

**INSTITUTO TECNOLÓGICO Y DE ESTUDIO SUPERIORES DE  
MONTERREY  
CAMPUS CIUDAD DE MÉXICO**

**División de Ingeniería y Arquitectura**

**Departamento de Ingeniería Eléctrica y Electrónica**



**TECNOLÓGICO  
DE MONTERREY®**

**Proyectos de Ingeniería II**

---

**Turbo decodificador de códigos producto**

---

**Alumnos**

Ernesto Jordán Cañas

Tonantzin Monroy Soberón

**Asesor**

Dr. Francisco Javier Cuevas Ordaz

**Sinodales**

Dr. Raúl Crespo Saucedo

Dr. Martín Rogelio Bustamante Bello

**Profesor**

MBA. José Vicente Quintanilla Bonifaz



<b>I. Introducción</b>	3
<b>I.1. Objetivos y Metas</b>	4
<b>I.2. Estado del Arte</b>	5
<b>II. Teoría(s)</b>	
<b>Capítulo 1: El sistema de comunicación digital</b>	
1.1 Introducción	9
1.2 Sistema de comunicación digital	9
1.3 Eficiencia de un sistema de transmisión	14
1.4 Límite de Shannon	14
1.5 Conclusión	14
<b>Capítulo 2: Códigos en bloque, códigos producto y decodificación iterativa</b>	
2.1 Introducción	15
2.2 Códigos en bloques	16
2.3 Decodificación binaria de códigos BCH	19
2.4 Decodificación ponderada de códigos BCH	20
2.5 Códigos producto	21
2.6 El desempeño del decodificador	28
2.7 Conclusiones	28
<b>Capítulo 3: Códigos en bloque, códigos producto y decodificación iterativa</b>	
3.1 Introducción	31
3.2 El código BCH extendido (32,26,4)	31
3.3 Decodificación del código producto BCH (32,26,4) $\otimes$ (32,26,4)	32
3.4 Conclusiones	48

**Capítulo 4: Validación con C++**

4.1 Introducción	51
4.2 Descripción del programa	51
4.3 Análisis de resultados	53
4.4 Conclusiones	55

**III. Resultados**

<b>III.1. Diseño</b>	59
<b>III.2. Programación</b>	59
<b>III.3. Contribuciones al estado del arte</b>	59

**IV. Conclusiones** 63

<b>IV.1. Perspectivas y trabajo a futuro</b>	65
--	----

**V. Actividades realizadas** 69

**VI. Referencias** 75

**VII. Anexos** 77

---

# I. Introducción

---

En 1993, C. Berrou introdujo el concepto de turbo códigos como una clase de códigos correctores de errores elaborados a partir de la concatenación paralela de dos códigos convolutivos codificados mediante un proceso iterativo, este principio es conocido como turbo código convolutivo. El desempeño de estos códigos en condiciones ideales de codificación se aproxima al límite teórico de Shannon, el cual es el principio en que se basa toda la teoría moderna de códigos correctores de errores. Estos resultados provocaron el interés en el mundo entero debido a sus diversas aplicaciones en los sistemas de comunicación digital.

R. Pyndiah, propuso en 1994 el concepto de turbo códigos en bloques como una alternativa a los turbo códigos convolutivos. El esquema de codificación se basa en la concatenación en serie de códigos en bloques. El algoritmo de decodificación está basado en un proceso iterativo utilizando decodificadores elementales a entradas y salidas ponderadas. El desempeño de este algoritmo de decodificación es comparable al de los turbo código convolutivos.

El objetivo de este proyecto es el estudio de los turbo códigos en bloques y la decodificación del código producto utilizando un código BCH extendido. Se presentará el diseño digital de un decodificador elemental en VHDL (Very high speed integrated circuit Hardware Descriptive Language), dividido en tres bloques principales: Recepción, tratamiento y emisión. Este decodificador elemental recibirá una matriz de ponderación correspondiente a los 32 bits de un renglón o columna del código producto, dentro de los cuales se incluyen 4 bits de redundancia y uno de paridad, esta matriz se procesará con el fin de corregir un bit de esta secuencia de bits recibidos.

En los capítulos siguientes se presentan algunas características de la teoría de la información y se analiza la estructura de una cadena de comunicación digital así como las bases de su funcionamiento para ubicar el área de desarrollo del proyecto. Posteriormente se examina la construcción de un código producto, se habla del vector llamado síndrome enfatizando su papel en la ubicación del error en la matriz y se finaliza con los algoritmos matemáticos que justifican esta decodificación.

También, se estudiará la arquitectura de la decodificación de un código producto construido a partir de dos códigos BCH extendidos (32,26,4) para la corrección de un error y se estudiara el decodificador elemental o unidad de tratamiento bloque por bloque.

## I.1. Objetivos y Metas

- Objetivo principal:
  - Diseñar un decodificador que permita la transmisión con corrección de errores a alta velocidad, y así contribuir al inicio de la investigación de turbo códigos en bloques en México.

Metas:

- Primer semestre:
  - Entender la matemática y los algoritmos utilizados en el proceso de decodificación de turbo códigos.
  - Diseñar un decodificador de turbo códigos en bloques.
  - Familiarizarnos con el programa VHDL con el cual programaremos y simularemos.
  - Programar cada bloque del decodificador en VHDL.
  - Documentar cada avance para ir recopilando el reporte final.
  
- Segundo semestre:
  - Investigar sobre turbo códigos en bloques.
  - Programar el decodificador elemental en VHDL para un determinado código.
  - Validar el programa en VHDL con un sistema de transmisión completo programado en C++.
  - Obtener gráficas de desempeño.
  - Investigar comparativamente los turbo códigos convolutivos.

## **I.2. Estado del Arte**

Es en 1993 cuando C. Berrou introdujo el concepto de turbo códigos a partir de la concatenación paralela de 2 códigos convolutivos, este principio es ahora conocido con el nombre de turbo códigos convolutivos. Siendo una de la maravillas de estos, la aproximación al límite teórico de Shannon en condiciones ideales de codificación.

En 1994 R. Pyndiah propuso el concepto de turbo códigos en bloques, los cuales se basan en la concatenación serial de códigos en bloques cuyo algoritmo es un proceso iterativo a partir de decodificadores elementales con entradas y salidas ponderadas. Teniendo estos, un rendimiento similar a los primeros turbo códigos existentes.

A partir de estas investigaciones ha habido varias publicaciones sobre este tema, generando una revolución en la decodificación digital.

Así se hace evidente que el proyecto que realizaremos tiene la característica de ser una tecnología de punta, pues es algo nuevo que únicamente países como Francia y Canadá han comenzado a desarrollar.

---

## **II. Teoría**

---



# Capítulo 1

---

## El sistema de comunicación digital

---

### 1.1 Introducción

Para entender la naturaleza del proyecto, en este capítulo se describe el funcionamiento básico de un sistema de comunicación digital así como los elementos que lo componen. Esta explicación facilita el entendimiento general del proceso de transmisión y recepción de datos para posteriormente profundizar en las etapas donde se sitúa el proyecto.

### 1.2 Sistema de comunicación digital

Los sistemas de comunicación están diseñados para enviar mensajes o información de una fuente emisora hacia uno o más destinatarios. El propósito de un sistema digital de comunicación es transmitir información de manera íntegra a pesar de las perturbaciones. En general, un sistema de comunicación puede ser representado por un diagrama a bloques como el mostrado en la figura 1.1, compuesto por fuente de información, codificador, modulador, canal, demodulador, decodificador y receptor de información.

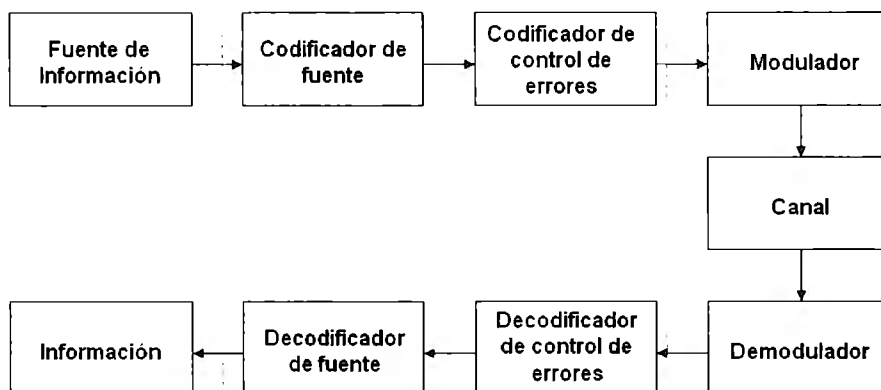


Figura 1.1 Cadena de transmisión digital

### 1.2.1 Fuente de información

En los sistemas de transmisión digital, la fuente puede ser tanto analógica como digital. Las señales digitales, emitidas por una computadora, se pueden transmitir directamente, sin embargo existen otro tipo de señales como la voz y las imágenes, donde la señal analógica requiere de un transductor. En el caso de información analógica, se debe discretizar (muestrear y cuantizar) para poder transmitir la señal. Cualquier fuente de información está descrita en términos probabilísticos.

### 1.2.2 Codificador de fuente

En un sistema digital los mensajes producidos por la fuente son convertidos en secuencia de dígitos binarios. Idealmente, se desea representar la salida de la fuente con la menor cantidad de dígitos binarios posible, esto es, tener una representación eficiente de la información proveniente de la fuente con poca o nula redundancia. Una consideración importante al codificar la información proveniente de la fuente es que hay datos que se pierden al discretizar, esto reduce la precisión de los datos.

El proceso de discretización es realizado por el codificador de fuente, también llamado compresor de datos. Debido a la manipulación de bits en el codificador, el intervalo entre emisión de datos aumenta, lo que provoca que la tasa de transmisión sea menor en el codificador que en la fuente.

### 1.2.3 Codificador de control de errores

El propósito del codificador de canal, o codificador de control de errores, es introducir redundancia en la secuencia de información binaria proveniente del codificador de fuente que pueda ser utilizada posteriormente por el receptor para traspasar los efectos de ruido e interferencia encontrados en la transmisión de la señal a través del canal. La redundancia sirve para incrementar la confiabilidad de la información transmitida. Existen diferentes maneras de codificar una secuencia binaria entre las que se encuentran la repetición y el mapeo. La cantidad de redundancia introducida por el codificador se mide como tasa de codificación.

El esquema básico de detección de errores puede detectarlos, pero no corregirlos. A medida que se aumentan bits de redundancia, la detección de errores mejora. Desafortunadamente, el aumento de bits de redundancia resulta en la disminución de tasa de transmisión y por lo tanto se puede convertir en una técnica poco eficaz. Por ello se han desarrollado códigos más sofisticados

que permiten la transmisión de información con una pequeña tasa de errores, por ejemplo los códigos de Bloques, códigos Hamming, códigos Reed-Solomon y códigos Convolutivos. El proyecto se desarrolla con códigos en Bloques, donde la tasa de rendimiento está dada por el número de bits de información divididos por la longitud de la palabra, conformada por los bits de información y los bits de redundancia.

#### 1.2.4 Modulador

La secuencia binaria a la salida del codificador de control de errores llega al modulador, el cual sirve como interfaz hacia el canal de comunicación. El modulador modifica las características de la señal de onda, como fase, amplitud y frecuencia para representar las secuencias de información binaria. Es posible modular cada 1 y 0 como una forma de onda diferente o bien, modular una cierta cantidad de bits de información  $b$  al mismo tiempo utilizando  $M=2^b$  formas de onda distintas para cada una de las posibles secuencias de  $b$  bits. Cuando la tasa de transmisión del canal es fija, el intervalo de tiempo disponible para transmitir una de las  $M$  formas de onda es  $b$  veces el periodo de bit en un sistema que utiliza modulación binaria.

#### 1.2.5 Canal

El canal de comunicación es el medio físico usado para enviar la señal desde el transmisor hasta el receptor. Este puede ser la atmósfera (espacio libre), cables o fibras ópticas, por mencionar algunos.

En general un canal puede ser considerado cualquier medio físico para la transmisión de una señal, la característica esencial es que la señal transmitida es alterada de manera aleatoria por una serie de eventos. La forma más común de degradación de una señal proviene del ruido aditivo generado a la llegada del receptor donde se tienen ampliificaciones de señal. Este tipo de ruido es comúnmente llamado ruido térmico.

Los sistemas inalámbricos tienen además degradación a causa del ruido generado por el humano así como ruido atmosférico tomado por la antena receptora. La interferencia con otros usuarios del canal es otra forma de ruido aditivo que se presenta tanto en sistemas cableados como inalámbricos. En sistemas de radio que utilizan el canal ionosférico para largas distancias y transmisiones de radio de onda corta, se tiene además una degradación de la señal debido a la propagación mutlitrayectoria. Dicha distorsión se considera no aditiva y se manifiesta como variaciones de tiempo en la amplitud de la señal, es decir, un defazamiento a la llegada de las ondas como interferencia de edificios y objetos sólidos. Tanto las distorsiones aditivas como las

no aditivas se caracterizan por ser fenómenos aleatorios descritos en términos estadísticos. El efecto de estas distorsiones en la señal debe ser tomado en cuenta en el diseño de sistemas de comunicación.

El diseñador de un sistema de comunicación trabaja con modelos matemáticos que caracterizan estadísticamente la distorsión de la señal encontrada en los canales físicos. Normalmente, la descripción estadística usada en un modelo matemático es el resultado de mediciones empíricas obtenidas de experimentos donde se involucra la transmisión de señales sobre dichos canales. En estos casos, existe una justificación física para el modelo matemático usado en el diseño. También, en el diseño de algunos sistemas de comunicación, las características estadísticas del canal pueden variar significativamente con el tiempo lo que obliga a diseñar sistemas robustos ante tal variedad de distorsiones.

#### 1.2.5.1 Canal a ruido aditivo Gaussiano (RGBA)

El canal Gaussiano se define por una entrada binaria  $X$  y una salida  $Y$  que hace referencia a la entrada alterada por el ruido  $Z$ , el cual determina el desempeño del canal de transmisión como se muestra en la figura 1.2.

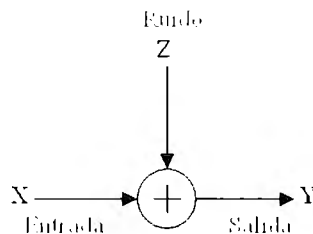


Figura 1.2 Esquema de Canal RGBA

El canal RGBA está definido por una probabilidad de transmisión dada por la ecuación 1.1.

$$P(y / X_k = x_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-x_i)^2}{2\sigma^2}} \quad 1.1$$

La varianza  $\sigma^2$  correspondiente es función de la razón señal a ruido, definida por la ecuación 1.2

$$\sigma^2 = \frac{1}{2} \left( \frac{E_b}{N_0} \right)^{-1} \quad 1.2$$

Donde  $E_b$  es la energía promedio utilizada para transmitir un símbolo binario y  $N_0$  es la densidad espectral de potencia monolateral de ruido aditivo.

### 1.2.6 Demodulador

El demodulador procesa la forma de onda transmitida y alterada por el canal, asignando a cada forma de onda un número que representa un estimado del símbolo transmitido. Por ejemplo, cuando se tiene modulación binaria, el demodulador procesa la forma de onda recibida y decide si fue transmitido un 1 ó 0. En el ejemplo anterior, se dice que el demodulador toma una decisión binaria. Otra alternativa es que el demodulador tome una decisión ternaria, es decir, decide si el bit transmitido es 0,1 o no toma decisión alguna dependiendo de la calidad aparente de la señal recibida.

### 1.2.7 Decodificador de control de errores

Usando la redundancia en la información transmitida, el decodificador de control de errores intenta llenar las posiciones donde no se obtuvo el bit deseado. Si vemos el proceso de decisión del demodulador como una forma de cuantización, observamos que la decisión binaria y ternaria son casos especiales de un demodulador que cuantiza a  $Q$  niveles, donde  $Q$  es mayor o igual a 2. En general, si el sistema de comunicación digital emplea una modulación  $M$ -aria donde se tienen  $M$  posibles símbolos para transmitir, cada uno correspondiendo a  $b = \log_2(M)$  bits, el demodulador tomará una decisión  $Q$ -aria donde  $Q$  es mayor o igual a  $M$ . Teniendo redundancia en el sistema la salida  $Q$ -aria del demodulador alimenta cada  $b/R$  segundos el decodificador de control de errores, quien reconstruye la secuencia de información original a partir del conocimiento del código empleado por el codificador de control de errores y la redundancia contenida en la información recibida.

### 1.2.8 Decodificador de fuente

El decodificador de fuente recibe la secuencia de salida compresada del decodificador de control de errores y, mediante el conocimiento del método de codificación del codificador de fuente, recupera la señal original de la fuente. Debido a los errores de decodificación de canal y las

posibles distorsiones introducidas por el codificador de fuente, la señal a la salida del decodificador de fuente es una aproximación de la salida de la fuente original.

### 1.3 Eficiencia de un sistema de transmisión

La calidad de la transmisión depende de la probabilidad de errores por elemento binario transmitido. La eficacia de un código corrector de errores se mide comparando las curvas de probabilidad obtenidas respectivamente a la salida del decodificador de canal y sobre el canal en ausencia de codificación. La diferencia entre las razones señal a ruido necesarias para lograr una probabilidad de error deseada es llamada ganancia de codificación dada en decibels. Estas mediciones se realizan después de la codificación y a la salida del canal en ausencia de decodificación.

### 1.4 Límite de Shannon

C.E.Shannon publicó en 1948 un trabajo en donde se relaciona la capacidad de información de un canal de comunicaciones con el ancho de banda y la relación señal a ruido. La ecuación matemática que define este límite esta dada por la fórmula 1.3

$$I = B \log_2 \left( 1 + \frac{S}{N} \right) \quad 1.3$$

Donde  $I$  es la capacidad de información expresada en bits por segundo,  $B$  la amplitud de banda expresada en Hertz y  $\left( \frac{S}{N} \right)$  la relación de potencias de señal a ruido.

Para entender este límite, se sabe que para la forma de acercarse a él es usar sistemas digitales con más de dos condiciones o símbolos de salida, lo que aumenta la rapidez de transmisión de información sobre el canal.

### 1.5 Conclusión

En este capítulo se explicó la manera en la que la información viaja desde transmisor hasta receptor. En resumen, la finalidad de un sistema de comunicación digital es la transmisión se realice con la mayor fidelidad posible a pesar de los errores adquiridos en el canal.

## Capítulo 2

---

# Códigos en bloque, códigos producto y decodificación iterativa

---

### 2.1 Introducción

En este capítulo se habla de la teoría de códigos en bloques definiendo longitud, dimensión, palabras de código, rendimiento, distancia mínima de Hamming y algunas propiedades principales.

También, se define el concepto de polinomio generador y el polinomio verificador de paridad para así abordar la forma sistemática de un código cíclico del cual también se dan definiciones y propiedades. En seguida, se describen los códigos BCH binarios dando su definición y características.

Posteriormente, se analizan los algoritmos de decodificación utilizados en este proyecto: decodificación óptima, decodificación por síndrome y decodificación algebraica, para esta última se aborda el método directo de Peterson que determina los coeficientes del polinomio localizador de errores, lo que a su vez hace necesario estudiar el algoritmo de Chien para encontrar raíces de un polinomio.

Más adelante se explican los códigos producto, su construcción y propiedades. Posteriormente se habla de los algoritmos de Chase y de Pyndiah, para finalizar el capítulo explicando el principio de turbo decodificación, su desempeño.

## 2.2 Códigos en bloques

### 2.2.1 Códigos en bloques lineales

Un código en bloques lineal está constituido por una cadena de bits de longitud  $n$ , donde la información a transmitir viene contenida en los primeros  $k$  símbolos y  $n-k$  representa la redundancia o bits de control, siendo siempre  $n > k$ . En un bloque de  $k$  bits de información hay  $2^k$  palabras de código. Los parámetros  $(n, k, d)$  definen completamente el código, siendo  $n$  la longitud de palabra de código,  $k$  la dimensión del mensaje,  $n-k$  los bits de redundancia,  $d$  la distancia mínima de Hamming,  $t$  el número máximo de bits a corregir y  $k/n$  el rendimiento del código.

La linealidad de códigos se puede demostrar si se analizan 2 palabras del código  $\mathbf{C}$  ( $\mathbf{C}_1$  y  $\mathbf{C}_2$ ). La suma de la codificación de las palabras deberá ser igual a la codificación de la suma de las mismas, es decir, teniendo 2 mensajes  $\mathbf{M}_1$  y  $\mathbf{M}_2$  podemos decir que el código es lineal sólo si se cumple con todas las condiciones expresadas en las ecuaciones 2.1 a 2.3.

$$\mathbf{C}_1 = f(\mathbf{M}_1) \quad 2.1$$

$$\mathbf{C}_2 = f(\mathbf{M}_2) \quad 2.2$$

$$\mathbf{C}_3 = f(\mathbf{M}_1 + \mathbf{M}_2) = f(\mathbf{M}_1) + f(\mathbf{M}_2) = \mathbf{C}_1 + \mathbf{C}_2 \quad 2.3$$

Es importante mencionar que en este subespacio vectorial, la suma corresponde a la función or exclusiva.

A partir de esta propiedad se puede generar un código en bloque del mensaje  $\mathbf{M}$  multiplicando este por la matriz generadora  $\mathbf{G}$  de  $k$  columnas y  $n$  renglones como muestra la ecuación 2.4.

$$\mathbf{C} = \mathbf{M}\mathbf{G} \quad 2.4$$

Con la finalidad de tener un código sistemático es necesario que la matriz generadora se construya a partir de la concatenación de la matriz identidad cuadrada de dimensión  $k$  y la matriz de paridad  $\mathbf{A}$ , siendo esta última de  $k$  líneas y  $n-k$  columnas como se muestra en la ecuación 2.5. Con esto, se logra que los elementos del mensaje  $\mathbf{M}$  sean los primeros  $k$  elementos del código  $\mathbf{C}$ .

$$\mathbf{G} = \left[ \left[ \mathbf{I}_k \right] \mid \left[ \mathbf{A} \right] \right] \quad 2.5$$

A todo código lineal se le asocia una matriz  $\mathbf{H}$  no nula definida como  $\mathbf{H} = \left[ \left[ \mathbf{A} \right] \mid \left[ \mathbf{I}_{n-k} \right] \right]$  llamada matriz de control de paridad de  $n-k$  renglones y  $n$  columnas, la cual se utiliza para obtener el



síndrome de error  $\mathbf{S}$ , el cual detecta errores de transmisión. La ecuación 2.6 define el síndrome a partir de las matrices y vectores ya mencionados.

$$\mathbf{S} = \mathbf{C}[\mathbf{H}]^t = \mathbf{M}[\mathbf{G}][\mathbf{H}]^t = 0 \quad 2.6$$

Como resultado se obtiene el vector  $\mathbf{S}$  con dimensión  $n-k$ . Al ser  $\mathbf{S}$  igual a cero, esto indica que no existieron errores en la transmisión de datos. La condición para que  $\mathbf{S}$  sea cero está dada por la ecuación 2.7.

$$[\mathbf{H}][\mathbf{G}]^t = [\mathbf{G}][\mathbf{H}]^t = 0 \quad 2.7$$

La distancia mínima de Hamming es una característica importante de un código, esta muestra el número mínimo de símbolos diferentes que debe haber entre 2 palabras de código para poder definir el poder de corrección  $t$ . El poder de corrección se obtiene a partir de la parte entera de la ecuación 2.8.

$$t = \left\lfloor \frac{(d_{\min} - 1)}{2} \right\rfloor \quad 2.8$$

### 2.2.2 Códigos cíclicos

Un código  $\mathbf{C}$  es cíclico si se cumple que cualquier vector dentro de este código al ser rotado se convierte en otro vector de dicho código.

Los códigos cíclicos utilizan una representación polinomial, por lo que el mensaje  $\mathbf{M}$  se convierte en el polinomio  $M(x)$  de grado  $k-1$  y la palabra de código  $\mathbf{C}$  en  $C(x)$  de grado  $n-1$  o menor. Para obtener el polinomio  $C(x)$  es necesario multiplicar el polinomio  $M(x)$  por el polinomio generador  $g(x)$ , este último de grado  $n-k$ , no nulo y divisor de  $x^n-1$ .

También, se forma el polinomio verificador de paridad o polinomio de control de código  $h(x)$  definido por la ecuación 2.9.

$$x^n-1=g(x).h(x) \quad 2.9$$

La codificación sistemática separa los bits de información de los bits de redundancia. Para realizar dicha codificación se multiplica el mensaje  $M(x)$  por  $x^{n-k}$ , se calcula  $B(x)$  como el resto de dividir  $x^{n-k}M(x)$  entre  $g(x)$  y finalmente se concatena  $B(x)$  con  $M(x)x^{n-k}$ .

### 2.2.3 Códigos BCH

Los códigos BCH dan una manera sistemática de construir códigos cíclicos con un gran poder de corrección de al menos  $t$  errores en un bloque de  $n$  símbolos codificados utilizando un algoritmo de decodificación simple.

Se puede precisar un código BCH en términos de su polinomio generador  $g(x)$ . Los parámetros principales de este tipo de códigos vienen definidos por las ecuaciones 2.9 a 2.13.

$$n = 2^m - 1 \quad 2.9$$

$$k \geq 2^m - 1 - mt \quad 2.10$$

$$n - k \leq mt \quad 2.11$$

$$d_{\min} \geq 2t + 1 \quad 2.12$$

$$t < \frac{n}{2} \quad 2.13$$

En el caso de los códigos BCH se trabaja con un polinomio primitivo irreducible de grado  $m$ , del cual se pueden obtener sus elementos no nulos al sustituir la variable  $x$  por  $\alpha$ . Estos últimos se construyen a partir de despejar el término  $\alpha^{m-1}$  del polinomio primitivo igualado a cero, reduciendo así los términos de grado mayor a  $\alpha^{m-1}$ . A partir de esto, se pueden conseguir los  $2^m-1$  elementos no nulos del subespacio vectorial perteneciente al polinomio primitivo.

Conociendo el polinomio primitivo despejado como  $\alpha^{m-1} = a_{m-2}\alpha^{m-2} + a_{m-3}\alpha^{m-3} + \dots + a_1\alpha^1 + a_0$  y recordando que  $a_i$  sólo puede tener valores de uno o cero, entonces los elementos primitivos se construyen de la siguiente manera:

$$\alpha^0 = 1$$

$$\alpha^1 = \alpha$$

M

$$\alpha^{m-1} = a_{m-2}\alpha^{m-2} + a_{m-3}\alpha^{m-3} + \dots + a_1\alpha^1 + a_0$$

$$\alpha^m = (a_{m-2}\alpha^{m-2} + a_{m-3}\alpha^{m-3} + \dots + a_1\alpha^1 + a_0)\alpha$$

M

$$\alpha^{2^m-1} = 1$$

### 2.3 Decodificación binaria de códigos BCH

Un proceso de decodificación debe encontrar los errores en las palabras recibidas y corregirlos. Para la decodificación de códigos BCH se considera que  $C(x)$  es la palabra emitida sobre un canal binario simétrico,  $R(x)$  la palabra recibida y  $E(x)$  el polinomio de errores, cuya relación se establece en la ecuación 2.14.

$$R(x) = C(x) + E(x) \quad 2.14$$

Si  $E(x)$  es igual a cero, entonces no se presentan errores; en caso contrario, la posición de coeficiente existente da la posición del error en la palabra recibida.

#### 2.3.1 Decodificación óptima

Este método de decodificación se basa en la comparación de la palabra recibida  $R$  con cada una de las  $2^k$  palabras de código, seleccionando la palabra con la distancia de Hamming más pequeña entre ellas. El número de comparaciones crece de manera exponencial a medida que  $k$  aumenta, lo que hace que este método sea útil sólo cuando  $k$  es pequeña.

#### 2.3.2 Decodificación por síndrome

La decodificación por síndrome tiene el mismo principio que el método óptimo. El síndrome  $S(C)$  es un vector de dimensión  $n-k$  llamado el síndrome de  $C$ . En este caso, en lugar de efectuar  $2^k$  comparaciones, se realizan únicamente  $2^{n-k}$ .

Como se mencionó anteriormente, si  $C(X)$  es una palabra del código  $C$  entonces el síndrome  $S(C(x))$  es igual a cero. En caso de recibir una palabra de código con error llamada  $R(x)$ , entonces se obtiene el síndrome de  $R(x)$  mediante la ecuación 2.15.

$$S(R(x)) = S(C(x) + E(x)) \quad 2.15$$

Aplicando las propiedades de linealidad se puede separar los síndromes (ecuación 2.16).

$$S(R(x)) = S(C(x)) + S(E(x)) \quad 2.16$$

Ya que el síndrome de la palabra de código es cero, finalmente se tiene que el síndrome del mensaje recibido es igual al síndrome del error (ecuación 2.17).

$$S(R(x)) = S(E(x)) \quad 2.17$$

Esto último indica la posición del error en la palabra recibida.

### 2.3.3 Decodificación algebraica

Para realizar la decodificación algebraica es necesario conocer el polinomio del síndrome  $S(x)$ , el polinomio localizador de errores  $\sigma(x)$  y el polinomio evaluador de errores  $\omega(x)$ . La relación entre estos está dada por la ecuación 2.18

$$\sigma(x)S(x) = \omega(x) \Big|_{x^{2t}} \quad 2.18$$

Considerando que el grado de  $\sigma(x)$  es igual al número de errores en la secuencia recibida cuyo máximo posible valor es  $t$ .

Las etapas de decodificación algebraica se resumen de la siguiente manera:

1. Cálculo del síndrome  $S$
2. Cálculo de  $\sigma(x) = 1 + \sigma_1x + \sigma_2x^2 + \dots + \sigma_t x^t$
3. Búsqueda de las raíces de  $\sigma(x)$

Como se mencionó anteriormente, el síndrome se define como el residuo de la división entre la palabra recibida y el polinomio generador.

### 2.4 Decodificación ponderada de códigos BCH.

La decodificación binaria de los códigos BCH es una técnica cuya principal ventaja es su poca complejidad, sin embargo, el desempeño de este tipo de decodificación se encuentra por debajo de lo óptimo. Ella consiste en la recepción de datos binarios mediante el monitoreo del canal, es decir, las muestras son recibidas en 2 niveles y la decodificación consiste en encontrar la palabra de código con distancia menor de Hamming.

Una alternativa con mejor desempeño es la decodificación ponderada, la cual ocupa mayor cantidad de información de las muestras recibidas del canal.

## 2.5 Códigos producto

A continuación se describen los códigos producto y su algoritmo iterativo de decodificación propuesto por R. Pyndiah.

### 2.5.1 Principio de códigos producto y sus propiedades

Los códigos producto fueron introducidos por P. Elias y se construyen mediante la concatenación en serie de 2 o más códigos en bloques lineales clásicos, esto permite tener una distancia de Hamming mayor. Si se consideran 2 códigos en bloques elementales  $C_1$  y  $C_2$  caracterizados por los parámetros  $(n_1, k_1, d_1)$  y  $(n_2, k_2, d_2)$  donde  $n_i$  es la longitud del código,  $k_i$  la longitud del mensaje y  $d_i$  la distancia mínima de Hamming de cada código.

El código producto  $C = C_1 \otimes C_2$  se presenta en forma de matriz  $[C]$  con  $n_1$  líneas y  $n_2$  columnas donde la información binaria forma una submatriz  $[M]$  de  $k_1$  líneas y  $k_2$  columnas; cada una de las  $k_2$  columnas de  $[M]$  es codificada por el código  $C_1$ , siendo posteriormente codificada cada una de las  $n_1$  líneas de la matriz de  $n_1 \times k_2$  mediante el código  $C_2$ . Los parámetros del código producto resultante son  $(n, k, d)$  donde  $n = n_1 \times n_2$ ,  $k = k_1 \times k_2$  y  $d = d_1 \times d_2$ . El rendimiento del código producto es igual al producto de los rendimientos de los códigos  $C_1$  y  $C_2$ .

Una propiedad importante de estos códigos es, si las  $n_2$  columnas de una palabra de código  $[C]$  son (por construcción) palabras de código  $C_1$ , entonces las  $n_1$  líneas de  $[C]$  son palabras del código  $C_2$ . Si a un código en bloques se le agrega un bit de paridad se crea un código BCH extendido. Al tomar 2 códigos en bloques extendidos, la distancia mínima del código producto correspondiente aumenta sin una modificación importante del rendimiento.

En la figura 2.1 se ilustra el principio de codificación de un código producto.

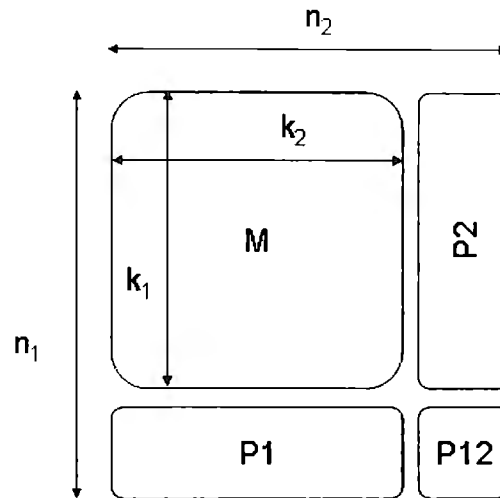


Figura 2.1 Codificación de un código producto

Donde:

$[M]$  es la matriz de información

$[P1]$  es la matriz de redundancia de las  $k_2$  columnas

$[P2]$  es la matriz de redundancia de las  $k_1$  columnas

$[P12]$  es la matriz de redundancia de  $[P1]$  y  $[P2]$

Realizar una decodificación directa de un código producto se vuelve un proceso complejo, por ello se sigue el principio “turbo” en el que se efectúa una decodificación iterativa de líneas y columnas. Con el fin de esperar desempeños óptimos se utiliza la decodificación ponderada.

### 2.5.2 Decodificación de entradas ponderadas: Algoritmo de Chase

El decodificador utiliza un algoritmo a entradas ponderadas para mejorar el proceso de turbo decodificación. En los últimos años se han venido desarrollando diversos algoritmos para esta decodificación ponderada, llegando a la conclusión de que los algoritmos de Chase tienen el mejor desempeño.

Chase propuso 3 algoritmos que permiten minimizar la probabilidad de error por palabra codificada (MVP). La decodificación MVP consiste en revisar todas las  $2^k$  palabras de código, esta revisión se ve optimizada de un algoritmo a otro reduciendo el número de comparaciones.

El primer algoritmo da mejores resultados pero requiere de un número considerable de secuencias de prueba. Este algoritmo toma en cuenta todas las palabras de código con una

distancia menor a  $d-1$  en comparación con la palabra de observación. El inconveniente de este algoritmo es que a medida que  $n$  y  $d$  aumentan, el número de secuencias de prueba crece debido a que esto está dado por la ecuación 2.19.

$$C_n^{\lfloor d/2 \rfloor} = \frac{n!}{(d/2)!(n-(d/2))!} \quad 2.19$$

El segundo algoritmo de Chase consiste en, dada una palabra recibida  $\mathbf{R}$ , se realiza una búsqueda de los  $\lfloor d/2 \rfloor$  símbolos menos confiables para formar  $2^{\lfloor d/2 \rfloor}$  secuencias de prueba. Este algoritmo tiene una mejor proporción entre complejidad y desempeño.

