Figura 5.1: Arquitectura hardware de la TNU.

5.1.1. Unidad de actualización de nodo (CNU)

La unidad *CNU* como ya se ha dicho realiza las operaciones descritas en (3.12) y (3.15). La figura 5.2 muestra la lógica digital utilizada para implementar la CNU. Sus entradas son Z_n e y_r, donde $Z_n(i)$ es la decisión del bit i e $y_r(i)$ es su confianza asociada al bit i. Cada unidad CNU actualiza k mensajes entrantes, para este caso 32, ya que el peso de columna de la matriz H es $k = 32$. Esta unidad calcula el síndrome simplemente con una puerta lógica XOR de 32 entradas tal y como se puede ver en la figura 5.2.

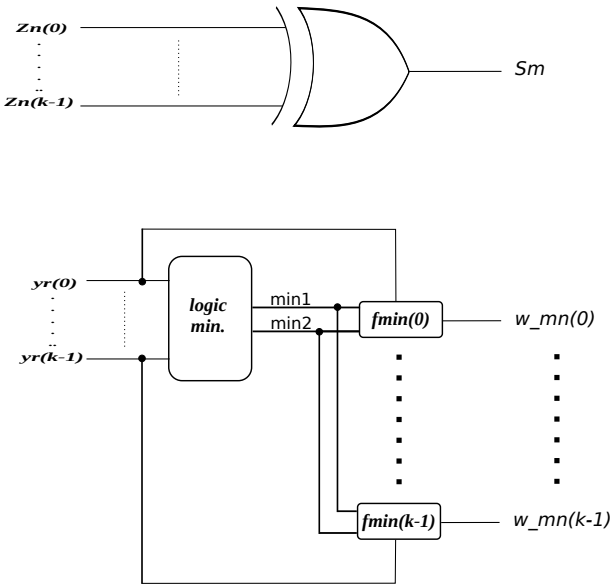


Figura 5.2: Arquitectura hardware de la CNU.

Para el cálculo de los pesos (3.15) de una manera óptima, se calculan dos valores mínimos (min_1, min_2) usando el bloque *min logic*. Este bloque realiza la operación descrita en (5.1) para el cálculo del min_1 y (5.2) para el cálculo del min_2. Los valores resultantos del cálculo de los dos mínimos entran a los bloques f_{min}, los cuales realizan la exclusión de la información del bit en cuestión (3.13) mediante la operación $f_{min} = \overline{yr(i)} \cdot min_2 + min_1$ calculando k mínimos distintos para cada *CNU*.

Cálculo de los mínimos para k entradas:

$$min_1 = yr(0) \cdot yr(1) \cdot .... \cdot yr(k-1) \tag{5.1}$$

$$min_2 = min_1 + (\overline{yr(0)} \cdot yr(1) \cdot ... \cdot yr(k-1)) + ... + (yr(0) \cdot yr(1) \cdot ... \cdot \overline{yr(k-1)}) \tag{5.2}$$

5.1.2. Unidad de actualización de nodo variable (VNU)

La unidad VNU implementa las ecuaciones del pseudocódigo 3.3 de las líneas 13,14 y 17. La VNU está formado por la memória ROM para el calculo de $E_{n_{it}}$, la suma con la división de la E_n del estado anterior y el bloque llamado *Bit Flip* respectivamente.

Para el diseño de la memoria se utilizan 4 LUTS para cada ROM. La ecuación que se calcula mediante estas LUTs es la de el pseudocódigo 3.3 línea 13. En hardware para trabajar con todas las operaciones sin signo se le añade un offset ($\beta_2 = 2.5$) a esta ecuación, para que todos los resultados sean positivos. En la 5.1 tenemos los valores con que se ha rellenado esta ROM. Dicho esto, el formato necesário para la codificación de estos valores es de 4 bits fraccionarios y un bit decimal.

$W_{mn}(i)$	S_m	$E_nit(i)$ u[4 1]
0	0	”0011” (1.5)
0	1	”0111” (3.5)
1	0	”0000” (0)
1	1	”1010” (5)

Tabla 5.1: Valores E_n almacenados en la ROM.

La línea 15 ($E_n = E_{n_{it}} + C_{E_n} \cdot E_n$) del pseudocódigo 3.3 se implementa en hardware, tal y como se puede ver en la figura 5.1. Mediante el sumador de 7 bits realiza la suma de $E_{n_{it}}$ y $C_{E_n} \cdot E_n$. Donde $C_{E_n} \cdot E_n$ ($C_{E_n} = 0.25$)es el desplazamiento de dos bits, quedando la salida En_{old} en formato [5 3]. Identificando las partes de la ecuación con el hardware: $E_{n_{it}}$ es la salida de la memoria ROM, por la otra parte el desplazamiento de la señal es el equivalente $C_{E_n} \cdot E_n$. Finalmente se suman las dos partes con el sumador de 7 bits obteniendo a la salida E_n.