El tercer algoritmo de Chase es similar al segundo, únicamente se reduce el número de secuencias de prueba a  $\lfloor (d/2) + 1 \rfloor$ .

Considerando que  $\mathbf{E}=[e_1, e_2, \dots, e_n]$  es la palabra de código emitido con  $e_i = \pm 1$ . A la salida del canal de transmisión se tiene el vector de observación  $\mathbf{R}=[r_1, r_2, \dots, r_n]$  que forma la palabra binaria por monitoreo de componentes  $\mathbf{Y}=[y_1, y_2, \dots, y_n]$  donde  $y_i = \text{signo}(r_i)$ .

El algoritmo escoge un subconjunto de palabras de código para aplicar la decodificación MVP. Se hace la búsqueda de palabras de código dentro de una esfera de centro  $\mathbf{Y}$  y radio  $d-1$  donde  $d$  es la distancia mínima del código. Dentro de esta esfera hay por lo menos una palabra de código. Para encontrar las palabras de código  $C_i$  dentro de la esfera se forman las secuencias de prueba  $\mathbf{T}^i$  con  $1 \leq i \leq 2^{\lfloor d/2 \rfloor}$  para generar los vectores de prueba  $Y^i = Y \oplus T^i$  donde  $\mathbf{T}^i$  son vectores binarios de longitud  $n$  con pesos de Hamming inferior o igual a  $\lfloor d/2 \rfloor$ .

Al final de la decodificación se debe decidir la palabra de código  $\mathbf{D}$  que se encuentra a distancia mínima del vector  $\mathbf{R}$ . Esto se hace comparando las probabilidades de cada palabra de código y sus distancias con respecto al vector de observación. Finalmente se obtiene del algoritmo la palabra de código  $\mathbf{C}^i$  más probable emitida por un vector de entrada  $\mathbf{R}$  de datos ponderados.

Los algoritmos 2 y 3 toman en cuenta la información de confiabilidad de los símbolos recibidos para calcular las secuencias de prueba  $\mathbf{T}^i$ . La componente  $r_j$  del vector de observación está dada por la confiabilidad del símbolo  $e_j$ , esto hace posible esperar desempeños óptimos.

La confiabilidad de cada símbolo de la palabra emitida es calculada mediante el logaritmo de la razón de la probabilidad de que el símbolo que se emitió sea +1 dado un símbolo recibido sobre

la probabilidad de que se haya emitido un  $-1$  dado el símbolo que se recibió. A este logaritmo se le llama LRV y su valor absoluto mide la confiabilidad del símbolo recibido.

### 2.5.3 Decodificación a salidas ponderadas: Algoritmo de Pyndiah

El algoritmo de Pyndiah es la segunda parte del algoritmo de decodificación, este realiza una decodificación a salida ponderada. Para la decodificación iterativa de códigos producto, R. Pyndiah propuso un algoritmo iterativo a entradas y salidas ponderadas. La decisión óptima  $\mathbf{D} = [d_1, \dots, d_j, \dots, d_n]$  que se encuentra a distancia mínima del conjunto de palabras de código es dada por el algoritmo de Chase. El algoritmo de Pyndiah, mide la confiabilidad de cada símbolo de  $\mathbf{D}$  utilizando el logaritmo LRV.

La suma de las muestras a la entrada del decodificador  $\mathbf{R}$  y la información que da el decodificador  $\mathbf{W}$ , llamada información extrínseca y depende directamente de  $\mathbf{R}$ , proporciona el vector  $\mathbf{R}'$  como se muestra en la ecuación 2.20.

$$\mathbf{R}' = \mathbf{R} + \mathbf{W} \quad 2.20$$

La información extrínseca juega un papel muy importante en el funcionamiento óptimo de un turbo decodificador. Esta información mejora la decisión del decodificador ya que, cuando el proceso de decodificación es iterado, esta cantidad permite a cada nueva iteración mejorar el desempeño en términos de la tasa de errores binarios. Para la  $k$ -ésima decodificación elemental, la información extrínseca se produce por la decodificación precedente. La ecuación 2.21 representa la  $k$ -ésima decodificación.

$$r_j^k = r_j + \alpha^k \times w_j^{k-1} \quad 2.21$$

En la ecuación 2.21,  $\alpha^k$  representa el coeficiente de ponderación, función de la semi-iteración en curso, y  $r_j$  la información recibida por el canal de transmisión.

Otra forma de representar  $r_j'$  a partir de la decisión óptima  $\mathbf{D}$ , el vector de observación  $\mathbf{R}$  y la palabra concursante  $\mathbf{C}$  para cada símbolo de la decisión  $d_j$ , está dada por la ecuación 2.22.

$$r_j' = \left( \frac{|R - C|^2 - |R - D|^2}{4} \right) d_j \quad 2.22$$



Con lo que la información extrínseca se puede expresar con la ecuación 2.23.

$$r_j = \left( \frac{|R - C|^2 - |R - D_j|^2}{4} \right) d_j - r_j' \quad 2.23$$

En caso de que el algoritmo de Chase no permita encontrar la palabra de código  $C$  para la decisión  $d_j$ , el cálculo se hace con las ecuaciones 2.24 y 2.25.

$$r_j' = \beta d_j \quad 2.24$$

$$r_j = \beta d_j - r_j' \quad 2.25$$

La constante  $\beta$  al igual que la constante  $\left( \frac{|R - C|^2 - |R - D_j|^2}{4} \right)$  es siempre positiva, juega un papel importante en el desempeño del algoritmo y depende de la semi-iteración en curso, la razón señal a ruido, el código elemental y la modulación utilizada. En caso de tener una  $\beta$  variable se logra que la decodificación se acerque 0.15 dB al límite teórico de Shannon para una cierta decodificación.

#### 2.5.4 Decodificación iterativa de códigos producto a entradas y salidas ponderadas

La turbo decodificación de códigos producto es un algoritmo iterativo. El principio consiste en efectuar una decodificación de líneas (o columnas) seguida de una decodificación de columnas (o líneas). La decodificación del conjunto de líneas o columnas corresponde a una semi-iteración efectuada por el decodificador elemental a entradas y salidas ponderadas. El proceso iterativo de decodificación se muestra en la figura 2.2.

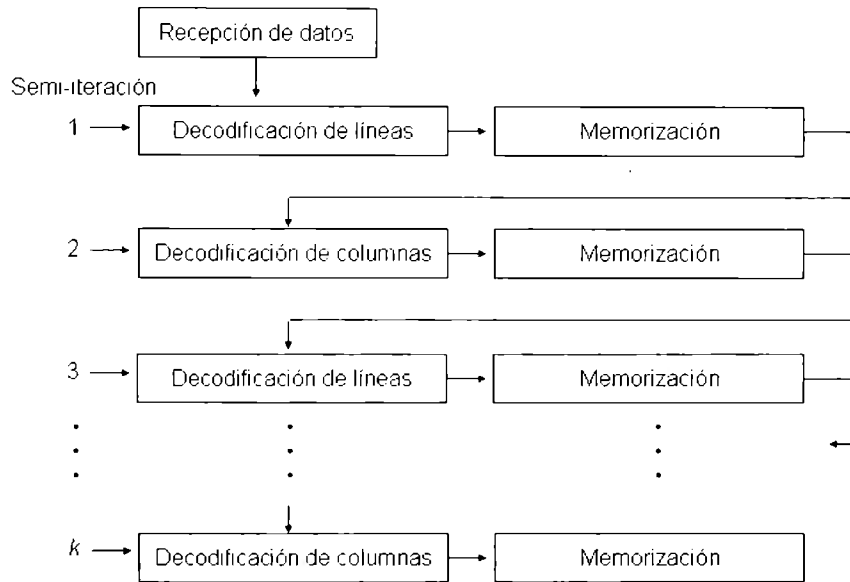


Figura 2.2 Proceso iterativo de decodificación

La decodificación iterativa consiste en colocar los decodificadores elementales en cascada. La figura 2.3 muestra el intercambio de información de una semi-iteración a otra, donde un vector representa una línea o columna del código producto.

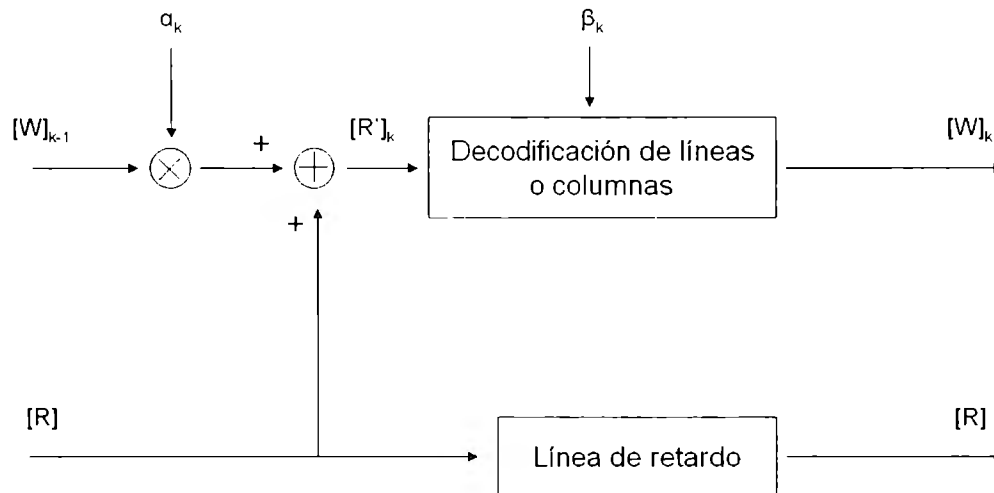


Figura 2.3 Intercambio de información entre semi-iteraciones

Esta decodificación elemental cuenta con dos entradas y dos salidas. La entrada  $[W]_{k-1}$  recibe la matriz de información extrínseca que viene de la decodificación anterior (que es igual a cero para

la primera decodificación). La entrada  $[\mathbf{R}]$  recibe la matriz que contiene el LRV proveniente de la salida del canal de transmisión. El decodificador hace la decodificación a entradas y salidas ponderadas de columnas o líneas de la matriz.

A la salida se obtiene la nueva matriz de información extrínseca llamada  $[\mathbf{W}]_k$ , obtenida por la diferencia entre la salida ponderada y la entrada ponderada, además del vector  $[\mathbf{R}]$  de entrada retardado.

Una forma de ver la secuencia de decodificación utilizando el algoritmo de Chase y calculando la información extrínseca viene dado por las ecuaciones 2.26 a 2.28.

$$[\mathbf{R}']_1 = [\mathbf{R}] + \alpha_1 [\mathbf{W}]_0 \quad 2.26$$

$$[\mathbf{R}']_2 = [\mathbf{R}] + \alpha_2 [\mathbf{W}]_1 \quad 2.27$$

⋮

$$[\mathbf{R}']_l = [\mathbf{R}] + \alpha_l [\mathbf{W}]_{l-1} \quad 2.28$$

$[\mathbf{W}]_{l-1}$  es la salida de la codificación elemental número  $(l-1)$ . El coeficiente  $\alpha_l$  evoluciona de una iteración a otra siendo cada vez optimizado, en la primera iteración su valor es cero, pero crece con cada nueva iteración. La información extrínseca es poco confiable en las primeras iteraciones pero el coeficiente  $\alpha_l$  se utiliza para ajustar su influencia.

### 2.5.5 Algoritmo de decodificación de códigos producto

Las etapas del algoritmo de decodificación son:

1. Localización de la posición de los  $\rho$  símbolos binarios menos fiables de  $\mathbf{R}'_i$  llamados  $I_1, I_2, \dots, I_\rho$ .
2. Generación de secuencias de prueba  $\mathbf{T}^Q$   $Q \in \{1, \dots, \tau\}$ , las cuales son combinaciones de los vectores de prueba elementales  $\mathbf{T}^j$  con "1" en la posición  $I_j$  y "0" en las demás posiciones.
3. Para cada vector de prueba  $\mathbf{T}^Q$ , calculamos  $\mathbf{Y}^Q = \mathbf{T}^Q \oplus \mathbf{R}_i$ .
4. Decodificación binaria de cada vector de prueba.
5. Cálculo de los parámetros de cada uno de los vectores de prueba y elección del vector elegido y demás vectores necesarios.
6. Cálculo de la nueva confiabilidad para cada uno de los símbolos de la palabra elegida.
7. Cálculo de información extrínseca.
8. Suma con la palabra recibida del canal.

## 2.6 El desempeño del decodificador

Para comparar los desempeños de dos o más turbo códigos en bloques es necesario conservar los mismos parámetros de simulación. Algunos de los parámetros que permiten comparar el desempeño son el rendimiento, la distancia mínima de Hamming, la ganancia asintótica y el límite de Shannon. El cálculo de la ganancia asintótica permite medir la eficiencia del algoritmo de codificación y se obtiene a partir de la distancia mínima  $d$  y del rendimiento  $R$  del código (ecuación 2.29).

$$G_a(dB) \approx 10 \log_{10}(R \cdot d) \quad 2.29$$

El límite de Shannon proporciona el desempeño del decodificador para una modulación de tipo MDP-4 sobre un canal Gaussiano mediante la ecuación 2.30.

$$G_s(dB) \leq 10 \log_{10} \left( \frac{2^{2R} - 1}{2R} \right) \quad 2.30$$

Este límite, otorga la facultad de comparar resultados de simulación.

Estudiando resultados de pruebas de eficiencia hechas a códigos BCH extendidos con distancia mínima  $d$  para cada uno de los códigos lineales igual a 4 y 6, y una distancia mínima  $D$  resultante de 16 y 36 para el código producto correspondiente, se observa que los códigos en bloques se sitúan en promedio a 1 dB del límite teórico de Shannon para una tasa de error binaria de  $10^{-5}$  en la cuarta iteración.

## 2.7 Conclusiones

En este capítulo, se estudiaron los códigos en bloques dando algunas definiciones y características como la matriz generatriz y la matriz de control de paridad. Se puntualizaron los códigos cíclicos para entender los códigos BCH y se explicaron los algoritmos de decodificación que se utilizan en el diseño del decodificador elemental.

Se detallaron los códigos producto y su decodificación a entradas y salidas ponderadas, se presentaron el principio de decodificación iterativa de códigos producto y el esquema del decodificador elemental.

Finalmente el estudio de desempeño realizado mostró que la ventaja principal de utilizar un código concatenado como código corrector de errores permite una decodificación ubicada a 1 dB del límite teórico de Shannon.



## Capítulo 3

---

# Arquitectura del decodificador elemental a entradas y salidas ponderadas

---

### 3.1 Introducción

En este capítulo se estudia la turbo decodificación de un código producto construido con dos códigos BCH extendidos con capacidad de corregir un error. Se presenta el algoritmo a entradas y salidas ponderadas, la arquitectura del decodificador y su desempeño.

Se muestran las propiedades más importantes del código BCH extendido (32,26,4), se describe la turbo decodificación del código producto construido con estos códigos BCH extendidos y se presenta el decodificador elemental o unidad de tratamiento que procesa dato por dato.

Más adelante, se analizan los bloques de recepción, procesamiento y emisión que componen el decodificador elemental mostrando el comportamiento de cada uno así como sus entradas y salidas principales.

### 3.2 El código BCH extendido (32,26,4)

En el código BCH extendido (32,26,4), se tiene  $n=32$ ,  $k=26$  y  $d=4$ . La redundancia que incluye un bit de paridad es de  $n-k=6$  y el rendimiento del código es de  $26/32$ . Tomando en cuenta la distancia mínima de este código ( $d=2t+2$ ), se concluye la corrección de un error ( $t=1$ ) en un bloque de 32 bits.

El código BCH está representado por una matriz de magnitud  $k \times k$ , es decir,  $k$  palabras de  $k$  bits de datos. Para cada una de las  $k$  líneas se efectúa la codificación que permite obtener la redundancia de línea  $R_L$  así como la paridad de línea  $P_L$ . Así, se obtiene un bloque de magnitud

$n \times k$  el cual se vuelve a codificar obteniendo la redundancia y paridad de las columnas  $R_C$  y  $P_C$ . Finalmente se logra una la matriz de magnitud  $n \times n$ , en este caso  $32 \times 32$ .

Este código producto tiene un rendimiento de  $R = R_{línea} \times R_{columna} = (26/32) \times (26/32)$ , siendo esto un rendimiento aproximado a  $R = 2/3$ , es decir, cada 2 bits de información se tiene un bit de redundancia. La distancia mínima del código producto es  $D = d_1 \times d_2 = 16$ . Un código extendido permite casi duplicar la distancia mínima, ya que esta pasa de 9 a 16.

Es importante recordar que las  $n$  columnas y las  $n$  líneas de la matriz son palabras del código, por lo que se deben efectuar  $n$  decodificaciones de columnas y  $n$  decodificaciones de líneas, lo que permite explotar la información extrínseca de los bits de redundancia.

### 3.3 Decodificación del código producto BCH $(32,26,4) \otimes (32,26,4)$

En esta parte se presentan las características más importantes de la decodificación de códigos producto.

#### 3.3.1 Principio de decodificación iterativa

La información recibida se presenta en forma de matrices. Cada dato es tratado dos veces, la primera por la decodificación de líneas y la segunda vez por la decodificación de columnas. Esto lleva a efectuar una decodificación iterativa.

Como se ha visto anteriormente, esta decodificación es a entradas y salidas ponderadas. Los símbolos de palabras procesadas son codificados en 5 bits por el demodulador: uno de signo y cuatro que representan el nivel de confianza con el que el bit de información es recibido. Entre mayor sea la confiabilidad, mayor la probabilidad de que la restitución del bit sea correcta.

La aportación del decodificador es llamada información extrínseca y representa la diferencia entre la entrada y la salida:  $[\mathbf{W}] = [\mathbf{F}] - [\mathbf{R}]$ . Esta información extrínseca es utilizada en la iteración siguiente después de ser multiplicada por el coeficiente  $\alpha$  que aumenta con cada nueva iteración. El coeficiente  $\alpha$  está caracterizado por un valor comprendido entre 0.4 y 0.7.



### 3.3.2 El algoritmo de decodificación

El algoritmo de decodificación utilizado en el decodificador elemental se divide en diferentes etapas presentadas a continuación:

- 1) Se reciben los vectores de datos [R] y [R'] provenientes de la unidad de memoria. [R] es el vector recibido inicialmente, proveniente del canal, y [R'] es el resultado de la última semi-iteración. En la primera semi-iteración [R]=[R'].
- 2) Se buscan los 5 símbolos menos fiables en la palabra recibida y se almacena su posición (MF1, MF2, MF3, MF4, MF5).

Durante esta etapa se realiza la división polinomial de los primeros 31 bits codificados (sin tomar en cuenta el bit de paridad) entre el polinomio generador,  $(x^5 + x^2 + 1)$ , el cual sirvió para codificar los datos. El resultado de esta división es un vector de dimensión 5 denominado síndrome. En el caso de que [R] sea una palabra de código el síndrome es cero, de lo contrario, indica la posición del bit a corregir.

- 3) Se generan 16 vectores de prueba por medio de combinaciones de una, dos o tres de las posiciones de los bits menos fiables de la palabra recibida. La forma en la que se obtienen estos 16 vectores de prueba se muestra en la tabla 3.1.
- 4) Se realiza la decodificación binaria de 16 vectores de prueba. Se calcula el síndrome asociado a cada uno de ellos de la manera  $S_i = S \oplus P_i$ , donde  $S_i$  es el síndrome del vector de prueba  $i$ -ésimo,  $S$  el síndrome de la palabra recibida y  $P_i$  la posición obtenida de la combinación para el  $i$ -ésimo vector de prueba. Este síndrome del vector de prueba da la posición del bit que puede haber sido recibido erróneamente.
- 5) Calculamos la métrica  $M^Q$  que está representada por la ecuación 3.1. De esta manera se calcula la distancia entre los vectores y la palabra recibida [R'] de cada palabra decodificada  $C^Q$ .

$$M^Q = \sum_{\substack{j=0 \\ c_j \neq y_j}}^{n-1} |r_j| \quad 3.1$$

1.	$Y^0$	0
2.	$Y^1$	MF1
3.	$Y^2$	MF2
4.	$Y^3$	MF3
5.	$Y^4$	MF4
6.	$Y^5$	MF5
7.	$Y^6$	MF1, MF2
8.	$Y^7$	MF1, MF3
9.	$Y^8$	MF1, MF4
10.	$Y^9$	MF1, MF5
11.	$Y^{10}$	MF1, MF2, MF3
12.	$Y^{11}$	MF1, MF2, MF4
13.	$Y^{12}$	MF1, MF2, MF5
14.	$Y^{13}$	MF1, MF3, MF4
15.	$Y^{14}$	MF1, MF3, MF5
16.	$Y^{15}$	MF1, MF4, MF5

Tabla 3.1

- 6) Se selecciona entre  $C^Q$  la palabra de código  $C^D$ , de tal forma que se encuentre a la distancia mínima de la palabra recibida  $R'$ . Es así como se obtiene el resultado binario de la decodificación.
- 7) Enseguida se asigna una medida de fiabilidad a cada bit de la decisión tomada, esto es el objetivo del algoritmo de Pyndiah. Esto se hace con el fin de poder clasificar los vectores de prueba en orden creciente de acuerdo a sus métricas.

Para cada posición  $j$ , se determina la palabra concursante, es decir la de menor métrica, para la cual la decisión binaria sobre el símbolo  $j$  es distinta de la tomada para la palabra elegida  $M^D$ . El cálculo de la confiabilidad del símbolo  $j$  es efectuado por la ecuación 3. 2.

$$|F| = \frac{M^C - M^D}{4} \quad 3.2$$

- 8) Se calcula la información extrínseca  $w_j$  asociada a cada uno de los bits de  $c_j^D$  de  $C^D$  como  $[W]=[F]-[R']$ .

9) Cálculo del nuevo valor de  $[R']=[R]+\alpha [W]$ .

### 3.3.3 Decodificador elemental

El decodificador elemental tiene como función la decodificación de líneas o columnas de la matriz codificada. Este genera el conjunto de señales de lectura, escritura y selección del modo de funcionamiento de las memorias. En esta parte se considera la arquitectura del decodificador desde un punto de vista funcional, el cual viene agrupado en 5 grandes bloques como muestra la figura 3.1.

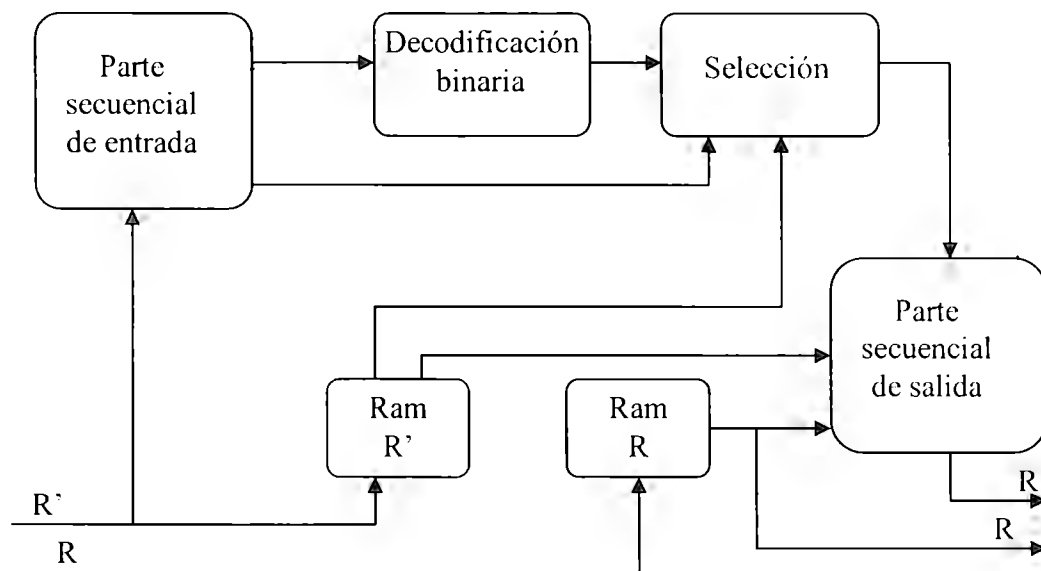


Figura 3.1

1. Parte secuencial de entrada: son bloques que trabajan de manera independiente procesando los datos binarios que son recibidos.
2. Decodificación binaria: determina la palabra de código óptima para el vector binario tratado.
3. Selección: elección de palabra elegida y concurrente.
4. Parte secuencial de salida: se realizar poco a poco la emisión de los datos y la generación de información para las semi-iteraciones consecutivas.

5. Parte de memoria: esta parte esta compuesta de dos memorias divididas en tres RAMs de  $32 \times 5$  cada una.

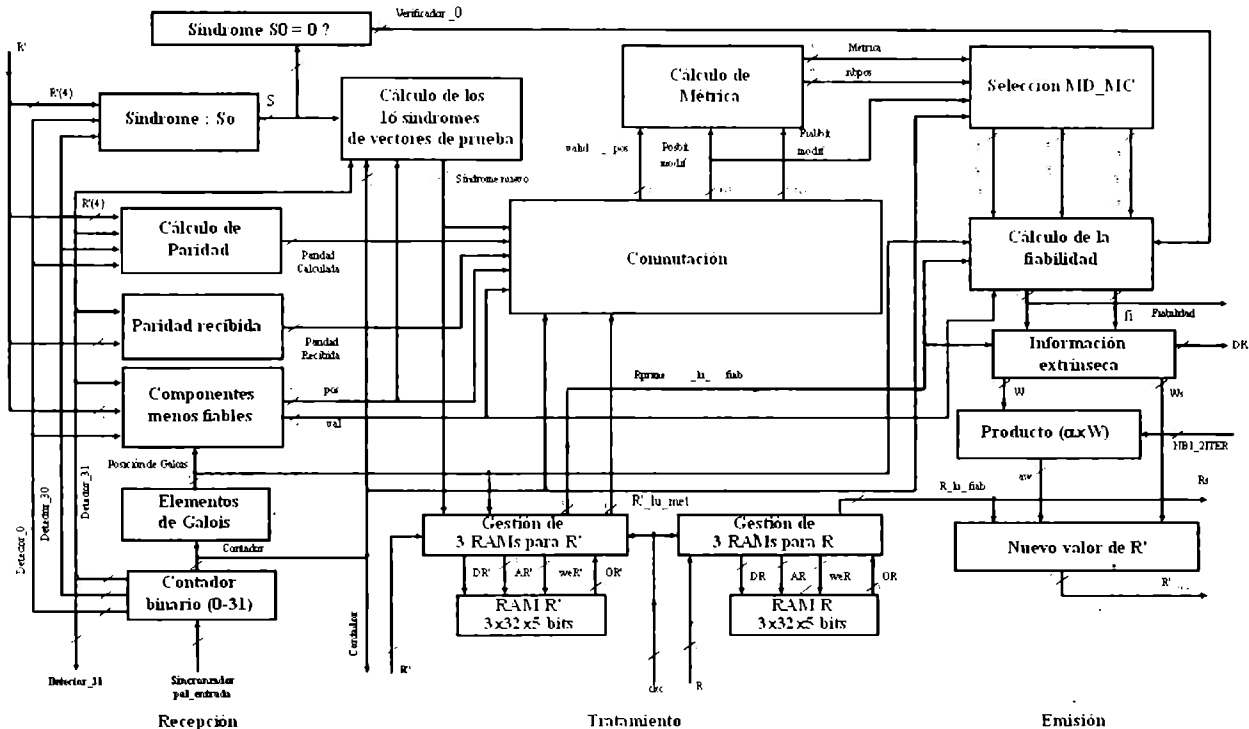


Figura 3.2

En la figura 3.2 se muestra detalladamente el diagrama a bloques de un decodificador elemental compuesto por los bloques de recepción, tratamiento y emisión. Las operaciones de la parte de tratamiento no pueden ser llevadas a cabo hasta terminada por completo la recepción de la palabra, de igual manera, una vez que termina el tratamiento se pasa a la parte de emisión de símbolos.

Para cada una de estas partes se necesitan 32 períodos de reloj, lo que genera una latencia del decodificador elemental de 64 períodos de reloj, correspondiente al tiempo que pasa desde la recepción del primer símbolo de la palabra  $[R']$  hasta la emisión del primer símbolo de la nueva palabra  $[R']$ .

Recepción de la palabra.- En esta parte se llevan a cabo la recepción de  $[R]$  y  $[R']$ , el cálculo del síndrome, revisión de paridad recibida y calculada, búsqueda de los cinco componentes menos confiables y almacenamiento de  $[R']$  y  $[R]$ .

Tratamiento de la palabra.- Es la segunda etapa de la decodificación. En esta etapa se ejecuta las funciones de cálculo del síndrome de los 16 vectores de prueba, cálculo de la posición del bit a corregir de cada vector, cálculo de las 16 métricas correspondientes a los 16 vectores de prueba, selección de la palabra elegida ( $M^D$ ) y selección de las palabras concursantes ( $M^C$ ).

Emisión de la palabra.- En esta tercera etapa se calcula la ponderación y se efectúa la transmisión de la nueva  $[R']$  y de  $[R]$ , aquí se realizan las siguientes funciones para cada símbolo de  $[R']$ : búsqueda de una métrica concursante ( $M^C$ ), cálculo de la información extrínseca ( $W$ ), cálculo y transferencia del nuevo símbolo de ( $R'$ ) y de ( $R$ ) inicial.

### 3.3.3.1 Bloques de recepción

#### 3.3.3.1.1 Contador binario de 0-31

El circuito contador (figura 3.3) permite sincronizar el decodificador, con lo que se determina la posición de cada símbolo recibido de la palabra. Este contador trabaja con una señal de reset y una entrada de control o sincronización para producir el conteo.

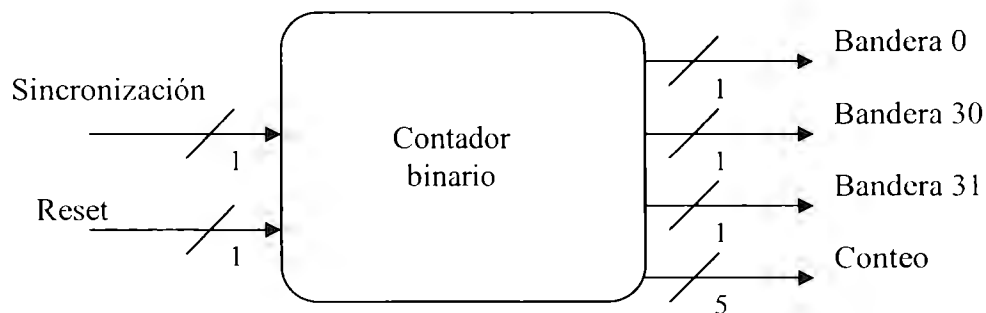


Figura 3.3 Circuito contador

Las salidas del contador son: el valor actual del contador (5 bits) y las señales de detección de 0, 30 y 31. Estas señales determinan la llegada de la palabra (detección de cero), el fin de la palabra y el bit de paridad (detección de 30 y 31). Entre el instante 0 y el 30 se encuentra la recepción de los bits de información con su redundancia. Los cambios son detectados por el frente de reloj.

### 3.3.3.1.2 Elementos del cuerpo de Galois

El bloque de los elementos del cuerpo de Galois (figura 3.4) funciona en conjunto con el contador. Este bloque hace corresponder la salida del contador (posición del símbolo recibido) y su equivalencia bajo la forma de un elemento en el cuerpo de Galois. Esta conversión se realiza en base a la sección 2.2.3 con ayuda de un polinomio irreducible, esto facilita la decodificación binaria de los vectores de prueba, la tabla 3.2 muestra la conversión entre el contador y el cuerpo de Galois.

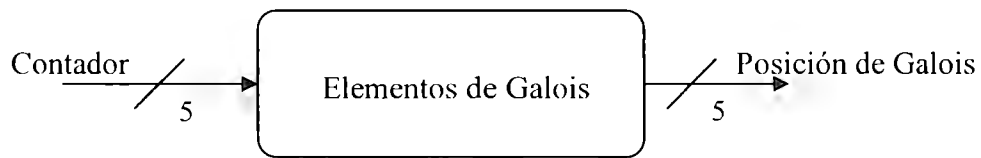


Figura 3.4 Circuito conversor de elemento de Galois

### 3.3.3.1.3 Los 5 componentes menos confiables

Este bloque (figura 3.5) analiza la confiabilidad de los símbolos recibidos, sin tomar en cuenta la paridad de la palabra y busca las 5 posiciones menos confiables, a estas posiciones se les asigna el nombre de: MF1, MF2, MF3, MF4 y MF5, donde MF1 es la posición del bit menos confiable de la palabra recibida.

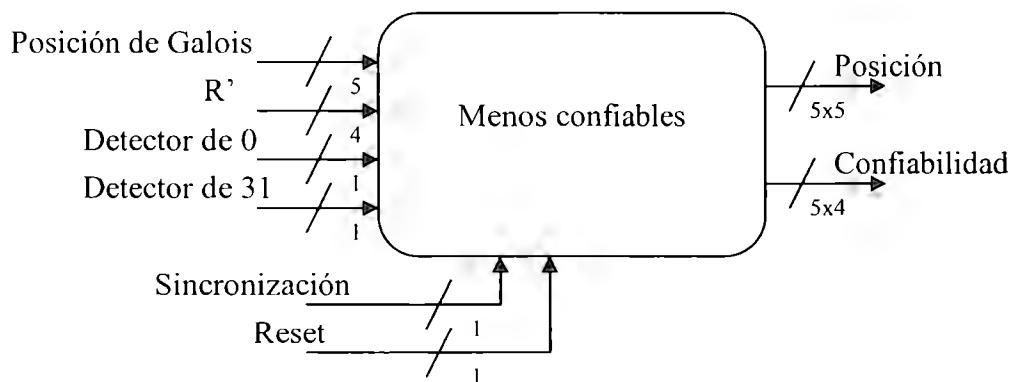


Figura 3.5 Circuito selector de símbolos menos fiables

Contador <sub>dec</sub>	Contador <sub>bin</sub>	Polinomio Primitivo	Galois <sub>bin</sub>	Galois <sub>hex</sub>
0	00000	$\alpha^{30} = \alpha^4 + \alpha$	10010	12
1	00001	$\alpha^{29} = \alpha^3 + 1$	01001	09
2	00010	$\alpha^{28} = \alpha^4 + \alpha^2 + \alpha$	10110	16
3	00011	$\alpha^{27} = \alpha^3 + \alpha + 1$	01011	0B
4	00100	$\alpha^{26} = \alpha^4 + \alpha^2 + \alpha + 1$	10111	17
5	00101	$\alpha^{25} = \alpha^4 + \alpha^3 + 1$	11001	19
6	00110	$\alpha^{24} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha$	11110	1E
7	00111	$\alpha^{23} = \alpha^3 + \alpha^2 + \alpha + 1$	01111	0F
8	01000	$\alpha^{22} = \alpha^4 + \alpha^2 + 1$	10101	15
9	01001	$\alpha^{21} = \alpha^4 + \alpha^3$	11000	18
10	01010	$\alpha^{20} = \alpha^3 + \alpha^2$	01100	0C
11	01011	$\alpha^{19} = \alpha^2 + \alpha$	00110	06
12	01100	$\alpha^{18} = \alpha + 1$	00011	03
13	01101	$\alpha^{17} = \alpha^4 + \alpha + 1$	10011	13
14	01110	$\alpha^{16} = \alpha^4 + \alpha^3 + \alpha + 1$	11011	1B
15	01111	$\alpha^{15} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1$	11111	1F
16	10000	$\alpha^{14} = \alpha^4 + \alpha^3 + \alpha^2 + 1$	11101	1D
17	10001	$\alpha^{13} = \alpha^4 + \alpha^3 + \alpha^2$	11100	1C
18	10010	$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha$	01110	0E
19	10011	$\alpha^{11} = \alpha^2 + \alpha + 1$	00111	07
20	10100	$\alpha^{10} = \alpha^4 + 1$	10001	11
21	10101	$\alpha^9 = \alpha^4 + \alpha^3 + \alpha$	11010	1A
22	10110	$\alpha^8 = \alpha^3 + \alpha^2 + 1$	01101	0D
23	10111	$\alpha^7 = \alpha^4 + \alpha^2$	10100	14
24	11000	$\alpha^6 = \alpha^3 + \alpha$	01010	0A
25	11001	$\alpha^5 = \alpha^2 + 1$	00101	05
26	11010	$\alpha^4 = \alpha^4$	10000	10
27	11011	$\alpha^3 = \alpha^3$	01000	08
28	11100	$\alpha^2 = \alpha^2$	00100	04
29	11101	$\alpha^1 = \alpha$	00010	02
30	11110	$\alpha^0 = 1$	00001	01
31	11111	$\alpha^{-\infty} = 0$	00000	00

Tabla 3.2

Los cuatro bits de peso de R' indican la fiabilidad del símbolo que es recibido en cada período de reloj.

En este bloque llega cada símbolo y se compara su confiabilidad con la de los 5 bits ya almacenados (los primeros cinco para el caso de las primeras comparaciones) y si esta es menor se almacena su posición, de otro modo no se toma en cuenta. Esto se realiza con cada símbolo de información que sea recibido.

### 3.3.3.1.4 Paridad calculada

El bloque de paridad calculada (figura 3.6) genera, conforme a la llegada de los primeros 31 símbolos, la paridad de la palabra recibida.

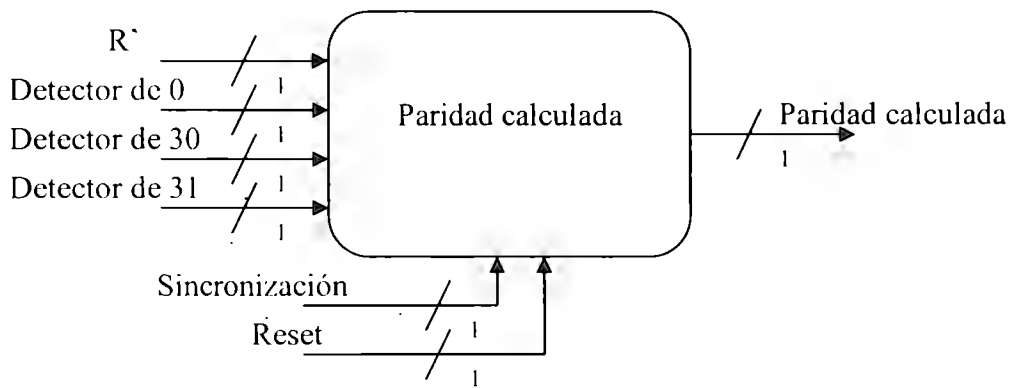


Figura 3.6 Circuito generador de paridad

### 3.3.3.1.5 Paridad recibida

El bloque de paridad recibida (figura 3.7) detecta y almacena el símbolo de la paridad recibida del vector  $R'$  así como su confiabilidad gracias al detector del bit 31.

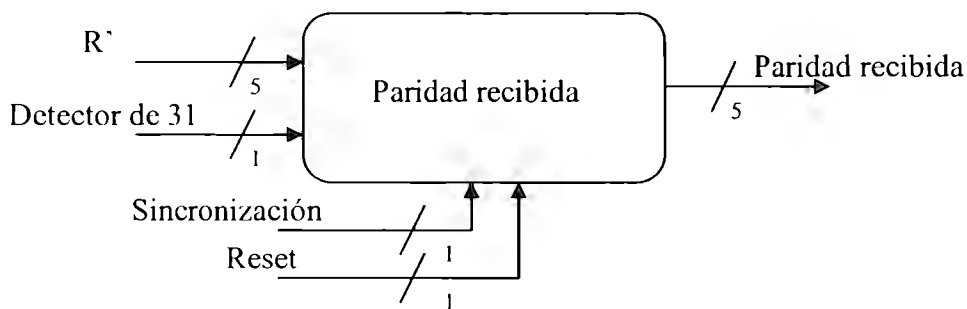


Figura 3.7 Circuito detector de paridad



3.3.3.1.6 Síndrome  $S_0$

En este bloque (figura 3.9) se determina el síndrome  $S_0$  el cual es residuo de la división de la palabra recibida vista como polinomio entre el polinomio generador del código ( $g(x) = x^5 + x^2 + 1$ ). Esto se hace tomando el bit mas significativo de  $R'$  y siguiendo las ecuaciones 3.3 a 3.7 que representan el funcionamiento del circuito presentado en la figura 3.8.

$$S(0) = \text{MSB XOR } S_0(4) \tag{3.3}$$

$$S(1) = S_0(0) \tag{3.4}$$

$$S(2) = S_0(1) \text{ XOR } S_0(4) \tag{3.5}$$

$$S(3) = S_0(2) \tag{3.6}$$

$$S(4) = S_0(3) \tag{3.7}$$

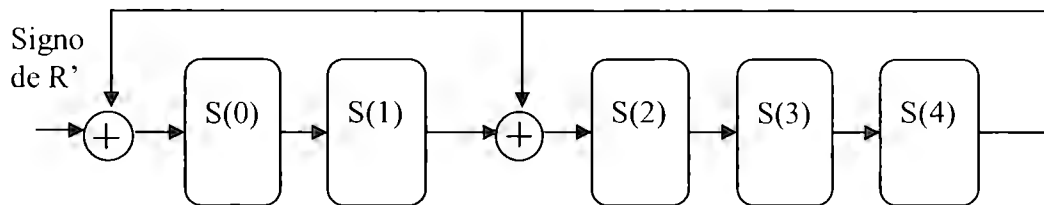


Figura 3.8 Cálculo de síndrome

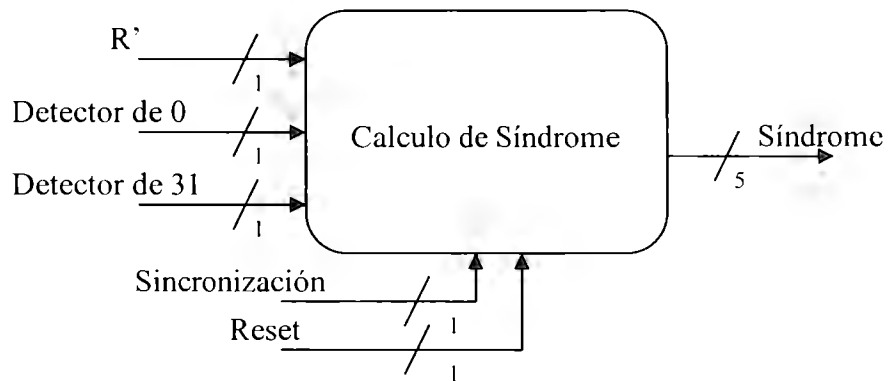


Figura 3.9 Circuito generador de síndrome

### 3.3.3.2 Bloques de tratamiento

#### 3.3.3.2.1 Cálculo de los 16 síndromes de los vectores de prueba

Los vectores de prueba con construidos a partir de las posiciones de los cinco componentes menos fiables (MF1, MF2, MF3, MF4 y MF5) y el primer síndrome  $S_0$ . Estos vectores con generados invirtiendo uno, dos o tres bits de los 5 menos fiables del vector recibido. Estos 16 vectores son tratados secuencialmente en dos períodos de reloj. La figura 3.10 esquematiza el cálculo de síndromes de vectores de prueba.

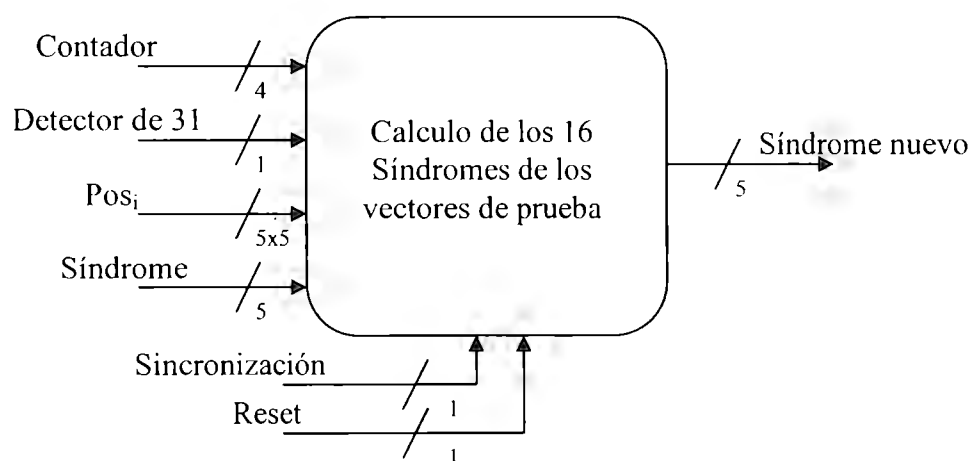


Figura 3.10 Circuito generador de síndromes de vectores de prueba

#### 3.3.2.2 Conmutación

Este bloque (figura 3.11) se encarga de encontrar entre todas las fiabilidades emitidas, aquellas que se deben tomar en cuenta en función del valor del contador y de la igualdad entre la paridad recibida y la calculada.

En la salida se tienen las fiabilidades y las posiciones de los bits a invertir en el vector de prueba, del bit corregido y del bit de paridad. El cálculo de la posición del bit de paridad no es necesario ya que esta siempre es la posición 31.

La tabla 3.3 muestra el estado del contador, los bits a invertir y su métrica.

Contador	Bits a invertir	Métrica
0000	0	0 ó 2
0001	1	1
0010	1	1
0011	1	1
0100	1	1
0101	1	1
0110	2	0 ó 2
0111	2	0 ó 2
1000	2	0 ó 2
1001	2	0 ó 2
1010	3	1
1011	3	1
1100	3	1
1101	3	1
1110	3	1
1111	3	1

Tabla 3.3

Para la métrica 0 y 2, existen 0 ó 2 bits invertidos en el vector de prueba. El estado del contador corresponde a: 0000, 0110, 0111, 1000, 1001. En este caso el síndrome es diferente de cero y la paridad recibida es la misma que la paridad calculada.

Para la métrica 1, existen uno o tres bits invertidos. El estado del contador corresponde a: 0001, 0010, 0011, 0100, 0101, 1010, 1011, 1100, 1101, 1110, 1111. Aquí el síndrome es diferente de cero y la paridad recibida es diferente a la calculada.

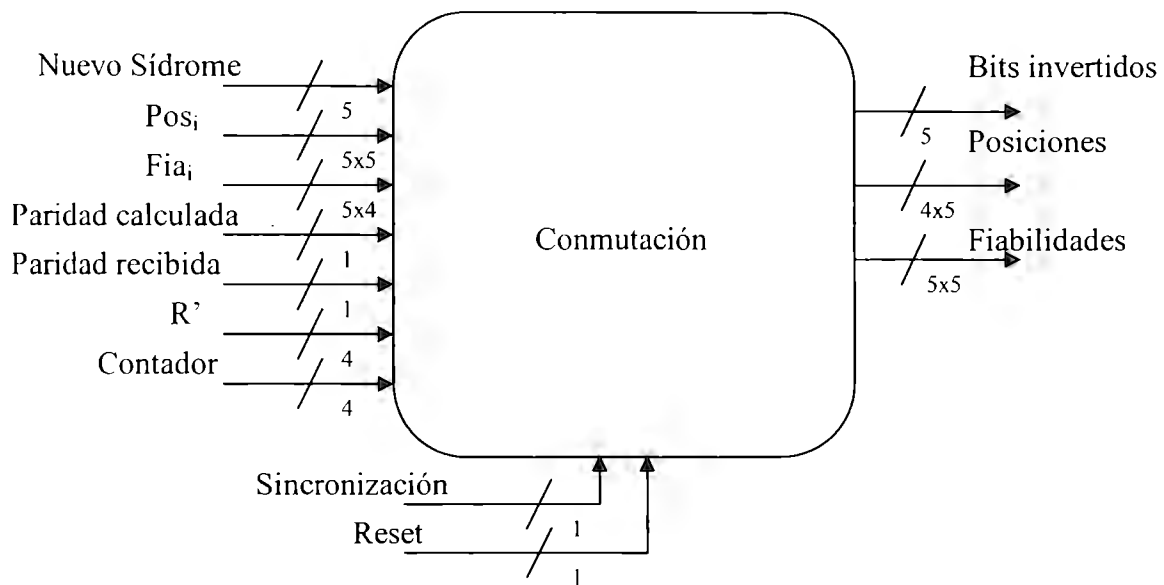


Figura 3.11 Circuito de conmutación

### 3.3.3.2.3 Cálculo de métrica

En este bloque (figura 3.12), la entrada esta constituida por las fiabilidades correspondientes a los bits invertidos en el vector de prueba considerado. Cualquiera que sea el número de bits invertidos, este bloque compara las posiciones de dichos bits con la del bit corregido por el síndrome inicial con el fin de no tomar en cuenta dos veces la misma fiabilidad. Cada métrica es la suma de las fiabilidades de los bits invertidos.

A la salida de este bloque se tiene el cálculo de la métrica correspondiente así como la actualización del registro de los bits invertidos (nbpos).

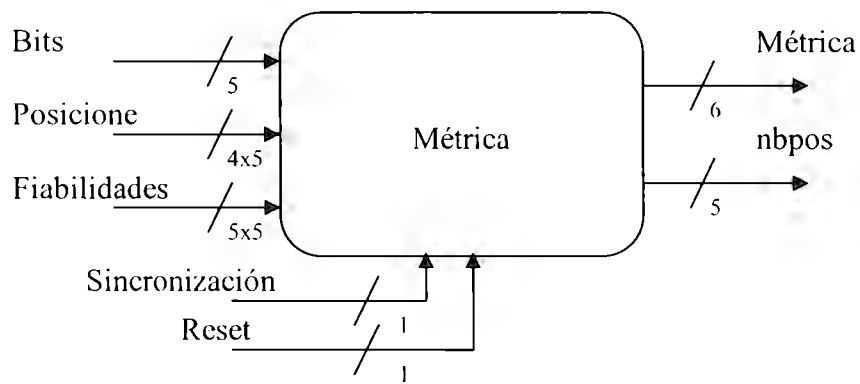


Figura 3.12 Circuito calculador de métrica

### 3.3.3.2.4 Cálculo de la palabra decidida y las concursantes

Este bloque (figura 3.13) selecciona el vector de prueba de métrica más pequeña  $M^D$  y los tres vectores de prueba de métrica inmediatamente inferior,  $M^{c1}$ ,  $M^{c2}$ ,  $M^{c3}$ . Estos vectores de prueba son entregados al bloque de cálculo de fiabilidad. Paralelamente este bloque entrega las posiciones alteradas a razón de la palabra recibida con las señales de validación nbpos del bloque anterior.

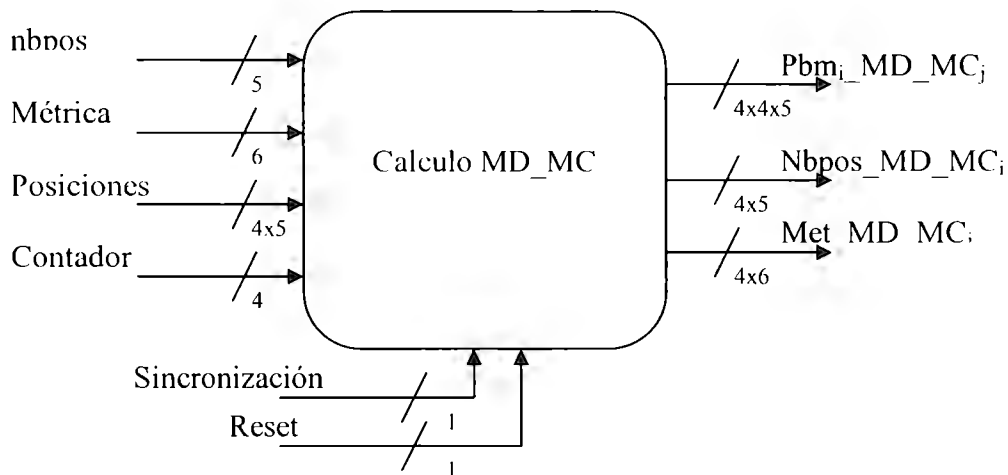


Figura 3.13 Circuito selector de vectores de prueba con métrica menor

Se analizan los 16 vectores durante los 32 periodos de reloj. En el segundo periodo de reloj se compara la métrica del vector actual con las de los vectores precedentes. Al final de los 32

periodos de reloj se entregan a la salida del bloque los valores correspondientes a las palabras decidida MD y concursantes  $MC_i$ .

### 3.3.3.3 Bloques de emisión

#### 3.3.3.3.1 Cálculo de la fiabilidad

Este bloque calcula la fiabilidad de los símbolos del vector recibido en función de la palabra decidida y las concursantes. El valor de la fiabilidad es igual a la diferencia entre cada palabra concursante y la decidida ( $M^c - M^D$ ).

Para el caso en el que no exista palabra concursante para una posición  $j$ , es decir, cuando las palabras concursantes tienen un símbolo idéntico al de la palabra decidida en esa posición, se atribuye a ese símbolo una fiabilidad  $\beta$  que depende de la iteración en curso.  $\beta$  es un parámetro muy importante en la decodificación de códigos producto ya que tiene influencia sobre el desempeño obtenido.

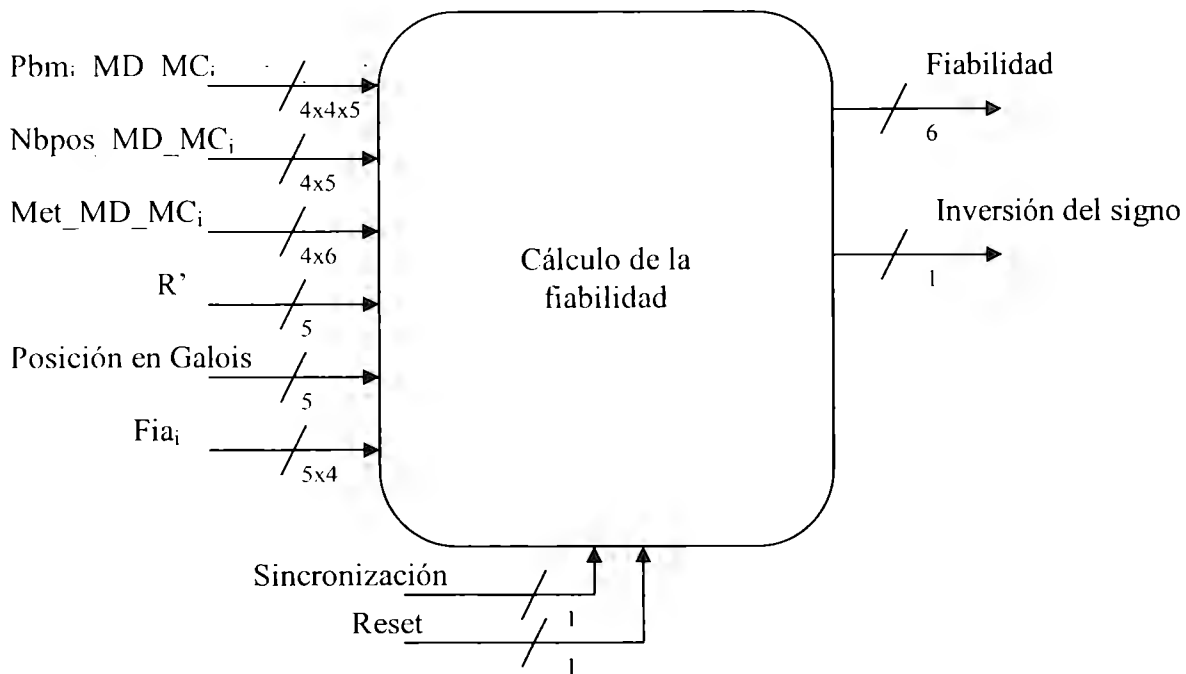


Figura 3.14 Circuito de cálculo de fiabilidad

3.3.3.3.2 Cálculo de la información extrínseca

En este bloque (figura 3.15) se calcula la información extrínseca  $W$ , la cual es la aportación del decodificador a la semi-iteración siguiente.

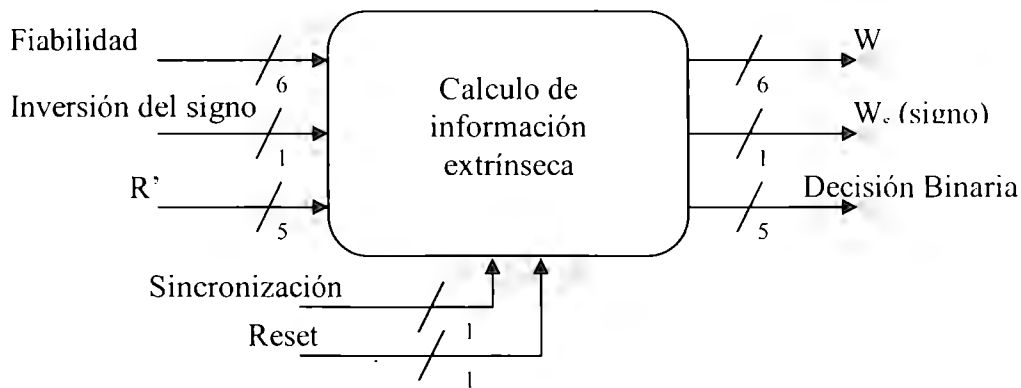


Figura 3.15 Circuito de cálculo de información extrínseca

3.3.3.3.3 Producto de  $\alpha$  por información extrínseca

Este bloque (figura 3.16) efectúa la multiplicación de  $\alpha$  por la información extrínseca  $W$ .  $\alpha$  es el coeficiente de contra-reacción que pondera la contribución de la información extrínseca, este toma un valor cercano a cero para las primera iteraciones y va aumentando en las siguientes iteraciones.

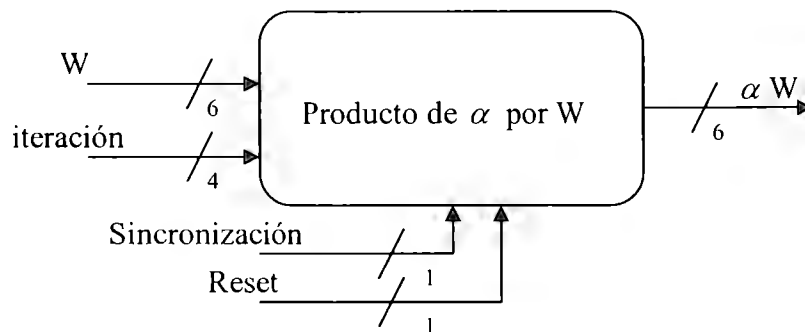


Figura 3.16 Circuito multiplicador de  $\alpha$  por  $W$

### 3.3.3.3.4 Cálculo del nuevo valor de $R'$

En este bloque (figura 3.17) se efectúa la adición  $R + \alpha W$ . El resultado de esta suma es el nuevo valor de  $R'$  para la semi-iteración siguiente. Para la decisión del nuevo valor se tienen que tomar en cuenta los signos de los operandos, si tienen el mismo signo se efectúa una suma y si son de signo contrario una resta guardando el signo del de la fiabilidad más grande.

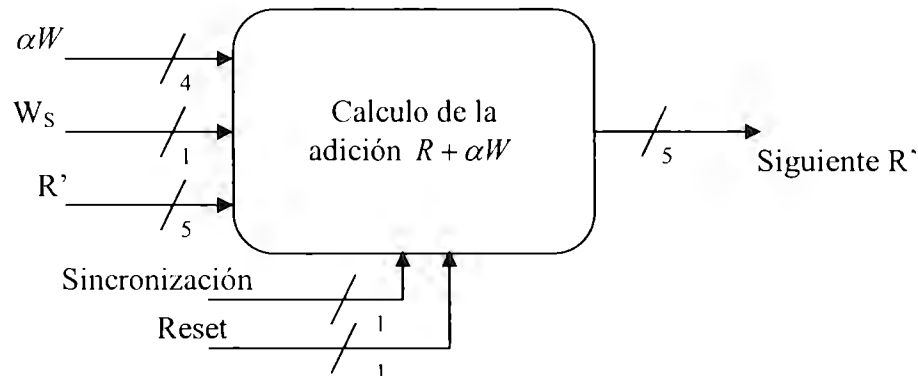


Figura 3.17 Circuito sumador de  $R + \alpha W$

### 3.3.3.4 Memorias para almacenamiento de $R'$ y $R$

La arquitectura del decodificador requiere de dos módulos de memoria, uno para almacenar  $R'$  y otro para  $R$ . Cada módulo está compuesto de tres memorias RAM de  $32 \times 5$  bits y un bloque de administración que genera las señales de control para las memorias como escritura o lectura.

Las memorias RAM funcionan de manera circular, esto es, si la primera RAM funciona para los datos de recepción, la segunda para los de tratamiento y la tercera para los de emisión, al finalizar los 32 pulsos siguientes la primera funcionara dando servicio a la etapa de tratamiento, la segunda a emisión y la tercera recibirá datos nuevos y así sucesivamente.

## 3.4 Conclusiones

En este capítulo, se analizaron los diferentes parámetros del código BCH extendido (32,26,4). Se representó una palabra de código, y se calculó la distancia mínima de un código producto y su rendimiento. Se estudio el algoritmo de decodificación y se presentó el decodificador elemental o unidad de tratamiento compuesto de tres bloques que se han analizado desde un punto de vista



funcional. Finalmente se detalló el funcionamiento de cada uno de estos bloques señalando sus características y sus señales de entrada y salida.



## Validación con C++

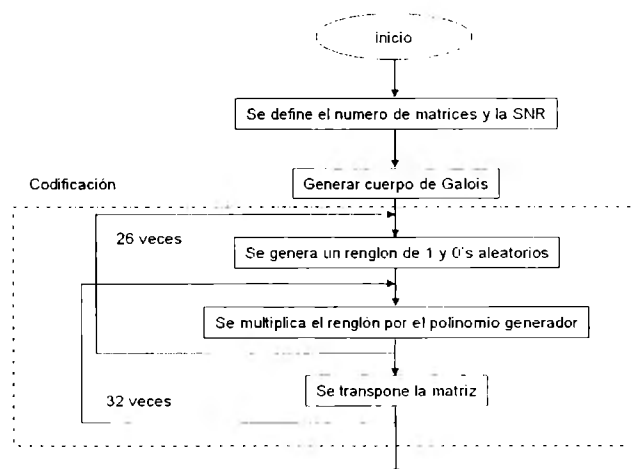
---

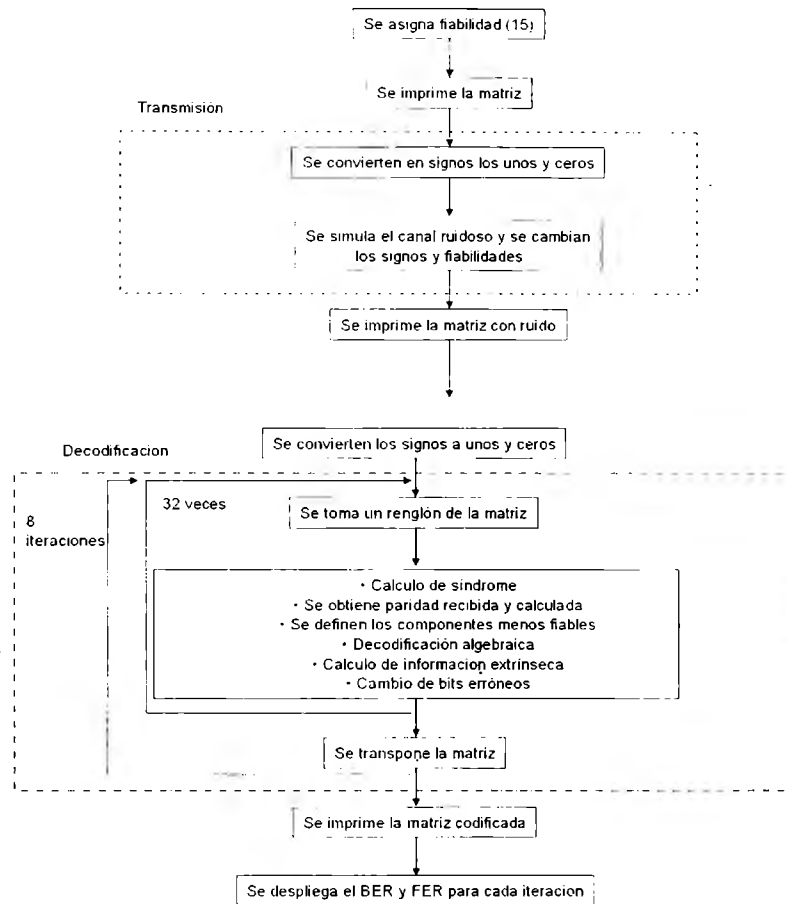
### 4.1 Introducción

Para validar la operación del proyecto, en este capítulo se describe el desarrollo del programa de validación en lenguaje C++, explicando cada una de las partes en las que está dividido. Esta explicación corrobora el trabajo realizado en VHDL en la sección anterior. El programa está dividido en 3 partes: generación y codificación de datos, transmisión sobre un canal gausiano y decodificación. Con este programa es posible observar que se corrigen errores en las diferentes iteraciones de la decodificación, obteniendo así la tasa de error binario a una relación señal a ruido dada para un determinado número de datos a transmitir.

### 4.2 Descripción del programa

A continuación se muestra el diagrama de flujo del programa utilizado para la validación en lenguaje C++.





#### 4.2.1 Generación de datos y codificación

La primera parte del programa, genera la matriz de datos de tamaño 26 por 26 de manera aleatoria, es decir, 1s y 0s de manera fortuita. Una vez generada esta matriz, se codifica mediante el polinomio generador  $x^5+x^2+1$  de manera horizontal y vertical generando así una nueva matriz de tamaño 32 por 32.

Los datos son generados por renglones, los cuales se codifican mediante la multiplicación de éste por el polinomio generador. Esto se hace de manera sucesiva 26 veces. Al finalizar este proceso, la traspuesta de esta matriz es leída por el programa para procesar la segunda codificación multiplicando por el mismo polinomio generador. Finalmente se tienen los datos codificados con su respectiva redundancia.

#### 4.2.2 Transmisión

En la segunda parte del programa se simula la transmisión de los datos sobre un canal con ruido blanco gaussiano aditivo y una modulación QPSK.

Para que los datos se vean alterados por el canal, se establecen los parámetros que determinan las características de afectación de éste para obtener una determinada relación señal a ruido al final de la transmisión. La matriz de 1s y 0s se convierte en una matriz con números reales enteros positivos y negativos que se utilizan en la decodificación como fiabilidades de entrada. En un principio todos los bits tienen una fiabilidad de 15 y al pasar por el canal esta fiabilidad puede o no reducirse así como el signo que define el valor del bit. Este proceso se puede entender como el manejo de una matriz en 3 dimensiones.

#### 4.2.3 Decodificación

Para realizar la decodificación en C++, se programaron todos los bloques de manera similar a como se hizo en VHDL con la diferencia de que no se utiliza el cuerpo de Galois para efectuar la decodificación pues la programación en C++ permite hacer funciones simplificadas como el cálculo del síndrome mediante una simple división.

El programa incluye funciones con las que se simulan las memorias necesarias en la circuitería, así como convertidores de binario a entero por la naturaleza de programación de alto nivel.

Al finalizar la decodificación se tiene la matriz con fiabilidades de salida mayores a aquellas de entrada y bit corregidos. Conforme se establece un mayor número de iteraciones en la decodificación es posible aumentar el poder de corrección del decodificador.

Después de la decodificación el programa hace una comparación entre matrices para detectar el número de errores y así medir la TEB del sistema, mostrando los resultados para varios números de iteraciones.

#### 4.3 Análisis de resultados

En la figura 4.1 se muestran las gráficas de desempeño logradas con la validación del decodificador elemental.

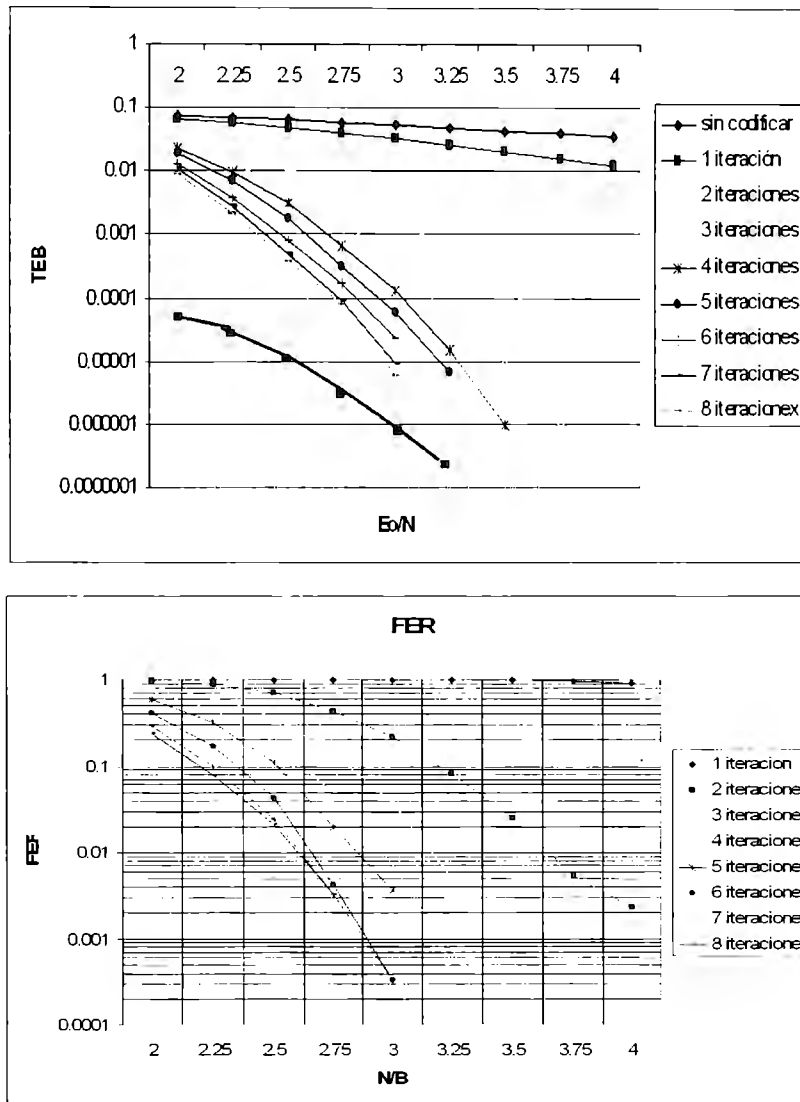


Figura 4.1

Con el programa se simuló la transmisión de datos para relaciones señal a ruido diferentes, estas simulaciones se hicieron para un canal con ruido AWGN y una modulación QPSK para 3000 matrices. Con esta gráfica se puede observar que la tasa de error binario (TEB) va decreciendo a medida que aumenta el número de iteraciones esto hace que las curvas de cada iteración caigan de manera logarítmica acercándose al límite teórico de Shannon.