El último bloque que conforma la VNU es el *bit flip* (figura 5.1), que implementa la inversión del bit z_n mediante la siguiente ecuación: $z_n =$ not z_n si $E_n \geq \delta^i$. Este bloque se implementa con un comparador, que compara la E_n calculada con un umbral para cada iteración almacenado en una memoria ROM. En el caso de que supere o sea igual a dicho umbral, al bit z_n correspondiente se le aplica la función logica not, que inverte el valor de dicho bit.

5.2. Resultados de implementación

La arquitectura *layered* propuesta en este proyecto se ha codificado mediante el lenguaje VHDL y se ha implementado en varios dispositivos FPGA. Los dispositivos utilizados para la implementación son la FPGA Stratix III EP3S2340H1152C2 y la Virtex-6 XC6VLX7C0-2FF1760 de los fabricantes Altera y Xilinx, respectivamente.

La velocidad de decodificación de datos conseguida por la arquitectura *layered* con el algoritmo TIWBF propuesto es :

$$T = \frac{N \cdot f_{CLK}}{It \cdot N_{GROUP}} \tag{5.3}$$

donde N es el número de bits de la palabra código, f_{clk} es la frecuencia de reloj, It es el número de iteraciones y N_{GROUP} el número de grupos que hay, y por tanto el número de ciclos por cada iteración.

En la tabla 5.2 se muestra los resultados de la implementación obtenidos con las FPGAs de Xilinx y Altera. En los dispositivos de Xilinx el sintetizador no ha sido capaz de implementar el diseño del algoritmo layered. Mientras con los dispositivos de Altera el decodificador layered trabaja a una frecuencia de reloj de 60 *MHz* alcanzando una velocidad de decodificación de 2.048*Gbps* con 10 iteraciones.

Algoritmo	**Dispositivo**	**Arquitectura**		**N^o Uni.**	**LUTs**	**REG**	f_{max}	**Velocidad dec.***
TIMWBF (RsBased)	Xilinx XC6VLX7C0-2FF1760	*layered*	*TNU*	64	337	289	-	-
			COMPL.	-	-	-		
	Altera EP3S23401411S2C2		*TNU*	64	304	289	60 MHz	2,048 Gbps
			COMPL.	-	54581	19833		

Tabla 5.2: Resultados de implementación en una FPGA de la arquitectura *layered* para el código RsBased (2048,1723) del estándar IEEE 802.3an (10GBase-T).

5.3. Conclusiones

Este trabajo se ha centrado en el desarrollo de la arquitecturas layered para decodificar códigos LDPC regulares a velocidades de Gbps. Se ha utilizado el código LDPC definido en el estándar 10GBase-T por ser un estándar que requiere velocidades de transmisión de hasta 10 Gbps. Se ha utilizado el algoritmo de decodificación IWBF y optimizando sus parámetros para el caso del código LDPC de 10GBase-T. La arquitectura se ha codificado en VHDL y se ha dotado de cierto grado de parametrización, como puede ser número de CNUs, VNUs, número de unos por filas y por columnas, es decir, las características de la matriz utilizada. Se han implementado en dispositivos FPGA de Xilinx y Altera. Se ha alcanzado una velocidad máxima de decodificación de 2.048 Gbps con 10 iteraciones en el dispositivo Stratix III con la arquitectura layered. Cabe destacar que esta arquitectura no puede ser rutada en ningún dispositivo de Xilinx. También cabe decir, que aunque la velocidad alcanzada por este decodificador queda por debajo de la esperada para su uso en dicho estándar, esto es debido a la tecnología de implementación utilizada, los dispositivos FPGAs. El mismo decodificador implementado con un processo CMOS actual alcanzaría velocidades muy por encima de las requeridas por el estándar 10GBase-T

BIBLIOGRAFÍA

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. on Information Theory*, vol. IT-18, pp. 21–28, Jan. 1962.

[2] "IEEE 10 GBase-T Task Force." [Online]. Available: http://www.ieee802.org/3/an/index.html

[3] F. Angarita, J. Marín-Roig, V. Almenar, and J. Valls, "Low-complexity low-density parity check decoding algorithm for high-speed very large scale integration implementation." 2012, pp. 2575–2581.

[4] J. S. Y. J. Zhang and M. P. C. Fossorier, "Low-latency decoding of EG LDPC codes," vol. 25, Sept. 2007, pp. 2879–2885.

[5] S. Y. Kou and M. Fossorier, "Low density parity check codes based on finite geometries: A rediscovery and new results. a rediscovery and new results," vol. 47, Nov. 2001, pp. 2711–2736.

[6] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," in *Design, Automation Test in Europe Conference*, vol. 8, Mar. 2004, pp. 165–167.

[7] Z. M. Jiang, C. Zhao and Y. Chen, "An improvement on the modfied weighted bit flipping decoding algorithm for ldpc codes," *IEEE Communications Letters*, vol. 9, pp. 814–816, Sept. 2005.

[8] 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.

[9] 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.

[10] 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.