Al hacer las simulaciones para decodificaciones con 4 iteraciones o más con relaciones señal a ruido mayores a 3 dB el programa daba una tasa de error binario de cero, por lo que se trunca la gráfica de estas iteraciones. Para Relaciones señal a ruido mayores a 3.75, la gráfica deja de caer de manera logarítmica y disminuye su pendiente.

A partir de 7 iteraciones las curvas son muy parecidas por lo que no es necesario hacer un circuito mayor para lograr más iteraciones ya que el desempeño no mejora de manera significativa.

#### 4.4 Conclusiones

En este capítulo se explicó la manera en la que se validó el funcionamiento del programa en VHDL con una programación de un sistema completo en C++. En resumen, el objetivo se cumplió al lograr la validación de cada una de las etapas del decodificador elemental con otro lenguaje. Los errores en los resultados debidos a redondeos que genera C++ fueron mínimos, y por consiguiente ello no alteró el funcionamiento del decodificador en C++.





---

## **III. Resultados**

---



### III.1. Diseño

La figura III.1 muestra la configuración del decodificador elemental implementado en VHDL. En las secciones anteriores se ha profundizado en el funcionamiento de este diseño.

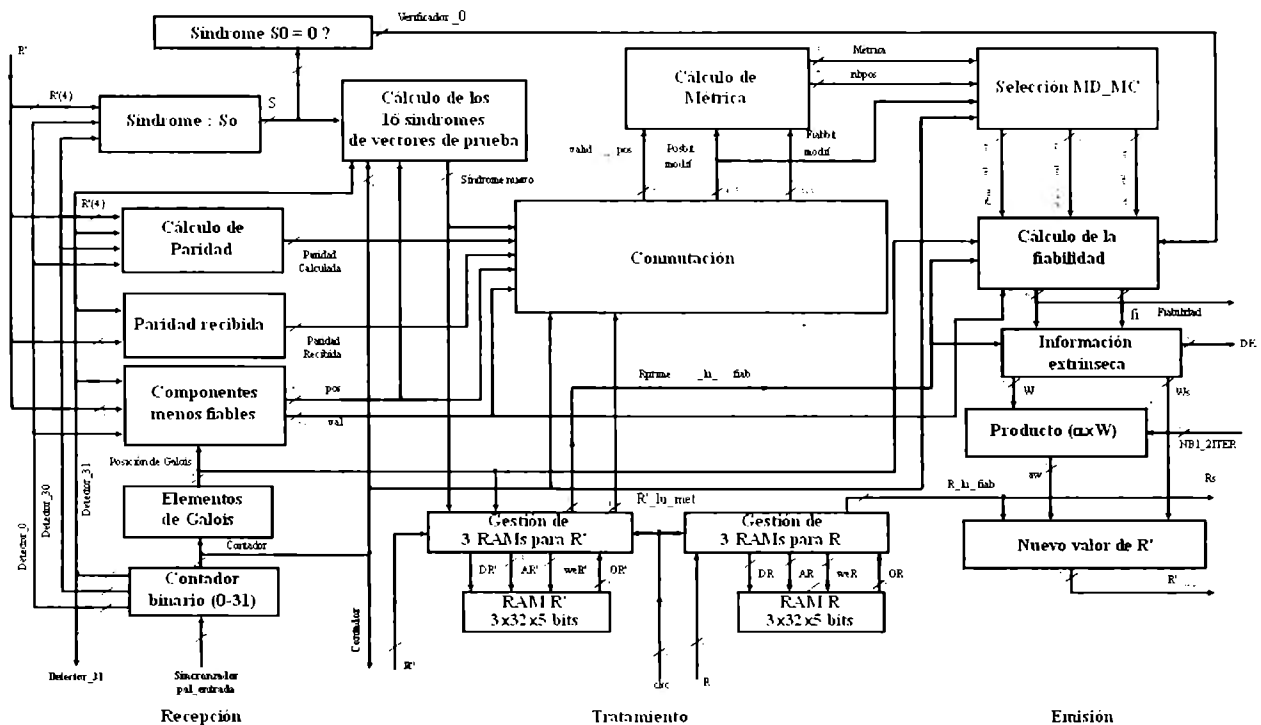


Figura III.1

### III.2. Programación

En el disco anexo se integra la programación de cada uno de los bloques del decodificador elemental en VHDL así como la programación del programa de validación en C++.

### III.3. Contribuciones al estado del arte

Con la realización de este proyecto se está contribuyendo al inicio de una línea de investigación sobre turbo decodificadores en México que actualmente sólo tiene desarrollo en países desarrollados a pesar de que su diseño poco complejo.

Se piensa que con este proyecto se pueda continuar con el diseño de nuevos decodificadores más complejos que en un futuro puedan ser comercializados una vez que se prueben en condiciones reales con un modulador y un canal no simulado que en este caso por cuestiones de tiempo no es posible completar.

Por todo lo anterior, teniendo finalizado el decodificador se crean nuevas áreas de oportunidad para tener en un futuro un dispositivo que pueda ser incorporado a cualquier sistema de transmisiones después de haberle realizado pruebas sobre distintos tipos de canal.

---

## **IV. Conclusiones**

---



Los CCE (códigos correctores de errores) constituyen una función imprescindible dentro de los nuevos sistemas de transmisión digital. Ya sea en transmisiones radioeléctricas (satelitales o móviles), sobre cable (coaxial o par trenzado) o bien sobre fibra óptica la señal observada en el receptor es perturbada por error de transmisión. La utilización de un CCE permite corregir estos errores dentro del receptor, esto hace más confiable la recepción aunque ello requiere el aumento en complejidad del receptor. La eficacia de la corrección de errores puede ser medida en términos de ganancia de codificación y el objetivo de un CCE es ofrecer el mejor compromiso entre su ganancia de codificación y su complejidad de implementación. Se han realizado progresos considerables en estos últimos años que permiten la implementación de circuitos más y más complejos, la complejidad de implementación es una gran preocupación.

El descubrimiento de los turbo códigos en 1993 permitió que se lograra una aproximación a los desempeños límite establecidos por Shannon en 1948 desde un punto de vista teórico con una complejidad razonable y de abrir nuevas perspectivas dentro del dominio de los CCE. Los turbo códigos en bloques introducidos en 1994 son una alternativa interesante en términos de desempeño para las aplicaciones que necesitan un rendimiento de codificación elevado y una tasa de error muy baja. Se ha probado que estas aplicaciones funcionan con tasas de transmisión muy elevadas (hasta 1 Gbps), lo que constituye un desafío técnico y tecnológico importante. A lo largo de este trabajo se ha estudiado una arquitectura de decodificación para altas tasas de transmisión y baja complejidad.

Los trabajos llevados a cabo en estos últimos años muestran que los turbo códigos en bloques de tipo BCH tienen muchas ventajas para las aplicaciones de alta tasa de transmisión para las cuales se busca bajas tasas de error (menor a  $10^{-7}$ ) con un rendimiento de codificación elevado (mayor a 0.8). Sin embargo, para rendimientos de codificación superiores a 0.9, los turbo códigos en bloques conducen a códigos producto de longitudes muy grandes, esto constituye un gran inconveniente para ciertas aplicaciones. En efecto, la utilización de estos códigos producto necesita de memorias de gran dimensión, lo que aumenta la complejidad e induce retardos importantes en la transmisión de información, lo que puede ser molesto para algunas aplicaciones.

Para responder a aplicaciones que requieren un rendimiento mayor a 0.9 se pueden utilizar códigos producto contruidos a partir de códigos RS (Reed-Solomon). Recientes trabajos muestran que los turbo códigos en bloques de tipo RS son capaces de funcionar a menos de un decibel del límite de Shannon con rendimiento elevador (mayor a 0.9) y para bloques de datos de tamaño inferior a 24000 bits.

Por otro lado, los turbo códigos en bloques son una alternativa aun no estandarizada, lo que ha llevado a diferentes investigaciones para comprobar su desempeño frente a los turbo códigos convolutivos. Dichas investigaciones afirman que la elección entre los turbo códigos en bloques y los convolutivos depende de la tasa de codificación del sistema. Para una tasa de error binario dada es posible definir un umbral de tasa. Este umbral define que para tasas menores a él los turbo códigos convolutivos deben ser utilizados, mientras que para tasas mayores a este umbral, la mejor opción son los turbo códigos en bloques. Ello se puede apreciar en las figuras C.1 y C.2 obtenidas de la referencia bibliográfica. Una aplicación analizada en particular son las WLANs, en donde se ha observado que ambas técnicas de codificación presentan un buen desempeño y flexibilidad. En esta aplicación los turbo códigos en bloques tienden a superar a los turbo códigos convolutivos para bajas tasas de error binario, mientras que ocurre lo contrario en la región de cascada.

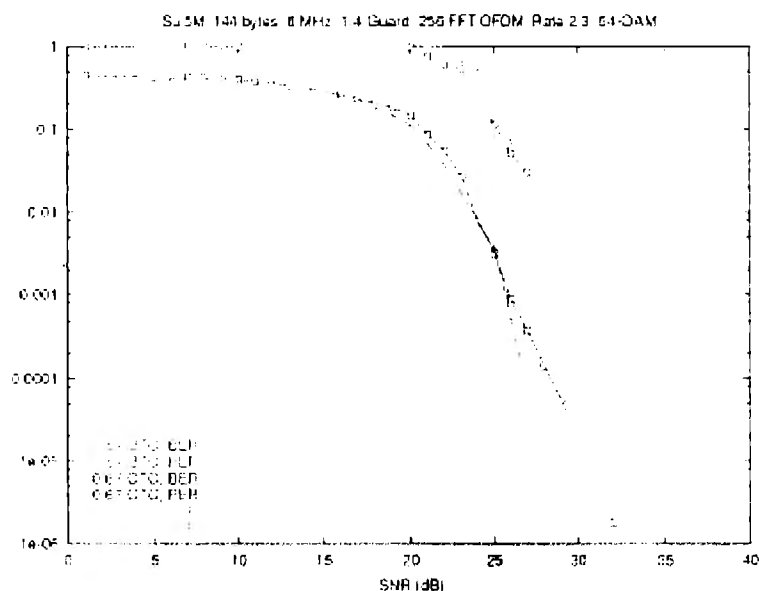
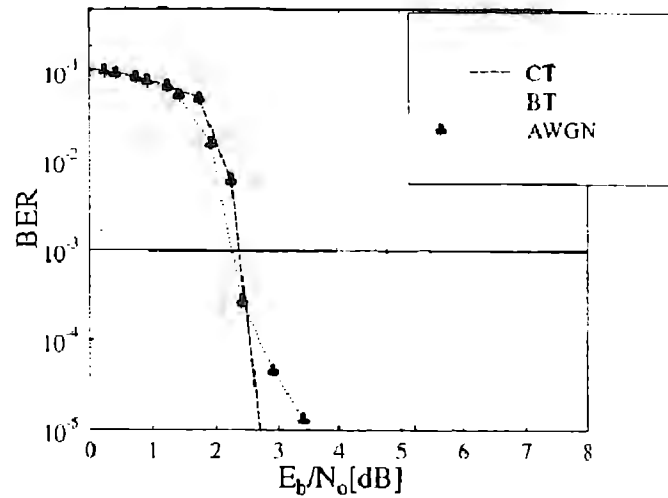


Figura C.1





(b)  $R = \frac{3}{4}$

Figura C.2

Finalmente se puede decir que al utilizar bloques de tamaño pequeño concatenando un código BCH con un solo bit de paridad es posible lograr que los turbo códigos en bloques sean compatibles con estándares que utilizan el mismo tamaño de bloque. Por todo lo anterior, es importante definir bien la tasa de error binario deseada en una cierta aplicación para hacer una elección apropiada de la técnica de codificación más adecuada.

## IV.1. Perspectivas y trabajo a futuro

Una vez finalizada la programación del decodificador elemental es posible la conexión de este en cascada con la finalidad de poder decodificar no sólo un reglón o columna sino toda una matriz. Una vez lograda la conexión en cascada se puede implementar el circuito en una tarjeta FPGA sabiendo las características de desempeño con que cuenta esta técnica de decodificación. Se espera que la implementación de este circuito sea un proyecto complementario a éste donde finalmente se pueda ir montando todo el decodificador de manera física para continuar con el desarrollo de esta técnica de decodificación en México.



---

## **V. Actividades realizadas**

---



30 de Agosto del 2005

- Visualización del proyecto.
- Se fijan horarios de trabajo con el equipo y asesor.

31 de Agosto del 2005

- Investigación sobre el sistema de comunicación digital.
- Junta con asesor. Explicación detallada de la cadena de comunicación y cuerpo de Galois.

1 de Septiembre del 2005

- Elaboración del capítulo 1 “Elementos de un sistema de decodificación digital”.

2 de Septiembre del 2005

- Elaboración del reporte preliminar
  - o Descripción
  - o Integrantes
  - o Asesor
  - o Beneficios
  - o Metas
  - o Objetivos

5 de Septiembre del 2005-10-25

- Revisión con el asesor del capítulo 1.
- Entrega del material para investigación sobre algoritmos de decodificación.
- Entrega de los códigos en C para simulación y comparaciones.

6 de Septiembre del 2005

- Calendarización de actividades para el semestre Agosto-Diciembre 2005.
- Corrección y finalización de anteproyecto.

7 de Septiembre del 2005

- Explicación de dos bloques de recepción (Contador y tabla de conversión de Galois).
- Investigación sobre propiedades de los códigos BCH.

8 de Septiembre del 2005

- Elaboración de la presentación sobre anteproyecto.
- Investigación sobre el síndrome y sus componentes.
- Solicitud del FPGA al proveedor.

13 de Septiembre del 2005

- Redacción de capítulo 2 sobre códigos en bloques, códigos producto y decodificación iterativa.
- Programación de contador y del bloque de la tabla de conversión.

14 de Septiembre del 2005

- Revisión de los primeros dos bloques de recepción.
- Explicación de los bloques de paridad recibida y paridad calculada.

Compromisos

- Terminar el capítulo 2 y revisarlo.
- Comenzar simulación de los bloques de recepción.

27 de Septiembre del 2005

- Elaboración de reporte.
- Se cumplieron con los compromisos presentados la semana anterior.

Compromisos

- Elaboración de presentación.
- Revisión por parte del asesor del avance.
- Programación de los bloques de paridad, síndrome y componentes menos fiables.

4 de Octubre del 2005

- Se cumplieron con los compromisos anteriores realizando el avance con todo lo incluido hasta el momento.

Compromisos

- Investigar sobre la arquitectura de codificadores elementales.
- Programación de bloques de tratamiento.
- Revisión de códigos en C para comparación de resultados.
- Comenzar a redactar el capítulo sobre la arquitectura de fiabilidades elementales.

18 de Octubre del 2005

- Se cumplió con todos los compromisos de las semanas anteriores y nos reunimos con nuestro asesor para que explicara el bloque de conmutación.

Compromisos

- Finalizar la programación de los bloques de tratamiento.
- Revisión de avances y presentación.
- Terminar de redactar el capítulo sobre la arquitectura de decodificadores elementales.

26 de Enero del 2006

- Redefinición de objetivos
- Plan de actividades para el segundo período
- Revisión del trabajo realizado

2 de Febrero del 2006

- Explicación del programa de verificación de C++

5 de Febrero del 2006

- Programación de bloques de recepción de C++
- Redacción de primer avance

Propósitos

- Programación de bloques de tratamiento de C++

18 de Febrero del 2006

- Se programaron los bloques de tratamiento y emisión

Propósitos

- Modificaciones para una sola iteración

25 de Febrero del 2006

- Configuración y programación del programa en C++ para varias iteraciones

Propósitos

- Verificación de funcionamiento con VHDL

28 de Febrero del 2006

- Revisión de avances

10 de Marzo del 2006

- Realización de curvas de tasa de error binario

17 de Marzo del 2006

- Realización de gráficas de desempeño para diferentes tipos de transmisiones

Propósitos

- Graficas de desempeño para transmisiones mayores

24 de Marzo del 2006

- Graficas para tasas mayores de información

Propósitos

- Fuentes para comparación con turbo convolutivos

6 de Abril del 2006

- obtención de graficas comparativas y elaboración de las conclusiones

15 de Abril del 2006

- Diseño del póster y elaboración de la presentación a puerta cerrada

25 de Abril del 2006

- Impresión del póster

29 de Abril del 2006

- Preparación de la exposición a puertas abiertas



---

## **VI. Referencias**

---



- *Turbo Decoding of product codes for Gigabit per second applications and beyond*, J. Cuevas, P. Adde, S.Kerouedan, GET – ENST Bretagne, France.
- *Turbo Décodeur de code produit BCH (32, 26, 4) x BCH (32, 26, 4) 7,5 itérations – 1 Décodeur Elémentaire*, P. Adde, S.Kerouedan, año 2000.
- *Communicating Systems Engineering* by John G. Proakis and Masoud Salehi, Prentice Hall PTR
- Schlegel, Christian B. *Trellis and turbo coding*. Piscataway, NJ, 2004, 386 pp.
- Lee, Charles. *Error-control block codes for communications engineers*. Artech House, Boston, 2000. 243 pp.
- C. Berrou, A. Glavieux, *Near Shannon limit error correcting coding and decoding: Turbo-codes*, Genova, Suiza, Mayo 1993
- B. Sklar, *Digital Communications Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1988
- J. Hagenauer, E. Offer. *Iterative decoding of binary block and convolutional codes*, Marzo 1996.
- R. Lucas, M. Bossert and M. Breitbart, *On iterative soft-decision decoding of linear binary block codes and product codes*, IEEE J. Sel. Areas Commun., vol. 16, no. 2, pp. 276-296, Feb. 1998.
- R. M. Pyndiah, *Near-optimum decoding of product codes: Block turbo codes*, IEEE Trans Commun., vol. 46, no. 8, pp. 1003-1010, Agosto 1998.
- B. Sklar, *A primer on turbo code concepts*, IEEE Communications Magazine, pp. 94-102, Diciembre 1997.



---

## **Anexos**

---



- Código del programa en VHDL

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

ENTITY recepcion IS
    PORT(

        -----
        -- ENTRADAS--
        -----

        -- contador
        CLK: in STD_LOGIC;
        RESET: in STD_LOGIC;

        -- genera R
        R: inout STD_LOGIC_VECTOR(4 downto 0);
        -- genera Ro
        Ro: inout STD_LOGIC_VECTOR(4 downto 0);

        -----
        -- IN OUT--
        -----

        -- contador
        FLAG00: inout STD_LOGIC;           --pasa por 0
        FLAG30: inout STD_LOGIC;           --pasa por 30
        FLAG31: inout STD_LOGIC;           --pasa por 31
        COUNT: inout STD_LOGIC_VECTOR (4 downto 0);
        RST: inout STD_LOGIC;               --bandera cuando reset y 0

        -- galois
        GALOIS: inout STD_LOGIC_VECTOR (4 downto 0);

        -- SINDROME S0
        SN_OUT: inout STD_LOGIC_VECTOR (4 downto 0);

        -- componentes menos fiables
        P0P_OUT: inout STD_LOGIC_VECTOR(4 downto 0);
        P1P_OUT: inout STD_LOGIC_VECTOR(4 downto 0);
        P2P_OUT: inout STD_LOGIC_VECTOR(4 downto 0);
        P3P_OUT: inout STD_LOGIC_VECTOR(4 downto 0);
        P4P_OUT: inout STD_LOGIC_VECTOR(4 downto 0);
        P0F_OUT: inout STD_LOGIC_VECTOR(3 downto 0);
        P1F_OUT: inout STD_LOGIC_VECTOR(3 downto 0);
        P2F_OUT: inout STD_LOGIC_VECTOR(3 downto 0);
        P3F_OUT: inout STD_LOGIC_VECTOR(3 downto 0);
        P4F_OUT: inout STD_LOGIC_VECTOR(3 downto 0);

        -- paridad recibida
        p_rec: inout STD_LOGIC_VECTOR (4 downto 0);

        -- PARIDAD CALCULADA
        PCALC: inout STD_LOGIC;

        --          INOUT  MEMORIAS Y GESTIONADOR R prima          --
        DO1: inout STD_LOGIC_VECTOR (4 downto 0);           -- salidas de las memorias
        DO2: inout STD_LOGIC_VECTOR (4 downto 0);
        DO3: inout STD_LOGIC_VECTOR (4 downto 0);
        RW1: inout STD_LOGIC;                                --

        entradas de las memorias

        RW2: inout STD_LOGIC;
    
```

```

RW3: inout STD_LOGIC;
SYNDROME) ADR1: inout STD_LOGIC_VECTOR (4 downto 0); -- direccion (recepcion y emisión- GALOIS; tratamiento-

ADR2: inout STD_LOGIC_VECTOR (4 downto 0);
ADR3: inout STD_LOGIC_VECTOR (4 downto 0);
DI: inout STD_LOGIC_VECTOR (4 downto 0); -- data in (R ó R')
R_TRA: inout STD_LOGIC_VECTOR (3 downto 0); -- salidas del gestorador

R_EMI: inout STD_LOGIC_VECTOR (4 downto 0);

-- INOUT MEMORIAS Y GESTIONADOR R cero
DO4: inout STD_LOGIC_VECTOR (4 downto 0);
DO5: inout STD_LOGIC_VECTOR (4 downto 0);
DO6: inout STD_LOGIC_VECTOR (4 downto 0);
ADRo: inout STD_LOGIC_VECTOR (4 downto 0); -- direccion (recepcion y emisión- GALOIS)
DIo: inout STD_LOGIC_VECTOR (4 downto 0); -- data in (Ro)

-- DELAY SINDROME S0
DELAY: inout STD_LOGIC;

-- VECTORES DE PRUEBA
SYNDROME: inout STD_LOGIC_VECTOR (4 downto 0);

-- CONMUTACION
REGISTRO: inout STD_LOGIC_VECTOR(4 downto 0);
POS_N0: inout STD_LOGIC_VECTOR(4 downto 0); -- Posiciones nuevas:
POS_N1: inout STD_LOGIC_VECTOR(4 downto 0);
POS_N2: inout STD_LOGIC_VECTOR(4 downto 0);
POS_N3: inout STD_LOGIC_VECTOR(4 downto 0);
POS_N4: inout STD_LOGIC_VECTOR(4 downto 0);
FIA_N0: inout STD_LOGIC_VECTOR(3 downto 0); -- Fiabilidades nuevas:
FIA_N1: inout STD_LOGIC_VECTOR(3 downto 0);
FIA_N2: inout STD_LOGIC_VECTOR(3 downto 0);
FIA_N3: inout STD_LOGIC_VECTOR(3 downto 0);
FIA_N4: inout STD_LOGIC_VECTOR(3 downto 0);

-- DELAY MENOS FIABLES
P0F: inout STD_LOGIC_VECTOR(3 downto 0);
P1F: inout STD_LOGIC_VECTOR(3 downto 0);
P2F: inout STD_LOGIC_VECTOR(3 downto 0);
P3F: inout STD_LOGIC_VECTOR(3 downto 0);
P4F: inout STD_LOGIC_VECTOR(3 downto 0);

-- CÁLCULO DE LA MÉTRICA
SUMA: inout STD_LOGIC_VECTOR(6 downto 0);
REGISTRO2: inout STD_LOGIC_VECTOR (4 downto 0);

-- CONCURRENCIA
MET0: inout STD_LOGIC_VECTOR(5 downto 0); -- Palabra designada
REG0: inout STD_LOGIC_VECTOR(4 downto 0);
POS00: inout STD_LOGIC_VECTOR(4 downto 0);
POS01: inout STD_LOGIC_VECTOR(4 downto 0);
POS02: inout STD_LOGIC_VECTOR(4 downto 0);
POS03: inout STD_LOGIC_VECTOR(4 downto 0);
MET1: inout STD_LOGIC_VECTOR(5 downto 0); -- Palabras concurrente 1
REG1: inout STD_LOGIC_VECTOR(4 downto 0);
POS10: inout STD_LOGIC_VECTOR(4 downto 0);
POS11: inout STD_LOGIC_VECTOR(4 downto 0);
POS12: inout STD_LOGIC_VECTOR(4 downto 0);
POS13: inout STD_LOGIC_VECTOR(4 downto 0);
MET2: inout STD_LOGIC_VECTOR(5 downto 0); -- Palabras concurrente 2
REG2: inout STD_LOGIC_VECTOR(4 downto 0);
POS20: inout STD_LOGIC_VECTOR(4 downto 0);
POS21: inout STD_LOGIC_VECTOR(4 downto 0);
POS22: inout STD_LOGIC_VECTOR(4 downto 0);
POS23: inout STD_LOGIC_VECTOR(4 downto 0);

```



## Turbo decodificador de códigos producto

---

```
MET3: inout STD_LOGIC_VECTOR(5 downto 0);           -- Palabras concurrente 3
REG3: inout STD_LOGIC_VECTOR(4 downto 0);
POS30: inout STD_LOGIC_VECTOR(4 downto 0);
POS31: inout STD_LOGIC_VECTOR(4 downto 0);
POS32: inout STD_LOGIC_VECTOR(4 downto 0);
POS33: inout STD_LOGIC_VECTOR(4 downto 0);

-- CALCULO DE FIABILIDAD

INVER:inout STD_LOGIC;
FIAB:   inout STD_LOGIC_VECTOR(6 downto 0);

-- INFORMACIÓN EXTRÍNSECA
IE_W: inout STD_LOGIC_VECTOR (6 downto 0 );
SIGNO_W: inout STD_LOGIC;
D_BIN: inout STD_LOGIC;

-- PRODUCTO
ITER: inout STD_LOGIC_VECTOR (3 downto 0);
AW: inout STD_LOGIC_VECTOR (3 downto 0);

-- NUEVO VALOR DE R
RPP: out STD_LOGIC_VECTOR (4 downto 0);

-----
-- SALIDAS --
-----

-- GESTIÓN DE MEMORIA RO
Ro_FIA:inout STD_LOGIC_VECTOR (4 downto 0)           -- Ro de emisión

);
END recepcion;
```

ARCHITECTURE recepcion\_arch OF recepcion IS

```
-----
-- SEÑALES --
-----

-- COMPONENTES MENOS FIABLES:pila intermedio
signal PILA0P: STD_LOGIC_VECTOR(4 downto 0);
signal PILA1P: STD_LOGIC_VECTOR(4 downto 0);
signal PILA2P: STD_LOGIC_VECTOR(4 downto 0);
signal PILA3P: STD_LOGIC_VECTOR(4 downto 0);
signal PILA4P: STD_LOGIC_VECTOR(4 downto 0);
signal PILA0F: STD_LOGIC_VECTOR(3 downto 0);
signal PILA1F: STD_LOGIC_VECTOR(3 downto 0);
signal PILA2F: STD_LOGIC_VECTOR(3 downto 0);
signal PILA3F: STD_LOGIC_VECTOR(3 downto 0);
signal PILA4F: STD_LOGIC_VECTOR(3 downto 0);

-- PARIDAD CALCULADA
signal P_CALC: STD_LOGIC;

-- SINDROME S0
signal Sn_1: STD_LOGIC_VECTOR (4 downto 0);
signal Sn: STD_LOGIC_VECTOR (4 downto 0);

-- DELAY SINDROME S0
SIGNAL VAR: STD_LOGIC;

-- GESTIONADOR DE MEMORIA
```

## Turbo decodificador de códigos producto

---

```
    signal gestion: STD_LOGIC_VECTOR (1 downto 0);

-- GESTIONADOR DE MEMORIA R CERO
    signal gestiono: STD_LOGIC_VECTOR (1 downto 0);

-- CONMUTACIÓN
    signal PAR: STD_LOGIC;
    signal SYND: STD_LOGIC;

-- INFORMACIÓN EXTRÍNSECA
--signal K: STD_LOGIC_VECTOR (5 downto 0);
    signal L: STD_LOGIC_VECTOR (6 downto 0);
    signal M: STD_LOGIC_VECTOR (6 downto 0);
    signal NEGADO: STD_LOGIC;
    signal NORMAL: STD_LOGIC;

-- PRODUCTO

-- NUEVO VALOR DE R
    signal V: STD_LOGIC;
    signal K: STD_LOGIC_VECTOR (4 downto 0);
    signal LL: STD_LOGIC_VECTOR (4 downto 0);
    signal SUMAWR: STD_LOGIC_VECTOR (4 downto 0);
    signal WMENOSR: STD_LOGIC_VECTOR (3 downto 0);
    signal RMENOSW: STD_LOGIC_VECTOR (3 downto 0);
    signal signo: STD_LOGIC;

-- CÁLCULO DE PALABRAS DESIGNADA Y CONCURRENTES
-- Palabra designada
-- .
-- SIGNAL SMET0: STD_LOGIC_VECTOR(5 downto 0);
-- SIGNAL SREG0: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS00: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS01: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS02: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS03: STD_LOGIC_VECTOR(4 downto 0);
--
-- -- Palabra concurrente 1
-- SIGNAL SMET1: STD_LOGIC_VECTOR(5 downto 0);
-- SIGNAL SREG1: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS10: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS11: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS12: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS13: STD_LOGIC_VECTOR(4 downto 0);
--
-- -- Palabra concurrente 2
-- SIGNAL SMET2: STD_LOGIC_VECTOR(5 downto 0);
-- SIGNAL SREG2: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS20: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS21: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS22: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS23: STD_LOGIC_VECTOR(4 downto 0);
--
-- -- Palabra concurrente 3
-- SIGNAL SMET3: STD_LOGIC_VECTOR(5 downto 0);
-- SIGNAL SREG3: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS30: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS31: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS32: STD_LOGIC_VECTOR(4 downto 0);
-- SIGNAL SPOS33: STD_LOGIC_VECTOR(4 downto 0);
--
--
--
begin
```

-----  
 ---ENTRADA (R)---  
 -----

process (COUNT)  
 begin

```

case COUNT is
  when "00000" => R(4 downto 0) <= "00100";
  when "00001" => R(4 downto 0) <= "00010";
    when "00010" => R(4 downto 0) <= "01100";
    when "00011" => R(4 downto 0) <= "10001";
    when "00100" => R(4 downto 0) <= "00001";
    when "00101" => R(4 downto 0) <= "01110";
    when "00110" => R(4 downto 0) <= "11011";
    when "00111" => R(4 downto 0) <= "01000";
    when "01000" => R(4 downto 0) <= "11110";
    when "01001" => R(4 downto 0) <= "11110";
    when "01010" => R(4 downto 0) <= "11110";
    when "01011" => R(4 downto 0) <= "10010";
    when "01100" => R(4 downto 0) <= "11111";
    when "01101" => R(4 downto 0) <= "11111";
    when "01110" => R(4 downto 0) <= "01111";
    when "01111" => R(4 downto 0) <= "01100";
  when "10000" => R(4 downto 0) <= "11111";
  when "10001" => R(4 downto 0) <= "11111";
    when "10010" => R(4 downto 0) <= "00100";
    when "10011" => R(4 downto 0) <= "01101";
    when "10100" => R(4 downto 0) <= "10100";
    when "10101" => R(4 downto 0) <= "10110";
    when "10110" => R(4 downto 0) <= "10101";
    when "10111" => R(4 downto 0) <= "00111";
    when "11000" => R(4 downto 0) <= "11111";
    when "11001" => R(4 downto 0) <= "01111";
    when "11010" => R(4 downto 0) <= "01000";
    when "11011" => R(4 downto 0) <= "01000";
    when "11100" => R(4 downto 0) <= "11111";
    when "11101" => R(4 downto 0) <= "01011";
    when "11110" => R(4 downto 0) <= "01010";
    when "11111" => R(4 downto 0) <= "01010";
  when others => R(4 downto 0) <= "00000";
end case;

```

case COUNT is

```

  when "00000" => Ro(4 downto 0) <= "00100";
  when "00001" => Ro(4 downto 0) <= "00010";
    when "00010" => Ro(4 downto 0) <= "01100";
    when "00011" => Ro(4 downto 0) <= "10001";
    when "00100" => Ro(4 downto 0) <= "00001";
    when "00101" => Ro(4 downto 0) <= "01110";
    when "00110" => Ro(4 downto 0) <= "11011";
    when "00111" => Ro(4 downto 0) <= "01000";
    when "01000" => Ro(4 downto 0) <= "11110";
    when "01001" => Ro(4 downto 0) <= "11110";
    when "01010" => Ro(4 downto 0) <= "11110";
    when "01011" => Ro(4 downto 0) <= "10010";
    when "01100" => Ro(4 downto 0) <= "11111";
    when "01101" => Ro(4 downto 0) <= "11111";
    when "01110" => Ro(4 downto 0) <= "01111";
    when "01111" => Ro(4 downto 0) <= "01100";
  when "10000" => Ro(4 downto 0) <= "11111";
  when "10001" => Ro(4 downto 0) <= "11111";
    when "10010" => Ro(4 downto 0) <= "00100";
    when "10011" => Ro(4 downto 0) <= "01101";
    when "10100" => Ro(4 downto 0) <= "10100";
    when "10101" => Ro(4 downto 0) <= "10110";
    when "10110" => Ro(4 downto 0) <= "10101";

```

```

        when "1011" => Ro(4 downto 0) <= "00111";
        when "11000" => Ro(4 downto 0) <= "11111";
        when "11001" => Ro(4 downto 0) <= "01111";
        when "11010" => Ro(4 downto 0) <= "01000";
        when "11011" => Ro(4 downto 0) <= "01000";
        when "11100" => Ro(4 downto 0) <= "11111";
        when "11101" => Ro(4 downto 0) <= "01011";
        when "11110" => Ro(4 downto 0) <= "01010";
        when "11111" => Ro(4 downto 0) <= "01010";
        when others => Ro(4 downto 0) <= "00000";

    end case;
end process;

-----
---CONTADOR, RESET Y BANDERAS---
-----

process (CLK)
variable tcont:std_logic;
begin
    if RESET='1' then
        tcont:='1';
        RST<='0';
    end if;
    if RESET='0' then
        RST<='0';
        FLAG00 <= '0';
        FLAG30 <= '0';
        FLAG31 <= '0';
        if CLK='1' and CLK'event and COUNT="11111" then
            FLAG00 <= '1';
            if tcont='1' then
                rst<='1';
            end if;
        elsif CLK='1' and CLK'event and COUNT="11101" then
            FLAG30 <= '1';
            RST<='0';
        elsif CLK='1' and CLK'event and COUNT="11110" then
            FLAG31 <= '1';
            RST<='0';
        elsif CLK='0' and CLK'event and COUNT="00000" then
            FLAG00 <= '1';
            if tcont='1' then
                rst<='1';
                tcont:='0';
            end if;
        elsif CLK='0' and CLK'event and COUNT="11110" then
            FLAG30 <= '1';
            RST<='0';
        elsif CLK='0' and CLK'event and COUNT="11111" then
            FLAG31 <= '1';
            RST<='0';
        end if;
        if CLK='1' and CLK'event then
            COUNT <= COUNT + 1;
        end if;
    else
        COUNT <= "11111";
        FLAG31 <= '1';
        FLAG00<='0';
        FLAG30<='0';
        RST<='0';
    end if;
end process;

```

```

-----
--- CONVERTIDOR A GALOIS---
-----
process (COUNT)
begin
  case COUNT is
    when "0000" => GALOIS(4 downto 0) <= "10010";
    when "00001" => GALOIS(4 downto 0) <= "01001";
    when "00010" => GALOIS(4 downto 0) <= "10110";
    when "00011" => GALOIS(4 downto 0) <= "01011";
    when "00100" => GALOIS(4 downto 0) <= "10111";
    when "00101" => GALOIS(4 downto 0) <= "11001";
    when "00110" => GALOIS(4 downto 0) <= "11110";
    when "00111" => GALOIS(4 downto 0) <= "01111";
    when "01000" => GALOIS(4 downto 0) <= "10101";
    when "01001" => GALOIS(4 downto 0) <= "11000";
    when "01010" => GALOIS(4 downto 0) <= "01100";
    when "01011" => GALOIS(4 downto 0) <= "00110";
    when "01100" => GALOIS(4 downto 0) <= "00011";
    when "01101" => GALOIS(4 downto 0) <= "10011";
    when "01110" => GALOIS(4 downto 0) <= "11011";
    when "01111" => GALOIS(4 downto 0) <= "11111";

    when "10000" => GALOIS(4 downto 0) <= "11101";
    when "10001" => GALOIS(4 downto 0) <= "11100";
    when "10010" => GALOIS(4 downto 0) <= "01110";
    when "10011" => GALOIS(4 downto 0) <= "00111";
    when "10100" => GALOIS(4 downto 0) <= "10001";
    when "10101" => GALOIS(4 downto 0) <= "11010";
    when "10110" => GALOIS(4 downto 0) <= "01101";
    when "10111" => GALOIS(4 downto 0) <= "10100";
    when "11000" => GALOIS(4 downto 0) <= "01010";
    when "11001" => GALOIS(4 downto 0) <= "00101";
    when "11010" => GALOIS(4 downto 0) <= "10000";
    when "11011" => GALOIS(4 downto 0) <= "01000";
    when "11100" => GALOIS(4 downto 0) <= "00100";
    when "11101" => GALOIS(4 downto 0) <= "00010";
    when "11110" => GALOIS(4 downto 0) <= "00001";
    when "11111" => GALOIS(4 downto 0) <= "00000";
    when others => GALOIS(4 downto 0) <= "00000";

  end case;
end process;

```

```

-----
---COMPONENTES MENOS FIABLES---
-- Funcionalidad: Calcula la fiabilidad. ---
-----
process (RESET, CLK, FLAG00, GALOIS, R, FLAG31)

```

```

  VARIABLE TP0P_OUT: STD_LOGIC_VECTOR(4 downto 0);
  VARIABLE TP1P_OUT: STD_LOGIC_VECTOR(4 downto 0);
  VARIABLE TP2P_OUT: STD_LOGIC_VECTOR(4 downto 0);
  VARIABLE TP3P_OUT: STD_LOGIC_VECTOR(4 downto 0);
  VARIABLE TP4P_OUT: STD_LOGIC_VECTOR(4 downto 0);
  VARIABLE TP0F_OUT: STD_LOGIC_VECTOR(3 downto 0);
  VARIABLE TP1F_OUT: STD_LOGIC_VECTOR(3 downto 0);
  VARIABLE TP2F_OUT: STD_LOGIC_VECTOR(3 downto 0);
  VARIABLE TP3F_OUT: STD_LOGIC_VECTOR(3 downto 0);
  VARIABLE TP4F_OUT: STD_LOGIC_VECTOR(3 downto 0);

begin
  if FLAG00='1' then

    POP_OUT<=TP0P_OUT;
    PIP_OUT<=TP1P_OUT;

```

```
P2P_OUT<=TP2P_OUT;
P3P_OUT<=TP3P_OUT;
P4P_OUT<=TP4P_OUT;
P0F_OUT<=TP0F_OUT;
P1F_OUT<=TP1F_OUT;
P2F_OUT<=TP2F_OUT;
P3F_OUT<=TP3F_OUT;
P4F_OUT<=TP4F_OUT;
end if;

if FLAG31='0' then
  if R(3 downto 0)<=PILA0F and CLK'event and CLK='1' then
    PILA4F<=PILA3F;
    PILA4P<=PILA3P;
    PILA3F<=PILA2F;
    PILA3P<=PILA2P;
    PILA2F<=PILA1F;
    PILA2P<=PILA1P;
    PILA1F<=PILA0F;
    PILA0F<=R(3 downto 0);
    PILA0P<=GALOIS;
  elsif R(3 downto 0)<=PILA1F and CLK'event and CLK='1' then
    PILA4F<=PILA3F;
    PILA4P<=PILA3P;
    PILA3F<=PILA2F;
    PILA3P<=PILA2P;
    PILA2F<=PILA1F;
    PILA2P<=PILA1P;
    PILA1F<=R(3 downto 0);
    PILA1P<=GALOIS;
  elsif R(3 downto 0)<=PILA2F and CLK'event and CLK='1' then
    PILA4F<=PILA3F;
    PILA4P<=PILA3P;
    PILA3F<=PILA2F;
    PILA3P<=PILA2P;
    PILA2F<=R(3 downto 0);
    PILA2P<=GALOIS;
  elsif R(3 downto 0)<=PILA3F and CLK'event and CLK='1' then
    PILA4F<=PILA3F;
    PILA4P<=PILA3P;
    PILA3F<=R(3 downto 0);
    PILA3P<=GALOIS;
  elsif R(3 downto 0)<=PILA4F and CLK'event and CLK='1' then
    PILA4F<=R(3 downto 0);
    PILA4P<=GALOIS;
  end if;
elsif CLK'event and CLK='1' and FLAG31='1' then
  TP0P_OUT:=PILA0P;
  TP1P_OUT:=PILA1P;
  TP2P_OUT:=PILA2P;
  TP3P_OUT:=PILA3P;
  TP4P_OUT:=PILA4P;
  TP0F_OUT:=PILA0F;
  TP1F_OUT:=PILA1F;
  TP2F_OUT:=PILA2F;
  TP3F_OUT:=PILA3F;
  TP4F_OUT:=PILA4F;

  PILA0P<="00000";
  PILA1P<="00000";
  PILA2P<="00000";
  PILA3P<="00000";
  PILA4P<="00000";
  PILA0F<="11111";
  PILA1F<="11111";
```

```

        PILA2F<="1111";
        PILA3F<="1111";
        PILA4F<="1111";

    end if;

end process;

-----
--PARIDAD RECIBIDA      --
--Objetivo: Atrapa R en bandera de 31 --
-----

process (R, FLAG31)
    variable vp_rec: STD_LOGIC_VECTOR(4 downto 0);
    begin
        if FLAG31='1' then
            vp_rec:=R;
        end if;
        if FLAG00='1' then
            p_rec<=vp_rec;
        end if;
    end process;

-----
--PARIDAD CALCULADA    --
--Objetivo: Xor de los bits más significativos--
-- de las primeras 31 palabras recibidas.    --
-----

process (FLAG00, FLAG31, R, CLK, RESET)
    begin
        if RESET = '1' then
            P_CALC<='0';
            PCALC<='0';
        elsif FLAG00 = '1' and CLK'event and CLK= '1' then
            P_CALC<=R(4);
        elsif FLAG31 /= '1' and CLK'event and CLK= '1' then
            P_CALC<=R(4) xor P_CALC;
        elsif FLAG31='1' and CLK'event and CLK='1' then
            PCALC<=P_CALC;
        end if;
    end process;

-----
-- SINDROME S0        --
-- Archivo: sindrome.vhd --
-- Funcionalidad: A partir del bit más significativo --
-- de los 31 bits R de la palabra. --
-----

process (R,FLAG00,FLAG31,RESET,CLK)
    variable var_s:std_logic_vector(4 downto 0);
    begin
        if reset='1' then
            Sn<="00000";
            Sn_1<="00000";
            var_s:="00000";
        end if;

        if FLAG00='1' then
            SN_OUT<=var_s;
            Sn<="00000";
            Sn_1<="00000";
        end if;
    end process;

```

```

elsif FLAG31='0' and CLK'event then
    Sn(0)<=R(4) xor Sn_1(4);
    Sn(1)<=Sn_1(0);
    Sn(2)<=Sn_1(1) xor Sn_1(4);
    Sn(3)<=Sn_1(2);
    Sn(4)<=Sn_1(3);
    Sn_1<=Sn;
elsif FLAG31='1' and CLK'event then
    var_s:=Sn;
end if;
end process;

```

```

-----
--          Delay Sindrome S0          --
--  Proyectos de Ingeniería: Turbodecodificador  --
--  Funcionalidad: A partir del bit más significativo  --
--  de las 31 palabras R de la cadena.  --
-----

```

```

process (FLAG00,FLAG31,RESET,CLK,SN_OUT)
begin
    if RESET='1' then
        VAR<='0';
        DELAY<='0';
    elsif FLAG31='1' and clk'event and clk='0' then
        if SN_OUT="00000" then
            DELAY<=VAR;
            VAR<='1';
        else
            DELAY<=VAR;
            VAR<='0';
        end if;
    end if;
    if FLAG31='1' then
        DELAY<=VAR;
    end if;
end process;

```

```

-----
--VECTORES DE PRUEBA          --
--Objetivo: Calcula los 16 síndromes de los 16 --
--vectores de prueba.          --
--Los nuevos síndromes se calculan: vectorNdePrueba xor SO --
-----

```

```

process (CLK, RESET, SN_OUT, COUNT)
begin
    if RESET='1' then
        SYNDROME<="00000";
    else
        if COUNT(4 downto 1)="0000" then
            SYNDROME<=SN_OUT;
        elsif COUNT(4 downto 1)="0001" then
            SYNDROME<=P0P_OUT xor SN_OUT;
        elsif COUNT(4 downto 1)="0010" then
            SYNDROME<=P1P_OUT xor SN_OUT;
        elsif COUNT(4 downto 1)="0011" then
            SYNDROME<=P2P_OUT xor SN_OUT;
        elsif COUNT(4 downto 1)="0100" then
            SYNDROME<=P3P_OUT xor SN_OUT;
        elsif COUNT(4 downto 1)="0101" then
            SYNDROME<=P4P_OUT xor SN_OUT;
        elsif COUNT(4 downto 1)="0110" then
            SYNDROME<=P0P_OUT xor P1P_OUT xor SN_OUT;
        elsif COUNT(4 downto 1)="0111" then

```



```

        SYNDROME<=P0P_OUT xor P2P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1000" then
        SYNDROME<=P0P_OUT xor P3P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1001" then
        SYNDROME<=P0P_OUT xor P4P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1010" then
        SYNDROME<=P0P_OUT xor P1P_OUT xor P2P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1011" then
        SYNDROME<=P0P_OUT xor P1P_OUT xor P3P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1100" then
        SYNDROME<=P0P_OUT xor P1P_OUT xor P4P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1101" then
        SYNDROME<=P0P_OUT xor P2P_OUT xor P3P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1110" then
        SYNDROME<=P0P_OUT xor P2P_OUT xor P4P_OUT xor SN_OUT;
    elsif COUNT(4 downto 1)="1111" then
        SYNDROME<=P0P_OUT xor P3P_OUT xor P4P_OUT xor SN_OUT;
    end if;
end if;
end process;

```

```

-----
--Conmutación
--Objetivo: cambia los bits que estan mal en el
--registro por sindrome nuevo (SYNDROME)
-----

```

```

process(CLK, RESET, COUNT, SYNDROME, R_TRA, PCALC, P_REC, P0P_OUT, P1P_OUT, P2P_OUT,
P3P_OUT, P4P_OUT, P0F_OUT, P1F_OUT, P2F_OUT, P3F_OUT, P4F_OUT)

```

```

begin
    if (PCALC=P_REC(4))then
        PAR<='1';
    else
        PAR<='0';
    end if;
    -- asignaciones
    if (SYNDROME="00000") then
        SYND<='0';
    else
        SYND<='1';
    end if;

    if RESET = '1' then
        REGISTRO<="00000";
        -- Posiciones nuevas:
        POS_N0<="00000";
        POS_N1<="00000";
        POS_N2<="00000";
        POS_N3<="00000";
        POS_N4<="00000";
        -- Fiabilidades nuevas:
        FIA_N0<="0000";
        FIA_N1<="0000";
        FIA_N2<="0000";
        FIA_N3<="0000";
        FIA_N4<="0000";
        -- signals
        PAR<='0';
        SYND<='0';
    else
        -- SINDROME: [Registro(3)]
        if SYNDROME="00000" then

```

```

REGISTRO(3)<='0';
FIA_N3<="0000";
POS_N3<=SYNDROME;
else
REGISTRO(3)<='1';
FIA_N3<=R_TRA;
POS_N3<=SYNDROME;
end if;

-- PARIDAD [Registro(4)]
if COUNT(4 downto 1)="0000" or COUNT(4 downto 1)="0110" or COUNT(4
downto 1)="0111" or COUNT(4 downto 1)="1000" or COUNT(4 downto 1)="1001" then
if SYND=PAR then
FIA_N4<=P_REC(3 downto 0); -- sólo parte
REGISTRO(4)<='1';
else
FIA_N4<="0000";
REGISTRO(4)<='0';
end if;
-- Palabras impares
else
if SYND=PAR then
FIA_N4<="0000";
REGISTRO(4)<='0';
else
FIA_N4<=P_REC(3 downto 0);
REGISTRO(4)<='1';
end if;
end if;
-- VECTORES DE PRUEBA (síndromes nuevos)
if COUNT(4 downto 1)="0000" then
REGISTRO(2 downto 0)<="000";
FIA_N0<="0000";
FIA_N1<="0000";
FIA_N2<="0000";
POS_N0<="00000";
POS_N1<="00000";
POS_N2<="00000";
elsif COUNT(4 downto 1)="0001" then
REGISTRO(2 downto 0)<="001";
FIA_N0<=P0F_OUT;
FIA_N1<="0000";
FIA_N2<="0000";
POS_N0<=P0P_OUT;
POS_N1<="00000";
POS_N2<="00000";
elsif COUNT(4 downto 1)="0010" then
REGISTRO(2 downto 0)<="001";
FIA_N0<=P1F_OUT;
FIA_N1<="0000";
FIA_N2<="0000";
POS_N0<=P1P_OUT;
POS_N1<="00000";
POS_N2<="00000";
elsif COUNT(4 downto 1)="0011" then
REGISTRO(2 downto 0)<="001";
FIA_N0<=P2F_OUT;
FIA_N1<="0000";
FIA_N2<="0000";
POS_N0<=P2P_OUT;
POS_N1<="00000";
POS_N2<="00000";
elsif COUNT(4 downto 1)="0100" then
REGISTRO(2 downto 0)<="001";
FIA_N0<=P3F_OUT;

```

```

FIA_N1<="0000";
FIA_N2<="0000";
POS_N0<=P3P_OUT;
POS_N1<="00000";
POS_N2<="00000";
elsif COUNT(4 downto 1)="0101" then
    REGISTRO(2 downto 0)<="001";
    FIA_N0<=P4F_OUT;
    FIA_N1<="0000";
    FIA_N2<="0000";
    POS_N0<=P4P_OUT;
    POS_N1<="00000";
    POS_N2<="00000";
elsif COUNT(4 downto 1)="0110" then
    REGISTRO(2 downto 0)<="011";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P1F_OUT;
    FIA_N2<="0000";
    POS_N0<=POP_OUT;
    POS_N1<=P1P_OUT;
    POS_N2<="00000";
elsif COUNT(4 downto 1)="0111" then
    REGISTRO(2 downto 0)<="011";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P2F_OUT;
    FIA_N2<="0000";
    POS_N0<=P0P_OUT;
    POS_N1<=P2P_OUT;
    POS_N2<="00000";
elsif COUNT(4 downto 1)="1000" then
    REGISTRO(2 downto 0)<="011";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P3F_OUT;
    FIA_N2<="0000";
    POS_N0<=P0P_OUT;
    POS_N1<=P3P_OUT;
    POS_N2<="00000";
elsif COUNT(4 downto 1)="1001" then
    REGISTRO(2 downto 0)<="011";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P4F_OUT;
    FIA_N2<="0000";
    POS_N0<=P0P_OUT;
    POS_N1<=P4P_OUT;
    POS_N2<="00000";
elsif COUNT(4 downto 1)="1010" then
    REGISTRO(2 downto 0)<="111";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P1F_OUT;
    FIA_N2<=P2F_OUT;
    POS_N0<=P0P_OUT;
    POS_N1<=P1P_OUT;
    POS_N2<=P2P_OUT;
elsif COUNT(4 downto 1)="1011" then
    REGISTRO(2 downto 0)<="111";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P1F_OUT;
    FIA_N2<=P3F_OUT;
    POS_N0<=P0P_OUT;
    POS_N1<=P1P_OUT;
    POS_N2<=P3P_OUT;
elsif COUNT(4 downto 1)="1100" then
    REGISTRO(2 downto 0)<="111";
    FIA_N0<=P0F_OUT;
    FIA_N1<=P1F_OUT;
    FIA_N2<=P4F_OUT;

```

```

        POS_N0<=P0P_OUT;
        POS_N1<=P1P_OUT;
        POS_N2<=P4P_OUT;
    elsif COUNT(4 downto 1)="1101" then
        REGISTRO(2 downto 0)<="111";
        FIA_N0<=P0F_OUT;
        FIA_N1<=P2F_OUT;
        FIA_N2<=P3F_OUT;
        POS_N0<=P0P_OUT;
        POS_N1<=P2P_OUT;
        POS_N2<=P3P_OUT;
    elsif COUNT(4 downto 1)="1110" then
        REGISTRO(2 downto 0)<="111";
        FIA_N0<=P0F_OUT;
        FIA_N1<=P2F_OUT;
        FIA_N2<=P4F_OUT;
        POS_N0<=P0P_OUT;
        POS_N1<=P2P_OUT;
        POS_N2<=P4P_OUT;
    elsif COUNT(4 downto 1)="1111" then
        REGISTRO(2 downto 0)<="111";
        FIA_N0<=P0F_OUT;
        FIA_N1<=P3F_OUT;
        FIA_N2<=P4F_OUT;
        POS_N0<=P0P_OUT;
        POS_N1<=P3P_OUT;
        POS_N2<=P4P_OUT;
    end if;
end if;
end process;

```

```

-----
-- GESTION DE MEMORIA PARA R PRIMA --
-- Objetivo: manejar las memorias de recepci3n, tratamiento y emisi3n para R' --
--Segun la pos de GALOIS va guardando R'Y segun el sindrome nuevo (SYNDROME), --
--saca los datos de la memoria --
-----

```

```

        process (CLK, RST, R, SYNDROME, GALOIS, DO1, DO2, DO3)
        begin
            --inicializaciones
            if RST = '1' then
                gestion<="01";
                R_TRA<="0000";
                R_EMI<="00000";
                RW1<='1';
                RW2<='0';
                RW3<='0';
            end if;
            if gestion="01" then
                -- mem1 recepci3n ; mem2-emisi3n mem3-tratamiento
                RW1<='1';
                RW2<='0';
                RW3<='0';
                DI<=R;
                ADR1<=GALOIS;
                ADR2<=GALOIS;
                ADR3<=SYNDROME;
                R_TRA<=DO3(3 downto 0);
                R_EMI<=DO2;
                if FLAG31='1' and clk='0' and clk'event then
                    gestion<="10";
                end if;
            end if;
        end process;

```

```

elsif gestion="10" then
-- mem1 trata      ;      mem2-recibe      mem3-emite
        RW1<='0';
        RW2<='1';
        RW3<='0';
        D1<=R;
        ADR1<=SYNDROME;
        ADR2<=GALOIS;
        ADR3<=GALOIS;
        R_TRA<=DO1(3 downto 0);
        R_EMI<=DO3;
        if FLAG31='1' and clk'event and clk='0' then
            gestion<="11";
        end if;
elsif gestion="11" then
-- gestion=3:      m1 emite, m2 trata, m3 recibe
        RW1<='0';
        RW2<='0';
        RW3<='1';
        D1<=R;
        ADR1<=GALOIS;
        ADR2<=SYNDROME;
        ADR3<=GALOIS;
        R_TRA<=DO2(3 downto 0);
        R_EMI<=DO1;
        if FLAG31='1' and clk'event and clk='0' then
            gestion<="01";
        end if;
end if;
end process;

```

```

-----
-- Gestión de Memorias para Ro
--Objetivo: tratamiento y emisión para R según la pos de GALOIS va guardando R
--Y según el síndrome nuevo (SYNDOME), saca los datos de la memoria
-----

```

```

process (CLK, RST, Ro, GALOIS, DO4, DO5, DO6)
begin
--inicializaciones

        ADRo<=GALOIS;
        if RST='1' then
            gestiono<="01";
        end if;
        if gestiono="01" then
            D1o<=Ro;
            Ro_FIA<=DO6;
            if FLAG31='1' and clk='0' and clk'event then
                gestiono<="10";
            end if;
        elsif gestiono="10" then
            D1o<=Ro;
            Ro_FIA<=DO4;
            if FLAG31='1' and clk'event and clk='0' then
                gestiono<="11";
            end if;
        elsif gestiono="11" then
            D1o<=Ro;
            Ro_FIA<=DO5;
            if FLAG31='1' and clk'event and clk='0' then
                gestiono<="01";
            end if;
        end if;
end process;

```

```

-----
-- Delay Fiabilidad --
--Funcionalidad: Guarda los valores de la fiabilidad --
--durante 32 periodos de reloj para --
--que se puedan usar en Emisión. --
-----
process (FLAG00,FLAG31,RESET,CLK,P0F_OUT,P1F_OUT,P2F_OUT,P3F_OUT,P4F_OUT)

    VARIABLE DEL0: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL1: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL2: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL3: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL4: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL0I: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL1I: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL2I: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL3I: STD_LOGIC_VECTOR(3 downto 0);
    VARIABLE DEL4I: STD_LOGIC_VECTOR(3 downto 0);

begin
    if RESET='1' then

        DEL0I:="0000";
        DEL1I:="0000";
        DEL2I:="0000";
        DEL3I:="0000";
        DEL4I:="0000";

        DEL0:="0000";
        DEL1:="0000";
        DEL2:="0000";
        DEL3:="0000";
        DEL4:="0000";

    elsif FLAG31='1' and clk'event and clk='0' then

        DEL0:=DEL0I;
        DEL1:=DEL1I;
        DEL2:=DEL2I;
        DEL3:=DEL3I;
        DEL4:=DEL4I;

        DEL0I:=P0F_OUT;
        DEL1I:=P1F_OUT;
        DEL2I:=P2F_OUT;
        DEL3I:=P3F_OUT;
        DEL4I:=P4F_OUT;

        P0F<=DEL0;
        P1F<=DEL1;
        P2F<=DEL2;
        P3F<=DEL3;
        P4F<=DEL4;

    end if;

end process;

-----
--CÁLCULO DE MÉTRICA --
--Objetivo:Calcula la métrica y pone a cero --
--los bits del REGISTRO2 de las --
--posiciones que se repiten. --
-----

```

```

FIA_N3, FIA_N4)
process (CLK, RESET, REGISTRO, POS_N0, POS_N1, POS_N2, POS_N3, POS_N4, FIA_N0, FIA_N1, FIA_N2,
FIA_N3, FIA_N4)

VARIABLE SUM_FIA0: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE SUM_FIA1: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE SUM_FIA2: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE SUM_FIA3: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE SUM_FIA4: STD_LOGIC_VECTOR(6 downto 0);
-- Para saber si se repitió con el síndrome alguna palabra:
VARIABLE REPETICION: STD_LOGIC_VECTOR(2 downto 0);

begin
    if RST='1' then
        REGISTRO2<="00000";
        REPETICION:="000";
        SUM_FIA0:="0000000";
        SUM_FIA1:="0000000";
        SUM_FIA2:="0000000";
        SUM_FIA3:="0000000";
        SUM_FIA4:="0000000";
        SUMA<="0000000";
    end if;
    -- posicion_del_sindrome == pos0 ?
    if POS_N0=POS_N3 then
        SUM_FIA0:="0000000";
        REPETICION(0):='1';
        REGISTRO2(0)<='0';
    else
        SUM_FIA0(3 downto 0):=FIA_N0;
        SUM_FIA0(6 downto 4):="000";
        REPETICION(0):='0';
        REGISTRO2(0)<=REGISTRO(0);
    end if;
    -- posicion_del_sindrome == pos1 ?
    if POS_N1=POS_N3 then
        SUM_FIA1:="0000000";
        REPETICION(1):='1';
        REGISTRO2(1)<='0';
    else
        SUM_FIA1(3 downto 0):=FIA_N1;
        SUM_FIA1(6 downto 4):="000";
        REPETICION(1):='0';
        REGISTRO2(1)<=REGISTRO(1);
    end if;
    -- posicion_del_sindrome == pos2 ?
    if POS_N2=POS_N3 then
        SUM_FIA2:="0000000";
        REPETICION(2):='1';
        REGISTRO2(2)<='0';
    else
        SUM_FIA2(3 downto 0):=FIA_N2;
        SUM_FIA2(6 downto 4):="000";
        REPETICION(2):='0';
        REGISTRO2(2)<=REGISTRO(2);
    end if;
    -- Sumatoria y cálculo de valores de salida:
    SUM_FIA3(3 downto 0):=FIA_N3;
    SUM_FIA3(6 downto 4):="000";
    SUM_FIA4(3 downto 0):=FIA_N4;
    SUM_FIA4(6 downto 4):="000";
    SUMA<=SUM_FIA4+SUM_FIA3+SUM_FIA2+SUM_FIA1+SUM_FIA0;

    -- REGISTRO2(4 downto 3):
    REGISTRO2(3)<=REGISTRO(3);
    REGISTRO2(4)<=REGISTRO(4);

```

```

end process;

-----
--PALABRAS CONCURRENTES          --
--Objetivo: Guarda la posición, métrica y registro --
--de la palabra designada y de las 3 --
--palabras concurrentes.         --
-----

--
--
process (CLK, COUNT, FLAG00, FLAG31, RESET, SUMA, REGISTRO2, POS_N0, POS_N1, POS_N2, POS_N3,
POS_N4)
begin
--
--
--          if FLAG00='1' then
--              -- SEÑALES
--              -- Palabra decidida
--              SMET0<="111111";
--              SREG0<="00000";
--              SPOS00<="00000";
--              SPOS01<="00000";
--              SPOS02<="00000";
--              SPOS03<="00000";
--
--              -- Palabra concurrente 1
--              SMET1<="111111";
--              SREG1<="00000";
--              SPOS10<="00000";
--              SPOS11<="00000";
--              SPOS12<="00000";
--              SPOS13<="00000";
--
--              -- Palabra concurrente 2
--              SMET2<="111111";
--              SREG2<="00000";
--              SPOS20<="00000";
--              SPOS21<="00000";
--              SPOS22<="00000";
--              SPOS23<="00000";
--
--              -- Palabra concurrente 3
--              SMET3<="111111";
--              SREG3<="00000";
--              SPOS30<="00000";
--              SPOS31<="00000";
--              SPOS32<="00000";
--              SPOS33<="00000";
--          end if;
--
--          if RESET = '1' then
--          -- SALIDAS:
--              -- Palabra decidida
--              MET0<="111111";
--              REG0<="00000";
--              POS00<="00000";
--              POS01<="00000";
--              POS02<="00000";
--              POS03<="00000";
--
--              -- Palabra concurrente 1
--              MET1<="111111";
--              REG1<="00000";
--              POS10<="00000";
--              POS11<="00000";
--              POS12<="00000";
--              POS13<="00000";
--
--

```



```

--
--
-- Palabra concurrente 2
MET2<="111111";
REG2<="00000";
POS20<="00000";
POS21<="00000";
POS22<="00000";
POS23<="00000";
--
--
-- Palabra concurrente 3
MET3<="111111";
REG3<="00000";
POS30<="00000";
POS31<="00000";
POS32<="00000";
POS33<="00000";
--
-- SEÑALES
-- Palabra decidida
SMET0<="111111";
SREG0<="00000";
SPOS00<="00000";
SPOS01<="00000";
SPOS02<="00000";
SPOS03<="00000";
--
-- Palabra concurrente 1
SMET1<="111111";
SREG1<="00000";
SPOS10<="00000";
SPOS11<="00000";
SPOS12<="00000";
SPOS13<="00000";
--
-- Palabra concurrente 2
SMET2<="111111";
SREG2<="00000";
SPOS20<="00000";
SPOS21<="00000";
SPOS22<="00000";
SPOS23<="00000";
--
-- Palabra concurrente 3
SMET3<="111111";
SREG3<="00000";
SPOS30<="00000";
SPOS31<="00000";
SPOS32<="00000";
SPOS33<="00000";
--
-- else
-- Cálculo de la menor métrica en pila:
if SUMA(5 DOWNT0 0)<=SMET0 and COUNT(1)='0' then--CLK'event and CLK='1' and
COUNT(1)='0' then
--
-- Guarda en pila los valores de 2 en 3:
SMET3<=SMET2;
SREG3<=SREG2;
SPOS30<=SPOS20;
SPOS31<=SPOS21;
SPOS32<=SPOS22;
SPOS33<=SPOS23;
--
-- Guarda en pila los valores de 1 en 2:
SMET2<=SMET1;
SREG2<=SREG1;
SPOS20<=SPOS10;
SPOS21<=SPOS11;
SPOS22<=SPOS12;
SPOS23<=SPOS13;
--

```

```

--                                     -- Guarda en pila los valores de 0 en 1:
--                                     SMET1<=SMET0;
--                                     SREG1<=SREG0;
--                                     SPOS10<=SPOS00;
--                                     SPOS11<=SPOS01;
--                                     SPOS12<=SPOS02;
--                                     SPOS13<=SPOS03;
--                                     -- Nuevos valores en 0:
--                                     SMET0<=SUMA(5 DOWNT0 0);
--                                     SREG0<=REGISTRO2;
--                                     SPOS00<=POS_N0;
--                                     SPOS01<=POS_N1;
--                                     SPOS02<=POS_N2;
--                                     SPOS03<=POS_N3;
--
--                                     elsif SUMA(5 DOWNT0 0)<=SMET1 and COUNT(1)='0' then--CLK'event and CLK='1' and
COUNT(1)='0' then
--                                     -- Guarda en pila los valores de 2 en 3:
--                                     SMET3<=SMET2;
--                                     SREG3<=SREG2;
--                                     SPOS30<=SPOS20;
--                                     SPOS31<=SPOS21;
--                                     SPOS32<=SPOS22;
--                                     SPOS33<=SPOS23;
--                                     -- Guarda en pila los valores de 1 en 2:
--                                     SMET2<=SMET1;
--                                     SREG2<=SREG1;
--                                     SPOS20<=SPOS10;
--                                     SPOS21<=SPOS11;
--                                     SPOS22<=SPOS12;
--                                     SPOS23<=SPOS13;
--                                     -- Nuevos valores en 1:
--                                     SMET1<=SUMA(5 DOWNT0 0);
--                                     SREG1<=REGISTRO2;
--                                     SPOS10<=POS_N0;
--                                     SPOS11<=POS_N1;
--                                     SPOS12<=POS_N2;
--                                     SPOS13<=POS_N3;
--
--                                     elsif SUMA(5 DOWNT0 0)<=SMET2 and COUNT(1)='0' then--CLK'event and CLK='1' and
COUNT(1)='0' then
--                                     -- Guarda en pila los valores de 2 en 3:
--                                     SMET3<=SMET2;
--                                     SREG3<=SREG2;
--                                     SPOS30<=SPOS20;
--                                     SPOS31<=SPOS21;
--                                     SPOS32<=SPOS22;
--                                     SPOS33<=SPOS23;
--                                     -- Nuevos valores en 2:
--                                     SMET2<=SUMA(5 DOWNT0 0);
--                                     SREG2<=REGISTRO2;
--                                     SPOS20<=POS_N0;
--                                     SPOS21<=POS_N1;
--                                     SPOS22<=POS_N2;
--                                     SPOS23<=POS_N3;
--
--                                     elsif SUMA(5 DOWNT0 0)<=SMET3 and COUNT(1)='0' then--CLK'event and CLK='1' and
COUNT(1)='0' then
--                                     -- Nuevos valores en 2:
--                                     SMET3<=SUMA(5 DOWNT0 0);
--                                     SREG3<=REGISTRO2;
--                                     SPOS30<=POS_N0;
--                                     SPOS31<=POS_N1;
--                                     SPOS32<=POS_N2;
--                                     SPOS33<=POS_N3;
--
--

```



```
-- Palabra concurrente 3
VARIABLE SMET3: STD_LOGIC_VECTOR(5 downto 0);
VARIABLE SREG3: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE SPOS30: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE SPOS31: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE SPOS32: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE SPOS33: STD_LOGIC_VECTOR(4 downto 0);
```

----- TEMPORALES

```
-- Palabra designada
VARIABLE TMET0: STD_LOGIC_VECTOR(5 downto 0);
VARIABLE TREG0: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS00: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS01: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS02: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS03: STD_LOGIC_VECTOR(4 downto 0);

-- Palabra concurrente 1
VARIABLE TMET1: STD_LOGIC_VECTOR(5 downto 0);
VARIABLE TREG1: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS10: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS11: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS12: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS13: STD_LOGIC_VECTOR(4 downto 0);

-- Palabra concurrente 2
VARIABLE TMET2: STD_LOGIC_VECTOR(5 downto 0);
VARIABLE TREG2: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS20: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS21: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS22: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS23: STD_LOGIC_VECTOR(4 downto 0);

-- Palabra concurrente 3
VARIABLE TMET3: STD_LOGIC_VECTOR(5 downto 0);
VARIABLE TREG3: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS30: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS31: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS32: STD_LOGIC_VECTOR(4 downto 0);
VARIABLE TPOS33: STD_LOGIC_VECTOR(4 downto 0);
```

```
begin
```

```
    if RESET = '1' then
-- SALIDAS:
        -- Palabra decidida
        TMET0:="000000";
        TREG0:="00000";
        TPOS00:="000000";
        TPOS01:="000000";
        TPOS02:="000000";
        TPOS03:="000000";

        -- Palabra concurrente 1
        TMET1:="000000";
        TREG1:="00000";
        TPOS10:="000000";
        TPOS11:="000000";
        TPOS12:="000000";
        TPOS13:="000000";

        -- Palabra concurrente 2
```

```

    TMET2:="000000";
    TREG2:="000000";
    TPOS20:="00000";
    TPOS21:="000000";
    TPOS22:="000000";
    TPOS23:="000000";

    -- Palabra concurrente 3
    TMET3:="000000";
    TREG3:="000000";
    TPOS30:="000000";
    TPOS31:="000000";
    TPOS32:="000000";
    TPOS33:="000000";

-- SEÑALES

    -- Palabra decidida
    SMET0:="111111";
    SREG0:="000000";
    SPOS00:="000000";
    SPOS01:="000000";
    SPOS02:="000000";
    SPOS03:="000000";

    -- Palabra concurrente 1
    SMET1:="111111";
    SREG1:="000000";
    SPOS10:="000000";
    SPOS11:="000000";
    SPOS12:="000000";
    SPOS13:="000000";

    -- Palabra concurrente 2
    SMET2:="111111";
    SREG2:="000000";
    SPOS20:="000000";
    SPOS21:="000000";
    SPOS22:="000000";
    SPOS23:="000000";

    -- Palabra concurrente 3
    SMET3:="111111";
    SREG3:="000000";
    SPOS30:="000000";
    SPOS31:="000000";
    SPOS32:="000000";
    SPOS33:="000000";

else
    if FLAG00='1' then
-- SALIDAS:

        -- Palabra decidida
        MET0<=TMET0;
        REG0<=TREG0;
        POS00<=TPOS00;
        POS01<=TPOS01;
        POS02<=TPOS02;
        POS03<=TPOS03;

        -- Palabra concurrente 1
        MET1<=TMET1;
        REG1<=TREG1;
        POS10<=TPOS10;
        POS11<=TPOS11;
```

```
    POS12<=TPOS12;
    POS13<=TPOS13;

    -- Palabra concurrente 2
    MET2<=TMET2;
    REG2<=TREG2;
    POS20<=TPOS20;
    POS21<=TPOS21;
    POS22<=TPOS22;
    POS23<=TPOS23;

    -- Palabra concurrente 3
    MET3<=TMET3;
    REG3<=TREG3;
    POS30<=TPOS30;
    POS31<=TPOS31;
    POS32<=TPOS32;
    POS33<=TPOS33;

    -- SEÑALES

    -- Palabra decidida
    SMET0:="111111";
    SREG0:="00000";
    SPOS00:="00000";
    SPOS01:="00000";
    SPOS02:="00000";
    SPOS03:="00000";

    -- Palabra concurrente 1
    SMET1:="111111";
    SREG1:="00000";
    SPOS10:="00000";
    SPOS11:="00000";
    SPOS12:="00000";
    SPOS13:="00000";

    -- Palabra concurrente 2
    SMET2:="111111";
    SREG2:="00000";
    SPOS20:="00000";
    SPOS21:="00000";
    SPOS22:="00000";
    SPOS23:="00000";

    -- Palabra concurrente 3
    SMET3:="111111";
    SREG3:="00000";
    SPOS30:="00000";
    SPOS31:="00000";
    SPOS32:="00000";
    SPOS33:="00000";

    elsif FLAG31='1' then--and CLK='1' and CLK'event then
-- TEMPORALES:
        -- Palabra decidida
        TMET0:=SMET0;
        TREG0:=SREG0;
        TPOS00:=SPOS00;
        TPOS01:=SPOS01;
        TPOS02:=SPOS02;
        TPOS03:=SPOS03;

        -- Palabra concurrente 1
        TMET1:=SMET1;
        TREG1:=SREG1;
```

```

TPOS10:=SPOS10;
TPOS11:=SPOS11;
TPOS12:=SPOS12;
TPOS13:=SPOS13;

-- Palabra concurrente 2
TMET2:=SMET2;
TREG2:=SREG2;
TPOS20:=SPOS20;
TPOS21:=SPOS21;
TPOS22:=SPOS22;
TPOS23:=SPOS23;

-- Palabra concurrente 3
TMET3:=SMET3;
TREG3:=SREG3;
TPOS30:=SPOS30;
TPOS31:=SPOS31;
TPOS32:=SPOS32;
TPOS33:=SPOS33;

end if;
-- Cálculo de la menor métrica en pila:
if SUMA(5 downto 0)<=SMET0 and COUNT(0)='1' then-- and CLK'event and CLK='1' and
COUNT1='0' then
-- Guarda en pila los valores de 2 en 3:
SMET3:=SMET2;
SREG3:=SREG2;
SPOS30:=SPOS20;
SPOS31:=SPOS21;
SPOS32:=SPOS22;
SPOS33:=SPOS23;
-- Guarda en pila los valores de 1 en 2:
SMET2:=SMET1;
SREG2:=SREG1;
SPOS20:=SPOS10;
SPOS21:=SPOS11;
SPOS22:=SPOS12;
SPOS23:=SPOS13;
-- Guarda en pila los valores de 0 en 1:
SMET1:=SMET0;
SREG1:=SREG0;
SPOS10:=SPOS00;
SPOS11:=SPOS01;
SPOS12:=SPOS02;
SPOS13:=SPOS03;
-- Nuevos valores en 0:
SMET0:=SUMA(5 downto 0);
SREG0:=REGISTRO2;
SPOS00:=POS_N0;
SPOS01:=POS_N1;
SPOS02:=POS_N2;
SPOS03:=POS_N3;
elsif SUMA(5 downto 0)<=SMET1 and COUNT(0)='1' then--CLK'event and CLK='1' and
COUNT1='0' then
-- Guarda en pila los valores de 2 en 3:
SMET3:=SMET2;
SREG3:=SREG2;
SPOS30:=SPOS20;
SPOS31:=SPOS21;
SPOS32:=SPOS22;
SPOS33:=SPOS23;
-- Guarda en pila los valores de 1 en 2:
SMET2:=SMET1;
SREG2:=SREG1;
SPOS20:=SPOS10;

```

```

SPOS21:=SPOS11;
SPOS22:=SPOS12;
SPOS23:=SPOS13;
-- Nuevos valores en 1:
SMET1:=SUMA(5 downto 0);
SREG1:=REGISTRO2;
SPOS10:=POS_N0;
SPOS11:=POS_N1;
SPOS12:=POS_N2;
SPOS13:=POS_N3;
elsif SUMA<=SMET2 and COUNT(0)='1' then--CLK'event and CLK='1' and COUNT1='0' then
-- Guarda en pila los valores de 2 en 3:
SMET3:=SMET2;
SREG3:=SREG2;
SPOS30:=SPOS20;
SPOS31:=SPOS21;
SPOS32:=SPOS22;
SPOS33:=SPOS23;
-- Nuevos valores en 2:
SMET2:=SUMA(5 downto 0);
SREG2:=REGISTRO2;
SPOS20:=POS_N0;
SPOS21:=POS_N1;
SPOS22:=POS_N2;
SPOS23:=POS_N3;
elsif SUMA<=SMET3 and COUNT(0)='1' then--CLK'event and CLK='1' and COUNT1='0' then

-- Nuevos valores en 2:
SMET3:=SUMA(5 downto 0);
SREG3:=REGISTRO2;
SPOS30:=POS_N0;
SPOS31:=POS_N1;
SPOS32:=POS_N2;
SPOS33:=POS_N3;
end if;
--end if;
end if;
end process;

-----
--MEMORIA RAM 1--
-----

process (RW1, ADRI, DI, CLK, RESET)

variable D100: STD_LOGIC_VECTOR (4 downto 0);
variable D101: STD_LOGIC_VECTOR (4 downto 0);
variable D102: STD_LOGIC_VECTOR (4 downto 0);
variable D103: STD_LOGIC_VECTOR (4 downto 0);
variable D104: STD_LOGIC_VECTOR (4 downto 0);
variable D105: STD_LOGIC_VECTOR (4 downto 0);
variable D106: STD_LOGIC_VECTOR (4 downto 0);
variable D107: STD_LOGIC_VECTOR (4 downto 0);
variable D108: STD_LOGIC_VECTOR (4 downto 0);
variable D109: STD_LOGIC_VECTOR (4 downto 0);
variable D110: STD_LOGIC_VECTOR (4 downto 0);
variable D111: STD_LOGIC_VECTOR (4 downto 0);
variable D112: STD_LOGIC_VECTOR (4 downto 0);
variable D113: STD_LOGIC_VECTOR (4 downto 0);
variable D114: STD_LOGIC_VECTOR (4 downto 0);
variable D115: STD_LOGIC_VECTOR (4 downto 0);
variable D116: STD_LOGIC_VECTOR (4 downto 0);
variable D117: STD_LOGIC_VECTOR (4 downto 0);

```



```
variable D118: STD_LOGIC_VECTOR (4 downto 0);
variable D119: STD_LOGIC_VECTOR (4 downto 0);
variable D120: STD_LOGIC_VECTOR (4 downto 0);
variable D121: STD_LOGIC_VECTOR (4 downto 0);
variable D122: STD_LOGIC_VECTOR (4 downto 0);
variable D123: STD_LOGIC_VECTOR (4 downto 0);
variable D124: STD_LOGIC_VECTOR (4 downto 0);
variable D125: STD_LOGIC_VECTOR (4 downto 0);
variable D126: STD_LOGIC_VECTOR (4 downto 0);
variable D127: STD_LOGIC_VECTOR (4 downto 0);
variable D128: STD_LOGIC_VECTOR (4 downto 0);
variable D129: STD_LOGIC_VECTOR (4 downto 0);
variable D130: STD_LOGIC_VECTOR (4 downto 0);
variable D131: STD_LOGIC_VECTOR (4 downto 0);

begin
--reset
    if RESET = '1' then
        D100:="00000";
        D101:="00000";
        D102:="00000";
        D103:="00000";
        D104:="00000";
        D105:="00000";
        D106:="00000";
        D107:="00000";
        D108:="00000";
        D109:="00000";
        D110:="00000";
        D111:="00000";
        D112:="00000";
        D113:="00000";
        D114:="00000";
        D115:="00000";
        D116:="00000";
        D117:="00000";
        D118:="00000";
        D119:="00000";
        D120:="00000";
        D121:="00000";
        D122:="00000";
        D123:="00000";
        D124:="00000";
        D125:="00000";
        D126:="00000";
        D127:="00000";
        D128:="00000";
        D129:="00000";
        D130:="00000";
        D131:="00000";
    --read
    elsif RW1 = '0' then
        case ADR1 is
            when "10010" => DO1(4 downto 0)<= D100;
            when "01001" => DO1(4 downto 0)<= D101;
            when "10110" => DO1(4 downto 0)<= D102;
            when "01011" => DO1(4 downto 0)<= D103;
            when "10111" => DO1(4 downto 0)<= D104;
            when "11001" => DO1(4 downto 0)<= D105;
            when "11110" => DO1(4 downto 0)<= D106;
            when "01111" => DO1(4 downto 0)<= D107;
            when "10101" => DO1(4 downto 0)<= D108;
            when "11000" => DO1(4 downto 0)<= D109;
            when "01100" => DO1(4 downto 0)<= D110;
            when "00110" => DO1(4 downto 0)<= D111;
            when "00011" => DO1(4 downto 0)<= D112;
```

```
when "10011" => DO1(4 downto 0) <= D113;
when "11011" => DO1(4 downto 0) <= D114;
when "11111" => DO1(4 downto 0) <= D115;
when "11101" => DO1(4 downto 0) <= D116;
when "11100" => DO1(4 downto 0) <= D117;
when "01110" => DO1(4 downto 0) <= D118;
when "00111" => DO1(4 downto 0) <= D119;
when "10001" => DO1(4 downto 0) <= D120;
when "11010" => DO1(4 downto 0) <= D121;
when "01101" => DO1(4 downto 0) <= D122;
when "10100" => DO1(4 downto 0) <= D123;
when "01010" => DO1(4 downto 0) <= D124;
when "00101" => DO1(4 downto 0) <= D125;
when "10000" => DO1(4 downto 0) <= D126;
when "01000" => DO1(4 downto 0) <= D127;
when "00100" => DO1(4 downto 0) <= D128;
when "00010" => DO1(4 downto 0) <= D129;
when "00001" => DO1(4 downto 0) <= D130;
when "00000" => DO1(4 downto 0) <= D131;
when others => D100(4 downto 0) := D100;
end case;

--write
else

case ADRI is
when "10010" => D100(4 downto 0) := DI;
when "01001" => D101(4 downto 0) := DI;
when "10110" => D102(4 downto 0) := DI;
when "01011" => D103(4 downto 0) := DI;
when "10111" => D104(4 downto 0) := DI;
when "11001" => D105(4 downto 0) := DI;
when "11110" => D106(4 downto 0) := DI;
when "01111" => D107(4 downto 0) := DI;
when "10101" => D108(4 downto 0) := DI;
when "11000" => D109(4 downto 0) := DI;
when "01100" => D110(4 downto 0) := DI;
when "00110" => D111(4 downto 0) := DI;
when "00011" => D112(4 downto 0) := DI;
when "10011" => D113(4 downto 0) := DI;
when "11011" => D114(4 downto 0) := DI;
when "11111" => D115(4 downto 0) := DI;
when "11101" => D116(4 downto 0) := DI;
when "11100" => D117(4 downto 0) := DI;
when "01110" => D118(4 downto 0) := DI;
when "00111" => D119(4 downto 0) := DI;
when "10001" => D120(4 downto 0) := DI;
when "11010" => D121(4 downto 0) := DI;
when "01101" => D122(4 downto 0) := DI;
when "10100" => D123(4 downto 0) := DI;
when "01010" => D124(4 downto 0) := DI;
when "00101" => D125(4 downto 0) := DI;
when "10000" => D126(4 downto 0) := DI;
when "01000" => D127(4 downto 0) := DI;
when "00100" => D128(4 downto 0) := DI;
when "00010" => D129(4 downto 0) := DI;
when "00001" => D130(4 downto 0) := DI;
when "00000" => D131(4 downto 0) := DI;
when others => D100(4 downto 0) := D100;
end case;

end if;
end process;

-----
--MEMORIA RAM 2--
```

```
-----
process (RW2, ADR2, DI, CLK, RESET)
  variable D200: STD_LOGIC_VECTOR (4 downto 0);
  variable D201: STD_LOGIC_VECTOR (4 downto 0);
  variable D202: STD_LOGIC_VECTOR (4 downto 0);
  variable D203: STD_LOGIC_VECTOR (4 downto 0);
  variable D204: STD_LOGIC_VECTOR (4 downto 0);
  variable D205: STD_LOGIC_VECTOR (4 downto 0);
  variable D206: STD_LOGIC_VECTOR (4 downto 0);
  variable D207: STD_LOGIC_VECTOR (4 downto 0);
  variable D208: STD_LOGIC_VECTOR (4 downto 0);
  variable D209: STD_LOGIC_VECTOR (4 downto 0);
  variable D210: STD_LOGIC_VECTOR (4 downto 0);
  variable D211: STD_LOGIC_VECTOR (4 downto 0);
  variable D212: STD_LOGIC_VECTOR (4 downto 0);
  variable D213: STD_LOGIC_VECTOR (4 downto 0);
  variable D214: STD_LOGIC_VECTOR (4 downto 0);
  variable D215: STD_LOGIC_VECTOR (4 downto 0);
  variable D216: STD_LOGIC_VECTOR (4 downto 0);
  variable D217: STD_LOGIC_VECTOR (4 downto 0);
  variable D218: STD_LOGIC_VECTOR (4 downto 0);
  variable D219: STD_LOGIC_VECTOR (4 downto 0);
  variable D220: STD_LOGIC_VECTOR (4 downto 0);
  variable D221: STD_LOGIC_VECTOR (4 downto 0);
  variable D222: STD_LOGIC_VECTOR (4 downto 0);
  variable D223: STD_LOGIC_VECTOR (4 downto 0);
  variable D224: STD_LOGIC_VECTOR (4 downto 0);
  variable D225: STD_LOGIC_VECTOR (4 downto 0);
  variable D226: STD_LOGIC_VECTOR (4 downto 0);
  variable D227: STD_LOGIC_VECTOR (4 downto 0);
  variable D228: STD_LOGIC_VECTOR (4 downto 0);
  variable D229: STD_LOGIC_VECTOR (4 downto 0);
  variable D230: STD_LOGIC_VECTOR (4 downto 0);
  variable D231: STD_LOGIC_VECTOR (4 downto 0);
begin
  --reset
  if RESET = '1' then
    D200:="00000";
    D201:="00000";
    D202:="00000";
    D203:="00000";
    D204:="00000";
    D205:="00000";
    D206:="00000";
    D207:="00000";
    D208:="00000";
    D209:="00000";
    D210:="00000";
    D211:="00000";
    D212:="00000";
    D213:="00000";
    D214:="00000";
    D215:="00000";
    D216:="00000";
    D217:="00000";
    D218:="00000";
    D219:="00000";
    D220:="00000";
    D221:="00000";
    D222:="00000";
    D223:="00000";
    D224:="00000";
    D225:="00000";
    D226:="00000";
    D227:="00000";
```

```

D228:="00000";
D229:="00000";
D230:="00000";
D231:="00000";

--read
    elsif RW2 = '0' then
        case ADR2 is
            when "10010" => DO2(4 downto 0)<= D200;
            when "01001" => DO2(4 downto 0)<= D201;
            when "10110" => DO2(4 downto 0)<= D202;
            when "01011" => DO2(4 downto 0)<= D203;
            when "10111" => DO2(4 downto 0)<= D204;
            when "11001" => DO2(4 downto 0)<= D205;
            when "11110" => DO2(4 downto 0)<= D206;
            when "01111" => DO2(4 downto 0)<= D207;
            when "10101" => DO2(4 downto 0)<= D208;
            when "11000" => DO2(4 downto 0)<= D209;
            when "01100" => DO2(4 downto 0)<= D210;
            when "00110" => DO2(4 downto 0)<= D211;
            when "00011" => DO2(4 downto 0)<= D212;
            when "10011" => DO2(4 downto 0)<= D213;
            when "11011" => DO2(4 downto 0)<= D214;
            when "11111" => DO2(4 downto 0)<= D215;
            when "11101" => DO2(4 downto 0)<= D216;
            when "11100" => DO2(4 downto 0)<= D217;
            when "01110" => DO2(4 downto 0)<= D218;
            when "00111" => DO2(4 downto 0)<= D219;
            when "10001" => DO2(4 downto 0)<= D220;
            when "11010" => DO2(4 downto 0)<= D221;
            when "01101" => DO2(4 downto 0)<= D222;
            when "10100" => DO2(4 downto 0)<= D223;
            when "01010" => DO2(4 downto 0)<= D224;
            when "00101" => DO2(4 downto 0)<= D225;
            when "10000" => DO2(4 downto 0)<= D226;
            when "01000" => DO2(4 downto 0)<= D227;
            when "00100" => DO2(4 downto 0)<= D228;
            when "00010" => DO2(4 downto 0)<= D229;
            when "00001" => DO2(4 downto 0)<= D230;
            when "00000" => DO2(4 downto 0)<= D231;
            when others => D200(4 downto 0):= D200;
        end case;

--write
    else
        case ADR2 is
            when "10010" => D200(4 downto 0):= DI;
            when "01001" => D201(4 downto 0):= DI;
            when "10110" => D202(4 downto 0):= DI;
            when "01011" => D203(4 downto 0):= DI;
            when "10111" => D204(4 downto 0):= DI;
            when "11001" => D205(4 downto 0):= DI;
            when "11110" => D206(4 downto 0):= DI;
            when "01111" => D207(4 downto 0):= DI;
            when "10101" => D208(4 downto 0):= DI;
            when "11000" => D209(4 downto 0):= DI;
            when "01100" => D210(4 downto 0):= DI;
            when "00110" => D211(4 downto 0):= DI;
            when "00011" => D212(4 downto 0):= DI;
            when "10011" => D213(4 downto 0):= DI;
            when "11011" => D214(4 downto 0):= DI;
            when "11111" => D215(4 downto 0):= DI;
            when "11101" => D216(4 downto 0):= DI;
            when "11100" => D217(4 downto 0):= DI;
            when "01110" => D218(4 downto 0):= DI;
            when "00111" => D219(4 downto 0):= DI;
            when "10001" => D220(4 downto 0):= DI;
            when "11010" => D221(4 downto 0):= DI;
        end case;
    end if;
end process;

```

```

when "01101" => D222(4 downto 0):= DI;
when "10100" => D223(4 downto 0):= DI;
when "01010" => D224(4 downto 0):= DI;
when "00101" => D225(4 downto 0):= DI;
when "10000" => D226(4 downto 0):= DI;
when "01000" => D227(4 downto 0):= DI;
when "00100" => D228(4 downto 0):= DI;
when "00010" => D229(4 downto 0):= DI;
when "00001" => D230(4 downto 0):= DI;
when "00000" => D231(4 downto 0):= DI;
when others => D200(4 downto 0):= D200;
end case;

end if;
end process;

```

-----  
--MEMORIA RAM 3--  
-----

```

process (RW3, ADR3, DI, CLK, RESET)

variable D300: STD_LOGIC_VECTOR (4 downto 0);
variable D301: STD_LOGIC_VECTOR (4 downto 0);
variable D302: STD_LOGIC_VECTOR (4 downto 0);
variable D303: STD_LOGIC_VECTOR (4 downto 0);
variable D304: STD_LOGIC_VECTOR (4 downto 0);
variable D305: STD_LOGIC_VECTOR (4 downto 0);
variable D306: STD_LOGIC_VECTOR (4 downto 0);
variable D307: STD_LOGIC_VECTOR (4 downto 0);
variable D308: STD_LOGIC_VECTOR (4 downto 0);
variable D309: STD_LOGIC_VECTOR (4 downto 0);
variable D310: STD_LOGIC_VECTOR (4 downto 0);
variable D311: STD_LOGIC_VECTOR (4 downto 0);
variable D312: STD_LOGIC_VECTOR (4 downto 0);
variable D313: STD_LOGIC_VECTOR (4 downto 0);
variable D314: STD_LOGIC_VECTOR (4 downto 0);
variable D315: STD_LOGIC_VECTOR (4 downto 0);
variable D316: STD_LOGIC_VECTOR (4 downto 0);
variable D317: STD_LOGIC_VECTOR (4 downto 0);
variable D318: STD_LOGIC_VECTOR (4 downto 0);
variable D319: STD_LOGIC_VECTOR (4 downto 0);
variable D320: STD_LOGIC_VECTOR (4 downto 0);
variable D321: STD_LOGIC_VECTOR (4 downto 0);
variable D322: STD_LOGIC_VECTOR (4 downto 0);
variable D323: STD_LOGIC_VECTOR (4 downto 0);
variable D324: STD_LOGIC_VECTOR (4 downto 0);
variable D325: STD_LOGIC_VECTOR (4 downto 0);
variable D326: STD_LOGIC_VECTOR (4 downto 0);
variable D327: STD_LOGIC_VECTOR (4 downto 0);
variable D328: STD_LOGIC_VECTOR (4 downto 0);
variable D329: STD_LOGIC_VECTOR (4 downto 0);
variable D330: STD_LOGIC_VECTOR (4 downto 0);
variable D331: STD_LOGIC_VECTOR (4 downto 0);
begin
--reset
if RESET = '1' then
D300:="00000";
D301:="00000";
D302:="00000";
D303:="00000";
D304:="00000";
D305:="00000";
D306:="00000";
D307:="00000";
D308:="00000";

```

```

D309:="00000";
D310:="00000";
D311:="00000";
D312:="00000";
D313:="00000";
D314:="00000";
D315:="00000";
D316:="00000";
D317:="00000";
D318:="00000";
D319:="00000";
D320:="00000";
D321:="00000";
D322:="00000";
D323:="00000";
D324:="00000";
D325:="00000";
D326:="00000";
D327:="00000";
D328:="00000";
D329:="00000";
D330:="00000";
D331:="00000";

--read
    elsif RW3 = '0' then
        case ADR3 is
            when "10010" => DO3(4 downto 0)<= D300;
            when "01001" => DO3(4 downto 0)<= D301;
            when "10110" => DO3(4 downto 0)<= D302;
            when "01011" => DO3(4 downto 0)<= D303;
            when "10111" => DO3(4 downto 0)<= D304;
            when "11001" => DO3(4 downto 0)<= D305;
            when "11110" => DO3(4 downto 0)<= D306;
            when "01111" => DO3(4 downto 0)<= D307;
            when "10101" => DO3(4 downto 0)<= D308;
            when "11000" => DO3(4 downto 0)<= D309;
            when "01100" => DO3(4 downto 0)<= D310;
            when "00110" => DO3(4 downto 0)<= D311;
            when "00011" => DO3(4 downto 0)<= D312;
            when "10011" => DO3(4 downto 0)<= D313;
            when "11011" => DO3(4 downto 0)<= D314;
            when "11111" => DO3(4 downto 0)<= D315;
            when "11101" => DO3(4 downto 0)<= D316;
            when "11100" => DO3(4 downto 0)<= D317;
            when "01110" => DO3(4 downto 0)<= D318;
            when "00111" => DO3(4 downto 0)<= D319;
            when "10001" => DO3(4 downto 0)<= D320;
            when "11010" => DO3(4 downto 0)<= D321;
            when "01101" => DO3(4 downto 0)<= D322;
            when "10100" => DO3(4 downto 0)<= D323;
            when "01010" => DO3(4 downto 0)<= D324;
            when "00101" => DO3(4 downto 0)<= D325;
            when "10000" => DO3(4 downto 0)<= D326;
            when "01000" => DO3(4 downto 0)<= D327;
            when "00100" => DO3(4 downto 0)<= D328;
            when "00010" => DO3(4 downto 0)<= D329;
            when "00001" => DO3(4 downto 0)<= D330;
            when "00000" => DO3(4 downto 0)<= D331;
            when others => D300(4 downto 0):= D300;
        end case;

--write
    else
        case ADR3 is
            when "10010" => D300(4 downto 0):= DI;
            when "01001" => D301(4 downto 0):= DI;
            when "10110" => D302(4 downto 0):= DI;

```

```

when "01011" => D303(4 downto 0):= DI;
when "10111" => D304(4 downto 0):= DI;
when "11001" => D305(4 downto 0):= DI;
when "11110" => D306(4 downto 0):= DI;
when "01111" => D307(4 downto 0):= DI;
when "10101" => D308(4 downto 0):= DI;
when "11000" => D309(4 downto 0):= DI;
when "01100" => D310(4 downto 0):= DI;
when "00110" => D311(4 downto 0):= DI;
when "00011" => D312(4 downto 0):= DI;
when "10011" => D313(4 downto 0):= DI;
when "11011" => D314(4 downto 0):= DI;
when "11111" => D315(4 downto 0):= DI;
when "11101" => D316(4 downto 0):= DI;
when "11100" => D317(4 downto 0):= DI;
when "01110" => D318(4 downto 0):= DI;
when "00111" => D319(4 downto 0):= DI;
when "10001" => D320(4 downto 0):= DI;
when "11010" => D321(4 downto 0):= DI;
when "01101" => D322(4 downto 0):= DI;
when "10100" => D323(4 downto 0):= DI;
when "01010" => D324(4 downto 0):= DI;
when "00101" => D325(4 downto 0):= DI;
when "10000" => D326(4 downto 0):= DI;
when "01000" => D327(4 downto 0):= DI;
when "00100" => D328(4 downto 0):= DI;
when "00010" => D329(4 downto 0):= DI;
when "00001" => D330(4 downto 0):= DI;
when "00000" => D331(4 downto 0):= DI;
when others => D300(4 downto 0):= D300;
end case;

```

```

end if;
end process;

```

```

-----
--MEMORIA RAM 4--
-----

```

```

process (RW1, ADRo, DIo, CLK, RESET)

```

```

variable D400: STD_LOGIC_VECTOR (4 downto 0);
variable D401: STD_LOGIC_VECTOR (4 downto 0);
variable D402: STD_LOGIC_VECTOR (4 downto 0);
variable D403: STD_LOGIC_VECTOR (4 downto 0);
variable D404: STD_LOGIC_VECTOR (4 downto 0);
variable D405: STD_LOGIC_VECTOR (4 downto 0);
variable D406: STD_LOGIC_VECTOR (4 downto 0);
variable D407: STD_LOGIC_VECTOR (4 downto 0);
variable D408: STD_LOGIC_VECTOR (4 downto 0);
variable D409: STD_LOGIC_VECTOR (4 downto 0);
variable D410: STD_LOGIC_VECTOR (4 downto 0);
variable D411: STD_LOGIC_VECTOR (4 downto 0);
variable D412: STD_LOGIC_VECTOR (4 downto 0);
variable D413: STD_LOGIC_VECTOR (4 downto 0);
variable D414: STD_LOGIC_VECTOR (4 downto 0);
variable D415: STD_LOGIC_VECTOR (4 downto 0);
variable D416: STD_LOGIC_VECTOR (4 downto 0);
variable D417: STD_LOGIC_VECTOR (4 downto 0);
variable D418: STD_LOGIC_VECTOR (4 downto 0);
variable D419: STD_LOGIC_VECTOR (4 downto 0);
variable D420: STD_LOGIC_VECTOR (4 downto 0);
variable D421: STD_LOGIC_VECTOR (4 downto 0);
variable D422: STD_LOGIC_VECTOR (4 downto 0);
variable D423: STD_LOGIC_VECTOR (4 downto 0);
variable D424: STD_LOGIC_VECTOR (4 downto 0);

```

```

variable D425: STD_LOGIC_VECTOR (4 downto 0);
variable D426: STD_LOGIC_VECTOR (4 downto 0);
variable D427: STD_LOGIC_VECTOR (4 downto 0);
variable D428: STD_LOGIC_VECTOR (4 downto 0);
variable D429: STD_LOGIC_VECTOR (4 downto 0);
variable D430: STD_LOGIC_VECTOR (4 downto 0);
variable D431: STD_LOGIC_VECTOR (4 downto 0);
begin
--reset
    if RESET = '1' then
        D400:="00000";
        D401:="00000";
        D402:="00000";
        D403:="00000";
        D404:="00000";
        D405:="00000";
        D406:="00000";
        D407:="00000";
        D408:="00000";
        D409:="00000";
        D410:="00000";
        D411:="00000";
        D412:="00000";
        D413:="00000";
        D414:="00000";
        D415:="00000";
        D416:="00000";
        D417:="00000";
        D418:="00000";
        D419:="00000";
        D420:="00000";
        D421:="00000";
        D422:="00000";
        D423:="00000";
        D424:="00000";
        D425:="00000";
        D426:="00000";
        D427:="00000";
        D428:="00000";
        D429:="00000";
        D430:="00000";
        D431:="00000";
    --read
    elsif RW1 = '0' then
        case ADRo is
            when "10010" => DO4(4 downto 0)<= D400;
            when "01001" => DO4(4 downto 0)<= D401;
            when "10110" => DO4(4 downto 0)<= D402;
            when "01011" => DO4(4 downto 0)<= D403;
            when "10111" => DO4(4 downto 0)<= D404;
            when "11001" => DO4(4 downto 0)<= D405;
            when "11110" => DO4(4 downto 0)<= D406;
            when "01111" => DO4(4 downto 0)<= D407;
            when "10101" => DO4(4 downto 0)<= D408;
            when "11000" => DO4(4 downto 0)<= D409;
            when "01100" => DO4(4 downto 0)<= D410;
            when "00110" => DO4(4 downto 0)<= D411;
            when "00011" => DO4(4 downto 0)<= D412;
            when "10011" => DO4(4 downto 0)<= D413;
            when "11011" => DO4(4 downto 0)<= D414;
            when "11111" => DO4(4 downto 0)<= D415;
            when "11101" => DO4(4 downto 0)<= D416;
            when "11100" => DO4(4 downto 0)<= D417;
            when "01110" => DO4(4 downto 0)<= D418;
            when "00111" => DO4(4 downto 0)<= D419;
            when "10001" => DO4(4 downto 0)<= D420;
        end case;
    end if;
end begin;

```



```

when "11010" => DO4(4 downto 0)<= D421;
when "01101" => DO4(4 downto 0)<= D422;
when "10100" => DO4(4 downto 0)<= D423;
when "01010" => DO4(4 downto 0)<= D424;
when "00101" => DO4(4 downto 0)<= D425;
when "10000" => DO4(4 downto 0)<= D426;
when "01000" => DO4(4 downto 0)<= D427;
when "00100" => DO4(4 downto 0)<= D428;
when "00010" => DO4(4 downto 0)<= D429;
when "00001" => DO4(4 downto 0)<= D430;
when "00000" => DO4(4 downto 0)<= D431;
when others => D400(4 downto 0);= D400;
end case;

--write
else

case ADRo is
when "10010" => D400(4 downto 0);= D400;
when "01001" => D401(4 downto 0);= D401;
when "10110" => D402(4 downto 0);= D402;
when "01011" => D403(4 downto 0);= D403;
when "10111" => D404(4 downto 0);= D404;
when "11001" => D405(4 downto 0);= D405;
when "11110" => D406(4 downto 0);= D406;
when "01111" => D407(4 downto 0);= D407;
when "10101" => D408(4 downto 0);= D408;
when "11000" => D409(4 downto 0);= D409;
when "01100" => D410(4 downto 0);= D410;
when "00110" => D411(4 downto 0);= D411;
when "00011" => D412(4 downto 0);= D412;
when "10011" => D413(4 downto 0);= D413;
when "11011" => D414(4 downto 0);= D414;
when "11111" => D415(4 downto 0);= D415;
when "11101" => D416(4 downto 0);= D416;
when "11100" => D417(4 downto 0);= D417;
when "01110" => D418(4 downto 0);= D418;
when "00111" => D419(4 downto 0);= D419;
when "10001" => D420(4 downto 0);= D420;
when "11010" => D421(4 downto 0);= D421;
when "01101" => D422(4 downto 0);= D422;
when "10100" => D423(4 downto 0);= D423;
when "01010" => D424(4 downto 0);= D424;
when "00101" => D425(4 downto 0);= D425;
when "10000" => D426(4 downto 0);= D426;
when "01000" => D427(4 downto 0);= D427;
when "00100" => D428(4 downto 0);= D428;
when "00010" => D429(4 downto 0);= D429;
when "00001" => D430(4 downto 0);= D430;
when "00000" => D431(4 downto 0);= D431;
when others => D400(4 downto 0);= D400;
end case;

end if;
end process;

-----
--MEMORIA RAM 5--
-----

process (RW2, ADRo, DIO, CLK, RESET)

variable D500: STD_LOGIC_VECTOR (4 downto 0);
variable D501: STD_LOGIC_VECTOR (4 downto 0);
variable D502: STD_LOGIC_VECTOR (4 downto 0);
variable D503: STD_LOGIC_VECTOR (4 downto 0);
variable D504: STD_LOGIC_VECTOR (4 downto 0);

```

```
variable D505: STD_LOGIC_VECTOR (4 downto 0);
variable D506: STD_LOGIC_VECTOR (4 downto 0);
variable D507: STD_LOGIC_VECTOR (4 downto 0);
variable D508: STD_LOGIC_VECTOR (4 downto 0);
variable D509: STD_LOGIC_VECTOR (4 downto 0);
variable D510: STD_LOGIC_VECTOR (4 downto 0);
variable D511: STD_LOGIC_VECTOR (4 downto 0);
variable D512: STD_LOGIC_VECTOR (4 downto 0);
variable D513: STD_LOGIC_VECTOR (4 downto 0);
variable D514: STD_LOGIC_VECTOR (4 downto 0);
variable D515: STD_LOGIC_VECTOR (4 downto 0);
variable D516: STD_LOGIC_VECTOR (4 downto 0);
variable D517: STD_LOGIC_VECTOR (4 downto 0);
variable D518: STD_LOGIC_VECTOR (4 downto 0);
variable D519: STD_LOGIC_VECTOR (4 downto 0);
variable D520: STD_LOGIC_VECTOR (4 downto 0);
variable D521: STD_LOGIC_VECTOR (4 downto 0);
variable D522: STD_LOGIC_VECTOR (4 downto 0);
variable D523: STD_LOGIC_VECTOR (4 downto 0);
variable D524: STD_LOGIC_VECTOR (4 downto 0);
variable D525: STD_LOGIC_VECTOR (4 downto 0);
variable D526: STD_LOGIC_VECTOR (4 downto 0);
variable D527: STD_LOGIC_VECTOR (4 downto 0);
variable D528: STD_LOGIC_VECTOR (4 downto 0);
variable D529: STD_LOGIC_VECTOR (4 downto 0);
variable D530: STD_LOGIC_VECTOR (4 downto 0);
variable D531: STD_LOGIC_VECTOR (4 downto 0);
begin
--reset
    if RESET = '1' then
        D500:="00000";
        D501:="00000";
        D502:="00000";
        D503:="00000";
        D504:="00000";
        D505:="00000";
        D506:="00000";
        D507:="00000";
        D508:="00000";
        D509:="00000";
        D510:="00000";
        D511:="00000";
        D512:="00000";
        D513:="00000";
        D514:="00000";
        D515:="00000";
        D516:="00000";
        D517:="00000";
        D518:="00000";
        D519:="00000";
        D520:="00000";
        D521:="00000";
        D522:="00000";
        D523:="00000";
        D524:="00000";
        D525:="00000";
        D526:="00000";
        D527:="00000";
        D528:="00000";
        D529:="00000";
        D530:="00000";
        D531:="00000";
    --read
    elsif RW2 = '0' then
        case ADRO is
            when "10010" => DO5(4 downto 0)<= D500;
```

```

when "01001" => DO5(4 downto 0) <= D501;
when "10110" => DO5(4 downto 0) <= D502;
when "01011" => DO5(4 downto 0) <= D503;
when "10111" => DO5(4 downto 0) <= D504;
when "11001" => DO5(4 downto 0) <= D505;
when "11110" => DO5(4 downto 0) <= D506;
when "01111" => DO5(4 downto 0) <= D507;
when "10101" => DO5(4 downto 0) <= D508;
when "11000" => DO5(4 downto 0) <= D509;
when "01100" => DO5(4 downto 0) <= D510;
when "00110" => DO5(4 downto 0) <= D511;
when "00011" => DO5(4 downto 0) <= D512;
when "10011" => DO5(4 downto 0) <= D513;
when "11011" => DO5(4 downto 0) <= D514;
when "11111" => DO5(4 downto 0) <= D515;
when "11101" => DO5(4 downto 0) <= D516;
when "11100" => DO5(4 downto 0) <= D517;
when "01110" => DO5(4 downto 0) <= D518;
when "00111" => DO5(4 downto 0) <= D519;
when "10001" => DO5(4 downto 0) <= D520;
when "11010" => DO5(4 downto 0) <= D521;
when "01101" => DO5(4 downto 0) <= D522;
when "10100" => DO5(4 downto 0) <= D523;
when "01010" => DO5(4 downto 0) <= D524;
when "00101" => DO5(4 downto 0) <= D525;
when "10000" => DO5(4 downto 0) <= D526;
when "01000" => DO5(4 downto 0) <= D527;
when "00100" => DO5(4 downto 0) <= D528;
when "00010" => DO5(4 downto 0) <= D529;
when "00001" => DO5(4 downto 0) <= D530;
when "00000" => DO5(4 downto 0) <= D531;
when others => D500(4 downto 0) = D500;
end case;

```

```
--write
```

```
else
```

```

case ADRo is
when "10010" => D500(4 downto 0) = Dlo;
when "01001" => D501(4 downto 0) = Dlo;
when "10110" => D502(4 downto 0) = Dlo;
when "01011" => D503(4 downto 0) = Dlo;
when "10111" => D504(4 downto 0) = Dlo;
when "11001" => D505(4 downto 0) = Dlo;
when "11110" => D506(4 downto 0) = Dlo;
when "01111" => D507(4 downto 0) = Dlo;
when "10101" => D508(4 downto 0) = Dlo;
when "11000" => D509(4 downto 0) = Dlo;
when "01100" => D510(4 downto 0) = Dlo;
when "00110" => D511(4 downto 0) = Dlo;
when "00011" => D512(4 downto 0) = Dlo;
when "10011" => D513(4 downto 0) = Dlo;
when "11011" => D514(4 downto 0) = Dlo;
when "11111" => D515(4 downto 0) = Dlo;
when "11101" => D516(4 downto 0) = Dlo;
when "11100" => D517(4 downto 0) = Dlo;
when "01110" => D518(4 downto 0) = Dlo;
when "00111" => D519(4 downto 0) = Dlo;
when "10001" => D520(4 downto 0) = Dlo;
when "11010" => D521(4 downto 0) = Dlo;
when "01101" => D522(4 downto 0) = Dlo;
when "10100" => D523(4 downto 0) = Dlo;
when "01010" => D524(4 downto 0) = Dlo;
when "00101" => D525(4 downto 0) = Dlo;
when "10000" => D526(4 downto 0) = Dlo;
when "01000" => D527(4 downto 0) = Dlo;

```

```
when "00100" => D528(4 downto 0):= Dlo;  
when "00010" => D529(4 downto 0):= Dlo;  
when "00001" => D530(4 downto 0):= Dlo;  
when "00000" => D531(4 downto 0):= Dlo;  
when others => D500(4 downto 0):= D500;  
end case;
```

```
end if;  
end process;
```

-----  
--MEMORIA RAM 6--  
-----

```
process (RW3, ADRo, DIO, CLK, RESET)
```

```
variable D600: STD_LOGIC_VECTOR (4 downto 0);  
variable D601: STD_LOGIC_VECTOR (4 downto 0);  
variable D602: STD_LOGIC_VECTOR (4 downto 0);  
variable D603: STD_LOGIC_VECTOR (4 downto 0);  
variable D604: STD_LOGIC_VECTOR (4 downto 0);  
variable D605: STD_LOGIC_VECTOR (4 downto 0);  
variable D606: STD_LOGIC_VECTOR (4 downto 0);  
variable D607: STD_LOGIC_VECTOR (4 downto 0);  
variable D608: STD_LOGIC_VECTOR (4 downto 0);  
variable D609: STD_LOGIC_VECTOR (4 downto 0);  
variable D610: STD_LOGIC_VECTOR (4 downto 0);  
variable D611: STD_LOGIC_VECTOR (4 downto 0);  
variable D612: STD_LOGIC_VECTOR (4 downto 0);  
variable D613: STD_LOGIC_VECTOR (4 downto 0);  
variable D614: STD_LOGIC_VECTOR (4 downto 0);  
variable D615: STD_LOGIC_VECTOR (4 downto 0);  
variable D616: STD_LOGIC_VECTOR (4 downto 0);  
variable D617: STD_LOGIC_VECTOR (4 downto 0);  
variable D618: STD_LOGIC_VECTOR (4 downto 0);  
variable D619: STD_LOGIC_VECTOR (4 downto 0);  
variable D620: STD_LOGIC_VECTOR (4 downto 0);  
variable D621: STD_LOGIC_VECTOR (4 downto 0);  
variable D622: STD_LOGIC_VECTOR (4 downto 0);  
variable D623: STD_LOGIC_VECTOR (4 downto 0);  
variable D624: STD_LOGIC_VECTOR (4 downto 0);  
variable D625: STD_LOGIC_VECTOR (4 downto 0);  
variable D626: STD_LOGIC_VECTOR (4 downto 0);  
variable D627: STD_LOGIC_VECTOR (4 downto 0);  
variable D628: STD_LOGIC_VECTOR (4 downto 0);  
variable D629: STD_LOGIC_VECTOR (4 downto 0);  
variable D630: STD_LOGIC_VECTOR (4 downto 0);  
variable D631: STD_LOGIC_VECTOR (4 downto 0);
```

```
begin  
--reset
```

```
if RESET = '1' then  
D600:="00000";  
D601:="00000";  
D602:="00000";  
D603:="00000";  
D604:="00000";  
D605:="00000";  
D606:="00000";  
D607:="00000";  
D608:="00000";  
D609:="00000";  
D610:="00000";  
D611:="00000";
```

```

D612:="00000";
D613:="00000";
D614:="00000";
D615:="00000";
D616:="00000";
D617:="00000";
D618:="00000";
D619:="00000";
D620:="00000";
D621:="00000";
D622:="00000";
D623:="00000";
D624:="00000";
D625:="00000";
D626:="00000";
D627:="00000";
D628:="00000";
D629:="00000";
D630:="00000";
D631:="00000";

--read
    elsif RW3 = '0' then
        case ADRo is
            when "10010" => DO6(4 downto 0)<= D600;
            when "01001" => DO6(4 downto 0)<= D601;
            when "10110" => DO6(4 downto 0)<= D602;
            when "01011" => DO6(4 downto 0)<= D603;
            when "10111" => DO6(4 downto 0)<= D604;
            when "11001" => DO6(4 downto 0)<= D605;
            when "11110" => DO6(4 downto 0)<= D606;
            when "01111" => DO6(4 downto 0)<= D607;
            when "10101" => DO6(4 downto 0)<= D608;
            when "11000" => DO6(4 downto 0)<= D609;
            when "01100" => DO6(4 downto 0)<= D610;
            when "00110" => DO6(4 downto 0)<= D611;
            when "00011" => DO6(4 downto 0)<= D612;
            when "10011" => DO6(4 downto 0)<= D613;
            when "11011" => DO6(4 downto 0)<= D614;
            when "11111" => DO6(4 downto 0)<= D615;
            when "11101" => DO6(4 downto 0)<= D616;
            when "11100" => DO6(4 downto 0)<= D617;
            when "01110" => DO6(4 downto 0)<= D618;
            when "00111" => DO6(4 downto 0)<= D619;
            when "10001" => DO6(4 downto 0)<= D620;
            when "11010" => DO6(4 downto 0)<= D621;
            when "01101" => DO6(4 downto 0)<= D622;
            when "10100" => DO6(4 downto 0)<= D623;
            when "01010" => DO6(4 downto 0)<= D624;
            when "00101" => DO6(4 downto 0)<= D625;
            when "10000" => DO6(4 downto 0)<= D626;
            when "01000" => DO6(4 downto 0)<= D627;
            when "00100" => DO6(4 downto 0)<= D628;
            when "00010" => DO6(4 downto 0)<= D629;
            when "00001" => DO6(4 downto 0)<= D630;
            when "00000" => DO6(4 downto 0)<= D631;
            when others => D600(4 downto 0):= D600;
        end case;

--write
    else
        case ADRo is
            when "10010" => D600(4 downto 0):= Dlo;
            when "01001" => D601(4 downto 0):= Dlo;
            when "10110" => D602(4 downto 0):= Dlo;
            when "01011" => D603(4 downto 0):= Dlo;

```

```

when "10111" => D604(4 downto 0):= Dlo;
when "11001" => D605(4 downto 0):= Dlo;
when "11110" => D606(4 downto 0):= Dlo;
when "01111" => D607(4 downto 0):= Dlo;
when "10101" => D608(4 downto 0):= Dlo;
when "11000" => D609(4 downto 0):= Dlo;
when "01100" => D610(4 downto 0):= Dlo;
when "00110" => D611(4 downto 0):= Dlo;
when "00011" => D612(4 downto 0):= Dlo;
when "10011" => D613(4 downto 0):= Dlo;
when "11011" => D614(4 downto 0):= Dlo;
when "11111" => D615(4 downto 0):= Dlo;
when "11101" => D616(4 downto 0):= Dlo;
when "11100" => D617(4 downto 0):= Dlo;
when "01110" => D618(4 downto 0):= Dlo;
when "00111" => D619(4 downto 0):= Dlo;
when "10001" => D620(4 downto 0):= Dlo;
when "11010" => D621(4 downto 0):= Dlo;
when "01101" => D622(4 downto 0):= Dlo;
when "10100" => D623(4 downto 0):= Dlo;
when "01010" => D624(4 downto 0):= Dlo;
when "00101" => D625(4 downto 0):= Dlo;
when "10000" => D626(4 downto 0):= Dlo;
when "01000" => D627(4 downto 0):= Dlo;
when "00100" => D628(4 downto 0):= Dlo;
when "00010" => D629(4 downto 0):= Dlo;
when "00001" => D630(4 downto 0):= Dlo;
when "00000" => D631(4 downto 0):= Dlo;
when others => D600(4 downto 0):= D600;
end case;

```

```

end if;
end process;

```

```

-----
--CALC FIAB--
-----

```

```

process (RESET,CLK,R_EM1,GALLOIS,P0F,P1F,P2F,P3F,P4F,MET0,REG0,
        POS00,POS01,POS02,POS03,MET1,REG1,POS10,POS11,POS12,
        POS13,MET2,REG2,POS20,POS21,POS22,POS23,MET3,REG3,POS30,
        POS31,POS32,POS33)

```

```
--BIT 0
```

```

-- Métrica de palabra designada (Concurrente 0):
VARIABLE MET_CONC00: STD_LOGIC;
-- Métrica de las palabras concurrentes (Concurrente 1,2,3):
VARIABLE MET_CONC01: STD_LOGIC;
VARIABLE MET_CONC02: STD_LOGIC;
VARIABLE MET_CONC03: STD_LOGIC;

```

```
--BIT 1
```

```

-- Métrica de palabra designada (Concurrente 0):
VARIABLE MET_CONC10: STD_LOGIC;
-- Métrica de las palabras concurrentes (Concurrente 1,2,3):
VARIABLE MET_CONC11: STD_LOGIC;
VARIABLE MET_CONC12: STD_LOGIC;
VARIABLE MET_CONC13: STD_LOGIC;

```

```
--BIT 2
```

```

-- Métrica de palabra designada (Concurrente 0):
VARIABLE MET_CONC20: STD_LOGIC;
-- Métrica de las palabras concurrentes (Concurrente 1,2,3):
VARIABLE MET_CONC21: STD_LOGIC;
VARIABLE MET_CONC22: STD_LOGIC;
VARIABLE MET_CONC23: STD_LOGIC;

```

## Turbo decodificador de códigos producto

---

```
--BIT 3
-- Métrica de palabra designada (Concurrente 0):
VARIABLE MET_CONC30: STD_LOGIC;
-- Métrica de las palabras concurrentes (Concurrente 1,2,3):
VARIABLE MET_CONC31: STD_LOGIC;
VARIABLE MET_CONC32: STD_LOGIC;
VARIABLE MET_CONC33: STD_LOGIC;

--BIT 4
-- Métrica de palabra designada (Concurrente 0):
VARIABLE MET_CONC40: STD_LOGIC;
-- Métrica de las palabras concurrentes (Concurrente 1,2,3):
VARIABLE MET_CONC41: STD_LOGIC;
VARIABLE MET_CONC42: STD_LOGIC;
VARIABLE MET_CONC43: STD_LOGIC;

-- Bits modificados:
VARIABLE BMODIF0: STD_LOGIC;
VARIABLE BMODIF1: STD_LOGIC;
VARIABLE BMODIF2: STD_LOGIC;
VARIABLE BMODIF3: STD_LOGIC;
-- Variables temporales para la suma/resta:
VARIABLE MET0S: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE MET1S: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE MET2S: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE MET3S: STD_LOGIC_VECTOR(6 downto 0);

VARIABLE FIA2S: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE FIA3S: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE FIA4S: STD_LOGIC_VECTOR(6 downto 0);
VARIABLE REMIS: STD_LOGIC_VECTOR(6 downto 0);

VARIABLE DOS: STD_LOGIC_VECTOR(6 downto 0);

-- Contador de 0 a 4:
--SIGNAL i: STD_LOGIC_VECTOR(2 downto 0);

-- XOR:
VARIABLE XOR_1: STD_LOGIC;
VARIABLE XOR_2: STD_LOGIC;
VARIABLE XOR_3: STD_LOGIC;

begin
    if RESET='1' then
        INVER<='0';
        FIAB<="0000000";
--    i=0:
        MET_CONC00:=0';
        MET_CONC01:=0';
        MET_CONC02:=0';
        MET_CONC03:=0';
--    i=1:
        MET_CONC10:=0';
        MET_CONC11:=0';
        MET_CONC12:=0';
        MET_CONC13:=0';
--    i=2:
        MET_CONC20:=0';
        MET_CONC21:=0';
        MET_CONC22:=0';
        MET_CONC23:=0';
--    i=3:
        MET_CONC30:=0';
        MET_CONC31:=0';
        MET_CONC32:=0';
        MET_CONC33:=0';
-- fin
```

```

--
i<="000";
BMODIF0:=0';
BMODIF1:=0';
BMODIF2:=0';
BMODIF3:=0';
-- Variables temporales para la suma:
MET0S(5 downto 0):=MET0;
MET0S(6):=0';
MET1S(5 downto 0):=MET1;
MET1S(6):=0';
MET2S(5 downto 0):=MET2;
MET2S(6):=0';
MET3S(5 downto 0):=MET3;
MET3S(6):=0';
FIA2S(3 downto 0):=P2F;    --FIA2
FIA2S(6 downto 4):="000";
FIA3S(3 downto 0):=P3F;    --FIA3
FIA3S(6 downto 4):="000";
FIA4S(3 downto 0):=P4F;    --FIA4
FIA4S(6 downto 4):="000";
REMIS(4 downto 0):=R_EMI;
REMIS(6 downto 5):="00";
DOS:"0000010";
--elsif CLK'event and CLK='1' then
else
--if CLK'event and CLK='1' then
-- Variables temporales para la suma:
MET0S(5 downto 0):=MET0;
MET0S(6):=0';
MET1S(5 downto 0):=MET1;
MET1S(6):=0';
MET2S(5 downto 0):=MET2;
MET2S(6):=0';
MET3S(5 downto 0):=MET3;
MET3S(6):=0';
FIA2S(3 downto 0):=P2F;
FIA2S(6 downto 4):="000";
FIA3S(3 downto 0):=P3F;
FIA3S(6 downto 4):="000";
FIA4S(3 downto 0):=P4F;
FIA4S(6 downto 4):="000";
REMIS(4 downto 0):=R_EMI;
REMIS(6 downto 5):="00";
DOS:"0000010";
--end if;

-- palabra designada con b0
--if CLK'event then
if GALOIS=POS00 and REG0(0)='1' then
MET_CONC00:='1';
else
MET_CONC00:=0';
end if;
-- palabra concurrente 1 con b0
if GALOIS=POS10 and REG1(0)='1' then
MET_CONC01:='1';
else
MET_CONC01:=0';
end if;
-- palabra concurrente 2 con b0
if GALOIS=POS20 and REG2(0)='1' then
MET_CONC02:='1';
else
MET_CONC02:=0';

```



```

end if;
-- palabra concurrente 3 con b0
if GALOIS=POS30 and REG3(0)='1' then
    MET_CONC03:='1';
else
    MET_CONC03:='0';
end if;
-- palabra designada con b1
if GALOIS=POS01 and REG0(1)='1' then
    MET_CONC10:='1';
else
    MET_CONC10:='0';
end if;
-- palabra concurrente 1 con b1
if GALOIS=POS11 and REG1(1)='1' then
    MET_CONC11:='1';
else
    MET_CONC11:='0';
end if;
-- palabra concurrente 2 con b1
if GALOIS=POS21 and REG2(1)='1' then
    MET_CONC12:='1';
else
    MET_CONC12:='0';
end if;
-- palabra concurrente 3 con b1
if GALOIS=POS31 and REG3(1)='1' then
    MET_CONC13:='1';
else
    MET_CONC13:='0';
end if;
-- palabra designada con b2
if GALOIS=POS02 and REG0(2)='1' then
    MET_CONC20:='1';
else
    MET_CONC20:='0';
end if;
-- palabra concurrente 1 con b2
if GALOIS=POS12 and REG1(2)='1' then
    MET_CONC21:='1';
else
    MET_CONC21:='0';
end if;
-- palabra concurrente 2 con b2
if GALOIS=POS22 and REG2(2)='1' then
    MET_CONC22:='1';
else
    MET_CONC22:='0';
end if;
-- palabra concurrente 3 con b2
if GALOIS=POS32 and REG3(2)='1' then
    MET_CONC23:='1';
else
    MET_CONC23:='0';
end if;
-- palabra designada con b3
if GALOIS=POS03 and REG0(3)='1' then
    MET_CONC30:='1';
else
    MET_CONC30:='0';
end if;
-- palabra concurrente 1 con b3
if GALOIS=POS13 and REG1(3)='1' then
    MET_CONC31:='1';
else
    MET_CONC31:='0';
end if;

```

```

end if;
-- palabra concurrente 2 con b3
if GALOIS=POS23 and REG2(3)='1' then
    MET_CONC32:='1';
else
    MET_CONC32:='0';
end if;
-- palabra concurrente 3 con b3
if GALOIS=POS33 and REG3(3)='1' then
    MET_CONC33:='1';
else
    MET_CONC33:='0';
end if;
-- palabra designada con b4
if GALOIS="00000" and REG0(4)='1' then
    MET_CONC40:='1';
else
    MET_CONC40:='0';
end if;
-- palabra concurrente 1 con b4
if GALOIS="00000" and REG1(4)='1' then
    MET_CONC41:='1';
else
    MET_CONC41:='0';
end if;
-- palabra concurrente 2 con b4
if GALOIS="00000" and REG2(4)='1' then
    MET_CONC42:='1';
else
    MET_CONC42:='0';
end if;
-- palabra concurrente 3 con b4
if GALOIS="00000" and REG3(4)='1' then
    MET_CONC43:='1';
else
    MET_CONC43:='0';
end if;
-- Bits modificados:
BMODIF0:=MET_CONC00 OR MET_CONC10 OR MET_CONC20 OR MET_CONC30 OR
MET_CONC40;
BMODIF1:=MET_CONC01 OR MET_CONC11 OR MET_CONC21 OR MET_CONC31 OR
MET_CONC41;
BMODIF2:=MET_CONC02 OR MET_CONC12 OR MET_CONC22 OR MET_CONC32 OR
MET_CONC42;
BMODIF3:=MET_CONC03 OR MET_CONC13 OR MET_CONC23 OR MET_CONC33 OR
MET_CONC43;

INVER<=MET_CONC00 OR MET_CONC10 OR MET_CONC20 OR MET_CONC30 OR
MET_CONC40;

XOR_1:=BMODIF1 xor BMODIF0;
XOR_2:=BMODIF2 xor BMODIF0;
XOR_3:=BMODIF3 xor BMODIF0;

if DELAY='1' then
    FIAB<=FIA2S+FIA3S+FIA4S+REMIS+DOS;
elsif XOR_1='1' then
    -- Palabra concurrente 1:
    FIAB<=MET1S-MET0S;
elsif XOR_2='1' then
    -- Palabra concurrente 2:
    FIAB<=MET2S-MET0S;
elsif XOR_3='1' then

```

```

        -- Palabra concurrente 3:
            FIAB<=MET3S-MET0S;
        else
        -- No hay palabra concurrente:
            FIAB<=FIA2S+FIA3S+FIA4S+REMIS-MET0S;
        end if;
    --end if;
end if;

end process ;

-----
--INFORMACIÓN EXTRÍNSECA--
-----

process (CLK, RESET, INVER, FIAB, R_EMI)

--VARIABLE REMIS2: STD_LOGIC_VECTOR(4 downto 0);
--VARIABLE FIAB2: STD_LOGIC_VECTOR(6 downto 0);

    VARIABLE VL: STD_LOGIC_VECTOR (6 downto 0);
    VARIABLE VM: STD_LOGIC_VECTOR (6 downto 0);
    VARIABLE VNEGADO: STD_LOGIC;
    VARIABLE VNORMAL: STD_LOGIC;

begin

    VM:="0000000";
    VL:=(VM+R_EMI(3 downto 0)+FIAB);
    --REMIS2:=R_EMI;
    VNEGADO:=NOT R_EMI(4);
    VNORMAL:=R_EMI(4);

    if RESET='1' then
        IE_W<="0000000";
        SIGNO_W<='0';
        D_BIN<='0';
        VL:= "0000000";
        VNORMAL:='0';
        VNEGADO:='0';
    --elsif --and CLK'event then
    else --elsif CLK'event then

        if(INVER='1')then
            if (VL>63) then
                IE_W<="0111111"; --K<="111111";
                --SIGNO_W<= NOT R_EMI(4);
                --if CLK'event then
                SIGNO_W<= VNEGADO; --NOT R_EMI(4);
                D_BIN<= VNEGADO; --NOT R_EMI(4);
                --D_BIN<= NEGADO;
                --IE_W<=K;
                --end if;
            else
                --IE_W<=(R_EMI(3 downto 0)+FIAB); -- K<=L;
                IE_W<=VL;
                --if CLK'event then
                SIGNO_W<= VNEGADO; --NOT R_EMI(4);
                D_BIN<= VNEGADO; --NOT R_EMI(4);
                --IE_W<=K;
                --end if;
            end if;
        else
    
```

```

        if(R_EMI(3 downto 0)>FIAB)then
            IE_W<=(R_EMI(3 downto 0)-FIAB);
            --if CLK'event then
            SIGNO_W<= VNEGADO;-- NOT R_EMI(4);
            D_BIN<= VNORMAL; --R_EMI(4);
            --IE_W<=K;
            --end if;
        else
            IE_W<=(FIAB-R_EMI(3 downto 0));
            --if CLK'event then
            SIGNO_W<= VNORMAL;--R_EMI(4);
            D_BIN<= VNORMAL;--R_EMI(4);
            --IE_W<=K;
            --end if;
        end if;
    end if;
end process;

```

```

-----
--PRODUCTO--
-----

```

```

process( CLK, RESET, IE_W, ITER)
begin

```

```

    ITER<="0001";

```

```

    if ITER=1 then      --alfa = 0.35
    if      IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0000";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0001";
    elsif IE_W="000100" then AW<="0001";
    elsif IE_W="000101" then AW<="0010";
    elsif IE_W="000110" then AW<="0010";
    elsif IE_W="000111" then AW<="0010";
    elsif IE_W="001000" then AW<="0011";
    elsif IE_W="001001" then AW<="0011";
    elsif IE_W="001010" then AW<="0100";
    elsif IE_W="001011" then AW<="0100";
    elsif IE_W="001100" then AW<="0100";
    elsif IE_W="001101" then AW<="0101";
    elsif IE_W="001110" then AW<="0101";
    elsif IE_W="001111" then AW<="0101";
    elsif IE_W="010000" then AW<="0110";
    elsif IE_W="010001" then AW<="0110";
    elsif IE_W="010010" then AW<="0110";
    elsif IE_W="010011" then AW<="0111";
    elsif IE_W="010100" then AW<="0111";
    elsif IE_W="010101" then AW<="0111";
    elsif IE_W="010110" then AW<="1000";
    elsif IE_W="010111" then AW<="1000";
    elsif IE_W="011000" then AW<="1000";
    elsif IE_W="011001" then AW<="1001";
    elsif IE_W="011010" then AW<="1001";
    elsif IE_W="011011" then AW<="1001";
    elsif IE_W="011100" then AW<="1010";
    elsif IE_W="011101" then AW<="1010";
    elsif IE_W="011110" then AW<="1011";
    elsif IE_W="011111" then AW<="1011";

```

```

    elsif IE_W="100000" then AW<="1011";-- Daria 1011= 11 en decimal

```

```

    elsif IE_W="100001" then AW<="1100";
    elsif IE_W="100010" then AW<="1100";
    elsif IE_W="100011" then AW<="1101";
    elsif IE_W="100100" then AW<="1101";
    elsif IE_W="100101" then AW<="1101";
    elsif IE_W="100110" then AW<="1110";
    elsif IE_W="100111" then AW<="1110";
    elsif IE_W="101000" then AW<="1110";
    elsif IE_W="101001" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=2 then
    -- alfa = 0.375
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0000";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0001";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0010";
    elsif IE_W="000110" then AW<="0010";
    elsif IE_W="000111" then AW<="0011";
    elsif IE_W="001000" then AW<="0011";
    elsif IE_W="001001" then AW<="0011";
    elsif IE_W="001010" then AW<="0100";
    elsif IE_W="001011" then AW<="0100";
    elsif IE_W="001100" then AW<="0101";
    elsif IE_W="001101" then AW<="0101";
    elsif IE_W="001110" then AW<="0101";
    elsif IE_W="001111" then AW<="0110";
    elsif IE_W="010000" then AW<="0110";
    elsif IE_W="010001" then AW<="0110";
    elsif IE_W="010010" then AW<="0111";
    elsif IE_W="010011" then AW<="0111";
    elsif IE_W="010100" then AW<="1000";
    elsif IE_W="010101" then AW<="1000";
    elsif IE_W="010110" then AW<="1000";
    elsif IE_W="010111" then AW<="1001";
    elsif IE_W="011000" then AW<="1001";
    elsif IE_W="011001" then AW<="1010";
    elsif IE_W="011010" then AW<="1010";
    elsif IE_W="011011" then AW<="1010";
    elsif IE_W="011100" then AW<="1011";
    elsif IE_W="011101" then AW<="1011";
    elsif IE_W="011110" then AW<="1011";
    elsif IE_W="011111" then AW<="1100";

    elsif IE_W="100000" then AW<="1100";
    elsif IE_W="100001" then AW<="1101";
    elsif IE_W="100010" then AW<="1101";
    elsif IE_W="100011" then AW<="1101";
    elsif IE_W="100100" then AW<="1110";
    elsif IE_W="100101" then AW<="1110";
    elsif IE_W="100110" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=3 then
    -- alfa = 0.4
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0000";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0001";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0010";
    elsif IE_W="000110" then AW<="0010";
    elsif IE_W="000111" then AW<="0011";
    elsif IE_W="001000" then AW<="0011";

```

```
elsif IE_W="001001" then AW<="0100";
elsif IE_W="001010" then AW<="0100";
elsif IE_W="001011" then AW<="0100";
elsif IE_W="001100" then AW<="0101";
elsif IE_W="001101" then AW<="0101";
elsif IE_W="001110" then AW<="0110";
elsif IE_W="001111" then AW<="0110";
elsif IE_W="010000" then AW<="0110";
elsif IE_W="010001" then AW<="0111";
elsif IE_W="010010" then AW<="0111";
elsif IE_W="010011" then AW<="1000";
elsif IE_W="010100" then AW<="1000";
elsif IE_W="010101" then AW<="1000";
elsif IE_W="010110" then AW<="1001";
elsif IE_W="010111" then AW<="1001";
elsif IE_W="011000" then AW<="1010";
elsif IE_W="011001" then AW<="1010";
elsif IE_W="011010" then AW<="1010";
elsif IE_W="011011" then AW<="1011";
elsif IE_W="011100" then AW<="1011";
elsif IE_W="011101" then AW<="1100";
elsif IE_W="011110" then AW<="1100";
elsif IE_W="011111" then AW<="1100";

elsif IE_W="100000" then AW<="1101";
elsif IE_W="100001" then AW<="1101";
elsif IE_W="100010" then AW<="1110";
elsif IE_W="100011" then AW<="1110";
elsif IE_W="100100" then AW<="1110";
else
    AW<="1111";
end if;
```

```
elsif ITER=4 then -- alfa = 0.425
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0000";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0001";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0010";
    elsif IE_W="000110" then AW<="0011";
    elsif IE_W="000111" then AW<="0011";
    elsif IE_W="001000" then AW<="0011";
    elsif IE_W="001001" then AW<="0100";
    elsif IE_W="001010" then AW<="0100";
    elsif IE_W="001011" then AW<="0101";
    elsif IE_W="001100" then AW<="0101";
    elsif IE_W="001101" then AW<="0110";
    elsif IE_W="001110" then AW<="0110";
    elsif IE_W="001111" then AW<="0110";
    elsif IE_W="010000" then AW<="0111";
    elsif IE_W="010001" then AW<="0111";
    elsif IE_W="010010" then AW<="1000";
    elsif IE_W="010011" then AW<="1000";
    elsif IE_W="010100" then AW<="1001";
    elsif IE_W="010101" then AW<="1001";
    elsif IE_W="010110" then AW<="1001";
    elsif IE_W="010111" then AW<="1010";
    elsif IE_W="011000" then AW<="1010";
    elsif IE_W="011001" then AW<="1011";
    elsif IE_W="011010" then AW<="1011";
    elsif IE_W="011011" then AW<="1011";
    elsif IE_W="011100" then AW<="1100";
    elsif IE_W="011101" then AW<="1100";
    elsif IE_W="011110" then AW<="1101";
    elsif IE_W="011111" then AW<="1101";
```

```
    elsif IE_W="100000" then AW<="1110";
    elsif IE_W="100001" then AW<="1110";
    elsif IE_W="100010" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=5 then
    -- alfa = 0.45
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0000";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0001";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0010";
    elsif IE_W="000110" then AW<="0011";
    elsif IE_W="000111" then AW<="0011";
    elsif IE_W="001000" then AW<="0100";
    elsif IE_W="001001" then AW<="0100";
    elsif IE_W="001010" then AW<="0101";
    elsif IE_W="001011" then AW<="0101";
    elsif IE_W="001100" then AW<="0101";
    elsif IE_W="001101" then AW<="0110";
    elsif IE_W="001110" then AW<="0110";
    elsif IE_W="001111" then AW<="0111";
    elsif IE_W="010000" then AW<="0111";
    elsif IE_W="010001" then AW<="1000";
    elsif IE_W="010010" then AW<="1000";
    elsif IE_W="010011" then AW<="1001";
    elsif IE_W="010100" then AW<="1001";
    elsif IE_W="010101" then AW<="1001";
    elsif IE_W="010110" then AW<="1010";
    elsif IE_W="010111" then AW<="1010";
    elsif IE_W="011000" then AW<="1011";
    elsif IE_W="011001" then AW<="1011";
    elsif IE_W="011010" then AW<="1100";
    elsif IE_W="011011" then AW<="1100";
    elsif IE_W="011100" then AW<="1101";
    elsif IE_W="011101" then AW<="1101";
    elsif IE_W="011110" then AW<="1110";
    elsif IE_W="011111" then AW<="1110";

    elsif IE_W="100000" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=6 then
    -- alfa = 0.475
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0000";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0001";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0010";
    elsif IE_W="000110" then AW<="0011";
    elsif IE_W="000111" then AW<="0011";
    elsif IE_W="001000" then AW<="0100";
    elsif IE_W="001001" then AW<="0100";
    elsif IE_W="001010" then AW<="0101";
    elsif IE_W="001011" then AW<="0101";
    elsif IE_W="001100" then AW<="0110";
    elsif IE_W="001101" then AW<="0110";
    elsif IE_W="001110" then AW<="0111";
    elsif IE_W="001111" then AW<="0111";
    elsif IE_W="010000" then AW<="1000";
    elsif IE_W="010001" then AW<="1000";
    elsif IE_W="010010" then AW<="1001";
    elsif IE_W="010011" then AW<="1001";
    elsif IE_W="010100" then AW<="1010";
```

```

elsif IE_W="010101" then AW<="1010";
elsif IE_W="010110" then AW<="1010";
elsif IE_W="010111" then AW<="1011";
elsif IE_W="011000" then AW<="1011";
elsif IE_W="011001" then AW<="1100";
elsif IE_W="011010" then AW<="1100";
elsif IE_W="011011" then AW<="1101";
elsif IE_W="011100" then AW<="1101";
elsif IE_W="011101" then AW<="1110";
elsif IE_W="011110" then AW<="1110";
else
    AW<="1111";
end if;

```

```

elsif ITER=7 then
    -- alfa = 0.5
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0010";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0011";
    elsif IE_W="000110" then AW<="0011";
    elsif IE_W="000111" then AW<="0100";
    elsif IE_W="001000" then AW<="0100";
    elsif IE_W="001001" then AW<="0101";
    elsif IE_W="001010" then AW<="0101";
    elsif IE_W="001011" then AW<="0110";
    elsif IE_W="001100" then AW<="0110";
    elsif IE_W="001101" then AW<="0111";
    elsif IE_W="001110" then AW<="0111";
    elsif IE_W="001111" then AW<="1000";
    elsif IE_W="010000" then AW<="1000";
    elsif IE_W="010001" then AW<="1001";
    elsif IE_W="010010" then AW<="1001";
    elsif IE_W="010011" then AW<="1010";
    elsif IE_W="010100" then AW<="1010";
    elsif IE_W="010101" then AW<="1011";
    elsif IE_W="010110" then AW<="1011";
    elsif IE_W="010111" then AW<="1100";
    elsif IE_W="011000" then AW<="1100";
    elsif IE_W="011001" then AW<="1101";
    elsif IE_W="011010" then AW<="1101";
    elsif IE_W="011011" then AW<="1110";
    elsif IE_W="011100" then AW<="1110";
    else
        AW<="1111";
    end if;

```

```

elsif ITER=8 then
    -- alfa = 0.525
    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0010";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0011";
    elsif IE_W="000110" then AW<="0011";
    elsif IE_W="000111" then AW<="0100";
    elsif IE_W="001000" then AW<="0100";
    elsif IE_W="001001" then AW<="0101";
    elsif IE_W="001010" then AW<="0101";
    elsif IE_W="001011" then AW<="0110";
    elsif IE_W="001100" then AW<="0110";
    elsif IE_W="001101" then AW<="0111";
    elsif IE_W="001110" then AW<="0111";
    elsif IE_W="001111" then AW<="1000";
    elsif IE_W="010000" then AW<="1000";
    elsif IE_W="010001" then AW<="1001";
    elsif IE_W="010010" then AW<="1001";

```



```

elsif IE_W="010011" then AW<="1010";
elsif IE_W="010100" then AW<="1011";
elsif IE_W="010101" then AW<="1011";
elsif IE_W="010110" then AW<="1100";
elsif IE_W="010111" then AW<="1100";
elsif IE_W="011000" then AW<="1101";
elsif IE_W="011001" then AW<="1101";
elsif IE_W="011010" then AW<="1110";
elsif IE_W="011011" then AW<="1110";
else
    AW<="1111";
end if;

```

```

elsif ITER=9 then
    -- alfa = 0.575

```

```

    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0010";
    elsif IE_W="000100" then AW<="0010";
    elsif IE_W="000101" then AW<="0011";
    elsif IE_W="000110" then AW<="0011";
    elsif IE_W="000111" then AW<="0100";
    elsif IE_W="001000" then AW<="0101";
    elsif IE_W="001001" then AW<="0101";
    elsif IE_W="001010" then AW<="0110";
    elsif IE_W="001011" then AW<="0110";
    elsif IE_W="001100" then AW<="0111";
    elsif IE_W="001101" then AW<="0111";
    elsif IE_W="001110" then AW<="1000";
    elsif IE_W="001111" then AW<="1001";
    elsif IE_W="010000" then AW<="1001";
    elsif IE_W="010001" then AW<="1010";
    elsif IE_W="010010" then AW<="1010";
    elsif IE_W="010011" then AW<="1011";
    elsif IE_W="010100" then AW<="1100";
    elsif IE_W="010101" then AW<="1100";
    elsif IE_W="010110" then AW<="1101";
    elsif IE_W="010111" then AW<="1101";
    elsif IE_W="011000" then AW<="1110";
    elsif IE_W="011001" then AW<="1110";
    else
        AW<="1111";
    end if;

```

```

elsif ITER=10 then
    -- alfa = 0.65

```

```

    if IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0010";
    elsif IE_W="000100" then AW<="0011";
    elsif IE_W="000101" then AW<="0011";
    elsif IE_W="000110" then AW<="0100";
    elsif IE_W="000111" then AW<="0101";
    elsif IE_W="001000" then AW<="0101";
    elsif IE_W="001001" then AW<="0110";
    elsif IE_W="001010" then AW<="0111";
    elsif IE_W="001011" then AW<="0111";
    elsif IE_W="001100" then AW<="1000";
    elsif IE_W="001101" then AW<="1000";
    elsif IE_W="001110" then AW<="1001";
    elsif IE_W="001111" then AW<="1010";
    elsif IE_W="010000" then AW<="1010";
    elsif IE_W="010001" then AW<="1011";
    elsif IE_W="010010" then AW<="1100";
    elsif IE_W="010011" then AW<="1100";
    elsif IE_W="010100" then AW<="1101";

```

## Turbo decodificador de códigos producto

---

```
    elsif IE_W="010101" then AW<="1110";
    elsif IE_W="010110" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=11 then
    -- alfa = 0.7
    if
        IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0001";
    elsif IE_W="000011" then AW<="0010";
    elsif IE_W="000100" then AW<="0011";
    elsif IE_W="000101" then AW<="0100";
    elsif IE_W="000110" then AW<="0100";
    elsif IE_W="000111" then AW<="0101";
    elsif IE_W="001000" then AW<="0110";
    elsif IE_W="001001" then AW<="0110";
    elsif IE_W="001010" then AW<="0111";
    elsif IE_W="001011" then AW<="1000";
    elsif IE_W="001100" then AW<="1000";
    elsif IE_W="001101" then AW<="1001";
    elsif IE_W="001110" then AW<="1010";
    elsif IE_W="001111" then AW<="1011";
    elsif IE_W="010000" then AW<="1011";
    elsif IE_W="010001" then AW<="1100";
    elsif IE_W="010010" then AW<="1101";
    elsif IE_W="010011" then AW<="1101";
    elsif IE_W="010100" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=12 then
    -- alfa = 0.8
    if
        IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0010";
    elsif IE_W="000011" then AW<="0010";
    elsif IE_W="000100" then AW<="0011";
    elsif IE_W="000101" then AW<="0100";
    elsif IE_W="000110" then AW<="0101";
    elsif IE_W="000111" then AW<="0110";
    elsif IE_W="001000" then AW<="0110";
    elsif IE_W="001001" then AW<="0111";
    elsif IE_W="001010" then AW<="1000";
    elsif IE_W="001011" then AW<="1001";
    elsif IE_W="001100" then AW<="1010";
    elsif IE_W="001101" then AW<="1010";
    elsif IE_W="001110" then AW<="1011";
    elsif IE_W="001111" then AW<="1100";
    elsif IE_W="010000" then AW<="1101";
    elsif IE_W="010001" then AW<="1110";
    elsif IE_W="010010" then AW<="1110";
    else
        AW<="1111";
    end if;

elsif ITER=13 then
    -- alfa = 0.9
    if
        IE_W="000000" then AW<="0000";
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0010";
    elsif IE_W="000011" then AW<="0011";
    elsif IE_W="000100" then AW<="0100";
    elsif IE_W="000101" then AW<="0101";
    elsif IE_W="000110" then AW<="0101";
    elsif IE_W="000111" then AW<="0110";
    elsif IE_W="001000" then AW<="0111";
    elsif IE_W="001001" then AW<="1000";
    elsif IE_W="001010" then AW<="1001";
    elsif IE_W="001011" then AW<="1010";
```

## Turbo decodificador de códigos producto

---

```
    elsif IE_W="001100" then AW<="1011";
    elsif IE_W="001101" then AW<="1100";
    elsif IE_W="001110" then AW<="1101";
    elsif IE_W="001111" then AW<="1110";
    elsif IE_W="010000" then AW<="1110";
    else      AW<="1111";
    end if;

elsif -- ITER=0 or ITER=14 or ITER=15 then
    IE_W="000000" then AW<="0000"; -- alfa = 1
    elsif IE_W="000001" then AW<="0001";
    elsif IE_W="000010" then AW<="0010";
    elsif IE_W="000011" then AW<="0011";
    elsif IE_W="000100" then AW<="0100";
    elsif IE_W="000101" then AW<="0101";
    elsif IE_W="000110" then AW<="0110";
    elsif IE_W="000111" then AW<="0111";
    elsif IE_W="001000" then AW<="1000";
    elsif IE_W="001001" then AW<="1001";
    elsif IE_W="001010" then AW<="1010";
    elsif IE_W="001011" then AW<="1011";
    elsif IE_W="001100" then AW<="1100";
    elsif IE_W="001101" then AW<="1101";
    elsif IE_W="001110" then AW<="1110";
    else      AW<="1111";
    end if;

end process;

-----
--NUEVO VALOR DE R--
-----

process (CLK, RESET, AW, SIGNO_W, Ro_FIA)

--variable K: STD_LOGIC_VECTOR(4 downto 0);
--variable L: STD_LOGIC_VECTOR(4 downto 0);
variable VSUMAWR: STD_LOGIC_VECTOR (4 downto 0);
variable VV: STD_LOGIC;
variable VWMENOSR: STD_LOGIC_VECTOR (3 downto 0);
variable VRMENOSW: STD_LOGIC_VECTOR (3 downto 0);

    ---signal V: STD_LOGIC;
    --signal K: STD_LOGIC_VECTOR (4 downto 0);
    --signal LL: STD_LOGIC_VECTOR (4 downto 0);
    --signal SUMAWR: STD_LOGIC_VECTOR (4 downto 0);
    --signal WMENOSR: STD_LOGIC_VECTOR (3 downto 0);
    --signal RMENOSW: STD_LOGIC_VECTOR (3 downto 0);
    --signal signo: STD_LOGIC;

begin

LL<="00000";
--V<=SIGNO_W XOR Ro_FIA(4);
-- Es la suma de AW con la Fiabilidad de R

VV:= SIGNO_W XOR Ro_FIA(4);
VSUMAWR:=(LL+AW+Ro_FIA(3 downto 0));
VWMENOSR:=(AW-Ro_FIA(3 downto 0));
VRMENOSW:=(Ro_FIA(3 downto 0)-AW);

--SUMAWR<=(LL+AW+Ro_FIA(3 downto 0));
```

## Turbo decodificador de códigos producto

---

```
-- AW menos la Fiabilidad de R
--WMENOSR<=(AW-Ro_FIA(3 downto 0));
-- Fiabilidad de R - AW
--RMENOSW<=(Ro_FIA(3 downto 0)-AW);

    if RESET='1' then
        RPP<="00000";
        LL<="00000";
        V<='0';
        --K:="00000";
        K<="00000";
        VSUMAWR:="00000";
        --SUMAWR<="00000";
        VWMENOSR:="00000";
        VRMENOSW:="00000";
    else --if CLK='0'then -- and CLK'event then
        if VV='0'then
            --Adicion
            K<=(LL+AW+Ro_FIA(3 downto 0));
            signo<= SIGNO_W;
            if VSUMAWR<15 or VSUMAWR=15 then
                if VSUMAWR="00000" then
                    RPP(3 downto 0)<= "0000";
                    RPP(4)<='1';
                else
                    RPP(3 downto 0)<=VSUMAWR(3 downto 0);
                    RPP(4)<=SIGNO_W;
                end if;
            else
                RPP(3 downto 0)<="1111";
                RPP(4)<=SIGNO_W;
            end if;
        elsif AW>Ro_FIA(3 downto 0)then
            -- Sustraccion
            VWMENOSR(3 downto 0):=AW-Ro_FIA(3 downto 0);
            signo<=SIGNO_W;
            if VWMENOSR<15 or VWMENOSR=15 then
                if VWMENOSR="00000" then
                    RPP(3 downto 0)<= "0000";
                    RPP(4)<='1';
                else
                    RPP(3 downto 0)<=VWMENOSR(3 downto 0);
                    RPP(4)<=SIGNO_W;
                end if;
            else
                RPP(3 downto 0)<="1111";
                RPP(4)<=SIGNO_W;
            end if;
        else
            VRMENOSW:=Ro_FIA(3 downto 0)-AW;
            signo<=Ro_FIA(4);
            if VRMENOSW<15 or VRMENOSW=15 then
                if VRMENOSW="00000" then
                    RPP(3 downto 0)<="0000";
                    RPP(4)<='1';
                else
                    RPP(3 downto 0)<=VRMENOSW(3 downto 0);
                    RPP(4)<=SIGNO_W;
                end if;
            else
                RPP(3 downto 0)<="1111";
                RPP(4)<=SIGNO_W;
            end if;
        end if;
    end if;
```

```
        end if;
    end process;

END recepcion_arch;
```

- Código del programa en C++

```
//-----

#pragma hdrstop

//-----
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <iostream.h>
#define K 26
#define NmoinsK 6
#define N 32
#define pi 3.1415926536
#define n 4
#define NbIter 7 // numero de iteraciones
#define q 15
#define m 5
#define nb_Mat 0 // numero de matrices

void GenererDonnees(int mes[]);
int BinaireBase10(int mot[],int max);
//int Signe(int entree[]);
int PolyGen[]={1,0,1,0,0,1}, beta, iter, ph_dec;
float rendement, sab, a[NbIter*2],fac_ech= 0.6;
int donnees, Par_init, Par_calc, Dif_par, RAMZ[q+2][N],C1[q+1],Bit1[q+1][2],Meq[q+1],MotDuCodeEntree,d,c,MemoinsMd[N],x,Nb_M_Tr;
double SigB, alfa, Alpha[]={0.35,0.375,0.4,0.425,0.45,0.475,0.5,0.525,0.575,0.65,0.7,0.7,0.8,0.8,0.9,1.0};
double Pa,Pb;
int Beta[]={11,11,11,11,12,12,13,13,14,15,16,17,19,21,22,23}, fer[NbIter*2];
int M_Em[N][N], M_R[N][N], NBI, sgnRprime[N], Rprime[N],Compos[16][2], S1[q+1][m], GF[N][m], ROM[N], pgGF[]={1,0,1,0,0,1},
M_Rprime[N][N];
void CodageBCH32(int mes[],int mot[]);
int NBIFAD[NbIter*2], R_Em[N], Wj[N], M_Est[N][N];
void Cod_Mat();
void SigmaDuBruit();
void BruitGaussien();
double Arrondi(double Nb);
void Transmission();
int Compar1(int a,int b);
void ComposantesNonFiabiles();
void DecodageVecteurDonnees();
FILE *fich1,*fich2,*fich3,*fich4,*fich5,*fich6,*fich7,*fich8,*fich9,*fich10,*fich28;
int Signe(int entree);
void Syndrome(int mot[],int synd[]);
void ModificateurDeSyndrome();
void GenererROM();
void GenererGF();
void DecodageAlgebriqueDesZ1();
void Met10(int v,int valpar,int indice);
void Met11(int v0[],int v[],int valpar,int indice);
void Met12(int v0[],int v1[],int v[],int valpar,int indice);
void Met13(int v0[],int v1[],int v2[],int v[],int valpar,int indice);
void CalculDesDistances1();
void DecisionEtConcurrent();
void Dec_Mat();
```

```

void conversion_bin(int entier,int mot_bin[5]);
float randomismo(int mi, int ma);

int main(int argc, char* argv[])
{int i,j,k;
fich1 = fopen("entradas_enteras", "wt");
fich2 = fopen("demi_iter_entera", "wt");
fich3 = fopen("entrada", "wt");
fich4 = fopen("demi_iter", "wt");
fich5 = fopen("syndrome", "wr");
fich6 = fopen("Rprime", "wt");
fich7 = fopen("f7", "wt");
fich8 = fopen("f8", "wt");
fich9 = fopen("f9", "wt");
fich10 = fopen("f10", "wt");
fich28 = fopen("W_aW", "wt");
donnees=K*K;
srand(time(0));
GenererGF();
GenererROM();
/*for (i=0;i<N;i++)
printf("GF[%d]=%d%d%d%d%d \n",i,GF[i][4],GF[i][3],GF[i][2],GF[i][1],GF[i][0]);
for (i=0;i<N;i++)
printf("ROM[%d]=%d \n",i,ROM[i]);*/

for (beta=-2;beta<6;beta+=11) for (alfa=0.35;alfa<=0.9;alfa+=10.1)
for (x=2;x<=10;x+=11) for (fac_ech=0.6;fac_ech<=0.955;fac_ech+=10.05)
{
printf("Alpha=");
for (iter=0;iter<2*Nblter;iter++) Alpha[iter]=alfa;Alpha[8]=0.4;
Alpha[9]=0.5;Alpha[10]=0.6;Alpha[11]=0.7;Alpha[12]=0.85;Alpha[13]=1.0;
for (iter=0;iter<2*Nblter;iter++) printf("%.2f ",Alpha[iter]);printf("\n");
printf("Facteur d'echelle=%.2f\n",fac_ech);
printf("Beta tient compte de MF%d,MF%d et MF%d. Gamma est utilise(Fiab.+2).\n",x,x+1,x+2,x+3);
for (sab=3;sab<=200;sab+=200)
{
SigmaDuBruit();
Nb_M_Tr=0;NBI=0;
for (iter=0;iter<2*Nblter;iter++) {NBIFAD[iter]=0;fer[iter]=0;}
while (Nb_M_Tr<=nb_Mat)
{
Nb_M_Tr+=1;
Cod_Mat();
Transmission();
//M_R[0][3]=floor(M_R[0][3]/-8);
//M_R[0][4]=floor(M_R[0][4]/6);
//M_R[0][5]=floor(M_R[0][5]/4);
printf("\n\n");
for (j=0;j<N;j++) {
Rprime[j]=M_R[0][j];
printf("%d", Rprime[j]);
}
Dec_Mat();
if ((Nb_M_Tr==nb_Mat+1)||((Nb_M_Tr%1000)==0))
{
printf("Nb de Matrices traitees a %.2f dB = %d\n",sab,Nb_M_Tr);
printf("TEB avant decodage: %.1e (%d)\n",
((float)NBI/((float)Nb_M_Tr*(float)N*(float)K)),NBI);
printf("TEB apres 1 iteration =%.1e (%d)",
(((float)NBIFAD[0]/(float)Nb_M_Tr)/(float)donnees),NBIFAD[0]);
printf(" [FER=%.1e(%d)]\n",
((float)fer[0]/(float)(Nb_M_Tr)),fer[0]);
for (iter=3;iter<2*Nblter;iter+=2)
{
printf("TEB apres %d iterations=%.1e (%d)",iter/2+1,
(((float)NBIFAD[iter]/(float)Nb_M_Tr)/(float)donnees),

```

```
        NBIFAD[iter]);
    printf(" [FER=%.1e(%d)]\n",
        ((float)fer[iter]/(float)Nb_M_Tr).fer[iter]);
    }
    printf("\n");
    }
}
}
fclose(fich1);
fclose(fich2);
fclose(fich3);
fclose(fich4);
fclose(fich6);
fclose(fich7);
fclose(fich8);
fclose(fich9);
fclose(fich10);
fclose(fich28);

for (i=0; i<N; i++){
    for (j=0; j<N; j++){
        printf("[%d]", M_Rprime[i][j]);
    }
    printf("\n");
}
getch();
}
//-----
//randomismo
float randomismo(int mi, int ma)
{
    float max, min, range, a, rmax, to_be_displayed;
    //srand(time(0));
    max=(float)ma;
    min=(float)mi;
    range = max - min;
    rmax = RAND_MAX;
    a = rand();
    to_be_displayed = (a / (rmax / range)) + min;
    //cout<<"Your random number is : "<<to_be_displayed;
    return to_be_displayed;
}

//-----
void Dec_Mat()
{
    int i,j,k,compt,comp,count,synd;
    double w;
    int mot_bin[N][N][5];
    for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        M_Rprime[i][j]=M_R[i][j];
    for (j=0; j<N; j++)
        for (i=0; i<N; i++){
            fprintf (fich1, "R[%d]=%d\n", i, M_R[i][j]);
            conversion_bin(M_R[i][j], mot_bin[i][j]);
            fprintf (fich3, " %d%d%d%d%d\n", mot_bin[i][j][0],
                mot_bin[i][j][4],
                mot_bin[i][j][3],
                mot_bin[i][j][2],
                mot_bin[i][j][1]);}

    for (iter=0; iter<NbIter; iter++)
    {
        ph_dec=2*iter; a[ph_dec]=Alpha[ph_dec]; compt=0; comp=0;
```

```

for (j=0;j<N;j++)
{
for (i=0;i<N;i++) {Rprime[i]=M_Rprime[i][j];}

for (i=0;i<N;i++) R_Em[i]=M_Em[i][j];
DecodageVecteurDonnees();
Rprime[N-1]=Rprime[N-1]*(2*Par_init-1)*(2*sgnRprime[N-1]-1);
fprintf (fich28, "j=%d\n",j);
for (i=0;i<N;i++)
{
Wj[i]=(2*RAMZ[d][i]-1)*MmoinsMd[i]-Rprime[i];
w=a[ph_dec]*Wj[i];if ((w-floor(w))>=0.5) w=w+1.0;
if (w>15.0) w=15.0;if (w<-15.0) w=-15.0;
Rprime[i]=M_R[i][j]+(int)floor(w);
if (Rprime[i]>15) Rprime[i]=15;if (Rprime[i]<-15) Rprime[i]=-15;
M_Rprime[i][j]=Rprime[i];M_Est[i][j]=RAMZ[d][i];

fprintf (fich2, "Rprime_plus[%d]=%d\n ",i,Rprime[i]);
conversion_bin(M_Rprime[i][j],mot_bin[i][j]);
fprintf (fich4, " %d%d%d%d%d\n",mot_bin[i][j][0],
mot_bin[i][j][4],
mot_bin[i][j][3],
mot_bin[i][j][2],
mot_bin[i][j][1]);
fprintf (fich28, "Wj[%d]=%d fiab[%d][%d]=%d ",i,Wj[i],i,j,(2*RAMZ[d][i]-1)*MmoinsMd[i]);
fprintf (fich28, " aW=%2f ",w);
fprintf (fich28, "M_R[%d][%d]=%d ",i,j,M_R[i][j]);
fprintf (fich28, " R'+[%d]=%d\n",i,Rprime[i]);
if ((i<K)&(j<K))
{
if (M_Em[i][j]^RAMZ[d][i]==1){compt+=1;comp=1;}
}
}
}
for[ph_dec]+=comp;NBIFAD[ph_dec]+=compt;
comp=0;comp=0;count=0;ph_dec=2*iter+1;a[ph_dec]=Alpha[ph_dec];
for (i=0;i<N;i++)
{
for (j=0;j<N;j++) {Rprime[j]=M_Rprime[i][j];}
*
fprintf(fich1, "Rprime[%d]=%d\n",j,Rprime[j]);
conversion_bin(M_Rprime[i][j],mot_bin[i][j]);
fprintf (fich3, " %d%d%d%d%d\n",mot_bin[i][j][0],
mot_bin[i][j][4],
mot_bin[i][j][3],
mot_bin[i][j][2],
mot_bin[i][j][1]);}*/

for (j=0;j<N;j++) R_Em[j]=M_Em[i][j];
for (j=0;j<N;j++) sgnRprime[j]=Signe(Rprime[j]);
Syndrome(sgnRprime,S1[0]);
synd=S1[0][0];for (j=1;j<m;j++) synd=synd||S1[0][j];
if (synd==0) count+=1;
DecodageVecteurDonnees();
Rprime[N-1]=Rprime[N-1]*(2*Par_init-1)*(2*sgnRprime[N-1]-1);

for (j=0;j<N;j++)
{
Wj[j]=(2*RAMZ[d][j]-1)*MmoinsMd[j]-Rprime[j];
w=a[ph_dec]*Wj[j];

if ((w-floor(w))>=0.5) w=w+1.0;
if (w>15.0) w=15.0;if (w<-15.0) w=-15.0;

Rprime[j]=M_R[i][j]+(int)floor(w);
if (Rprime[j]>15) Rprime[j]=15;if (Rprime[j]<-15) Rprime[j]=-15;

```



```
M_Rprime[i][j]=Rprime[j];M_Est[i][j]=RAMZ[d][j];

/* fprintf (fich2, "Rprime_plus[%d]=%d\n ",j,Rprime[j]);
conversion_bin(M_Rprime[i][j],mot_bin[i][j]);
fprintf (fich4, " %d%d%d%d%d\n",mot_bin[i][j][0],
            mot_bin[i][j][4],
            mot_bin[i][j][3],
            mot_bin[i][j][2],
            mot_bin[i][j][1]);*/

if ((i<K)&&(j<K))
{
    if (M_Em[i][j]^RAMZ[d][j]==1){compt+=1;comp=1;}
}
}

if (count==N)
{
    for(i=ph_dec;i<2*Nbiter;i++) {fer[i]+=comp;NBIFAD[i]+=compt;}
    iter=Nbiter;
}
else {fer[ph_dec]+=comp;NBIFAD[ph_dec]+=compt;}
}
return;
}
/-----

void GenererDonnees(int mes[])
{
    int Nb;
    int i;
    //srand(time(NULL));

    for( i=0; i<K; i++)
    {
        Nb= rand()%2;
        mes[i]= Nb;

        //printf ("%d", mes[i]);
    }
    printf("\n");
    //getch();

    return;
}

void CodageBCH32(int mes[],int mot[])
{
    int V[NmoinsK],i,j,retour;
    for(i=K;i<N;i++)
        mot[i]=0;
    for (i=0;i<NmoinsK;i++)
        V[i]=0;
    for(i=0;i<K;i++)
        mot[i]=mes[i];
    for(i=0;i<N-1;i++)
    {
        retour=V[NmoinsK-1];
        for(j=NmoinsK-1;j>1;j--)
            V[j]=V[j-1]^PolyGen[j-1]*retour;
        V[1]=mot[i]^PolyGen[0]*retour;
        V[0]=V[0]^mot[i];
    }
}
```

```
    }

    for (i=1;i<NmoinsK;i++) V[0]=V[0]^V[i];

    for (i=1;i<NmoinsK;i++) mot[N-1-i]=V[i];

    mot[N-1]=V[0];

    return;
}

void Cod_Mat ()
{
    int Message[K],MotDeCode[N],i,j;
    for (i=0;i<K;i++)
    {
        GenererDonnees(Message);
        CodageBCH32(Message,MotDeCode);
        for (j=0;j<N;j++)
            M_Em[i][j]=MotDeCode[j];
    }
    for (j=0;j<N;j++)
    {
        for (i=0;i<K;i++) Message[i]=M_Em[i][j];
        CodageBCH32(Message,MotDeCode);

        for (i=K;i<N;i++)
        {
            M_Em[i][j]=MotDeCode[i];
        }
    }

    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            printf("%d", M_Em[i][j]);
        }
        printf("\n");
    }

    return;
}

int BinaireBase10(int mot[], int max)
{
    int i, Nb=0;
    for (i=0;i<=max-1;i++) Nb=(Nb*2)+mot[i];
    return(Nb);
}

void SigmaDuBruit()
{
    double interm;int nn;
    nn=N*N;rendement=((float)donnees/(float)nn);
    printf("Rendement=%3f (%d lignes de %d data):n",rendement,K,K);
    interm=-0.1*((double)sab+10.0*log10(rendement));
    SigB=sqrt(pow(10.0,interm)/2);printf("Sigma= %f\n",SigB);
    return;
}

void BruitGaussien()
{
    double x,y;
    x=sqrt(-2.0*log((double)randomismo(0,1)));
    y=(double)randomismo(0,1);
    Pa=x*sin(2*pi*y);
    Pb=x*cos(2*pi*y);
}
```

```
//printf("pa=%f, pb= %f", Pa, Pb);
return;
}

double Arrondi(double Nb)
{
double PartieEntiere;
PartieEntiere = floor(Nb);
if ( (Nb-PartieEntiere)>=0.5) PartieEntiere=PartieEntiere+1.0;
return(PartieEntiere);
}

void Transmission()
{
double x,y,a,b,max=pow(2.0,(float)n)-1.0;
int i,j,compt=0,signe,redondance,module;
for(i=0;i<N;i++)
for(j=0;j<N;j=j+2)
{
BruitGaussien();
a=Pa;
b=Pb;
x=(double)M_Em[i][j];
x=((2*x)-1)+(SigB*a);
x=x*(fac_ech*max);
if (x<0) signe=0;
else signe=1;
x=Arrondi(fabs(x));
if (x>max) x=max;
M_R[i][j]=(int)x*(2*signe-1);
if (j+1<N)
{
y=(double)M_Em[i][j+1];y=((2*y)-1)+(SigB*b);y=y*(fac_ech*max);
if (y<0) signe=0;else signe=1;
y=Arrondi(fabs(y));if (y>max) y=max;
M_R[i][j+1]=(int)y*(2*signe-1);
}
}
compt=0;
for (i=0;i<K;i++) for (j=0;j<N;j++)
{
if (M_R[i][j]>=0) signe=1;
else signe=0;
compt+=M_Em[i][j]^signe;
}
//aquí se cambian valores en la matriz con ruido
NBI+=compt;
for (i=0; i<N; i++){
for (j=0; j<N; j++){
printf("[%d]", M_R[i][j]);
}
printf("\n");
}
//getch(); // aquí muestra la matriz con ruido
return;
}

int Compar1(int a,int b)
{
int sortie=0;
if (a<=b) sortie=1;
return(sortie);
}

void DecodageVecteurDonnees()
{
```

```
int i,j;
for (i=0;i<N;i++)
    sgnRprime[i]=Signe(Rprime[i]);
Syndrome(sgnRprime,S1[0]);
Par_init=sgnRprime[N-1];
Par_calc=sgnRprime[0];
for (i=1;j<N-1;i++)
    Par_calc=Par_calc^sgnRprime[i];
Dif_par=Par_calc^Par_init;
Rprime[N-1]=Rprime[N-1]*(1-2*Dif_par);
sgnRprime[N-1]=Par_calc;
ComposantesNonFiables();
//printf("Rprima = %d y el signo de Rprima es %d\n", Rprime[N-1], sgnRprime[N-1]);
//getch();
ModificateurDeSyndrome();
DecodageAlgebriqueDesZ1();
CalculDesDistances1();
DecisionEtConcurrent();
return;
}

void Syndrome(int mot[],int synd[])
{
int i,j,intermediaire[m];
for(i=0;i<m;i++) synd[i]=0;
for(i=0;i<N-1;i++)
{
intermediaire[0]=mot[i]^synd[4];
intermediaire[1]=synd[0];
intermediaire[2]=synd[1]^synd[4];
for(j=3;j<m;j++) intermediaire[j]=synd[j-1];
for(j=0;j<m;j++) synd[j]=intermediaire[j];
}
//printf("\n");
//printf("syndrome=%d%d%d%d%d\n",synd[4],synd[3],synd[2],synd[1],synd[0]);
//getch();
return;
}

void ComposantesNonFiables()
{
int i;
int j;
int k;
int command[12];
int modul_rp;
for (i=0;i<5;i++)
{
Compos[i][0]=N-1;
Compos[i][1]=16;
}
for(i=0;i<N-1;i++)
{
modul_rp=(2*sgnRprime[i]-1)*Rprime[i];
for (j=0;j<5;j++) command[j]=Compar1(modul_rp,Compos[j][1]);
for(j=4;j>0;j--)
{
if ((command[j-1]^command[j])==0)
{
for (k=0;k<2;k++)
Compos[j][k]=command[j]*Compos[j-1][k+!command[j]*Compos[j][k];
}
else {Compos[j][0]=i;Compos[j][1]=modul_rp;}
}
if (command[0]==1) {Compos[0][0]=i;Compos[0][1]=modul_rp;}
}
}
```

```
for (i=0;i<5;i++)
#printf("position[%d]=%d:valeur=%d \n",i,Compos[i][0],Compos[i][1]);
return;
#getch();
}
```

```
int Signe(int entree)
{
int sortie;
if (entree<0)
sortie=0;
else sortie=1;
return(sortie);
}
```

```
void ModificateurDeSyndrome()
{
int i,j,pos[m];
for (i=1;i<=q;i++)
for (j=0;j<m;j++)
S1[i][j]=S1[0][j];

for (j=0;j<m;j++)
pos[j]=GF[Compos[0][0]][j];
for (j=0;j<m;j++)
{
S1[1][j]=S1[1][j]^pos[j];
for (i=6;i<=q;i++)
S1[i][j]=S1[i][j]^pos[j];
}

for (j=0;j<m;j++)
pos[j]=GF[Compos[1][0]][j];
for (j=0;j<m;j++)
{
S1[2][j]=S1[2][j]^pos[j];
S1[6][j]=S1[6][j]^pos[j];
for (i=10;i<=12;i++)
S1[i][j]=S1[i][j]^pos[j];
}

for (j=0;j<m;j++)
pos[j]=GF[Compos[2][0]][j];
for (j=0;j<m;j++)
{
S1[3][j]=S1[3][j]^pos[j];
S1[7][j]=S1[7][j]^pos[j];
S1[10][j]=S1[10][j]^pos[j];
S1[13][j]=S1[13][j]^pos[j];
S1[14][j]=S1[14][j]^pos[j];
}

for (j=0;j<m;j++)
pos[j]=GF[Compos[3][0]][j];
for (j=0;j<m;j++)
{
S1[4][j]=S1[4][j]^pos[j];
S1[8][j]=S1[8][j]^pos[j];
for (i=11;i<=q;i=i+2)
S1[i][j]=S1[i][j]^pos[j];
}

for (j=0;j<m;j++)
pos[j]=GF[Compos[4][0]][j];
for (j=0;j<m;j++)
{
S1[5][j]=S1[5][j]^pos[j];
```

```
    S1[9][j]=S1[9][j]^pos[j];
    S1[12][j]=S1[12][j]^pos[j];
    S1[14][j]=S1[14][j]^pos[j];
    S1[15][j]=S1[15][j]^pos[j];
}
return;
}
/-----
void GenererGF()
{
int i,j;
GF[N-2][0]=1;
for (j=1;j<=m;j++) GF[N-2][j]=0;
for (i=N-3;i>=0;i--)
{
    GF[i][0]=GF[i+1][m-1];
    for (j=m-1;j>0;j--) GF[i][j]=GF[i+1][j-1]^GF[i][0]*pgGF[j];
}
for (j=0;j<=m;j++) GF[N-1][j]=0;

return;
}
/*-----*/
void GenererROM()
{
int i;
for(i=0;i<N;i++) ROM[BinaireBase10(GF[i].m)]=i;
return;
}
/-----
void DecodageAlgebriqueDesZ1()
{
int i,j,pos,val;
for (i=0;i<=q;i++)
    for (j=0;j<N;j++)
        RAMZ[i][j]=sgnRprime[j];
for (j=0;j<N;j++)
    RAMZ[16][j]=0;
pos=Compos[0][0];
val=!RAMZ[1][pos];
RAMZ[1][pos]=val;
for (i=6;i<=q;i++)
    RAMZ[i][pos]=val;
pos=Compos[1][0];
val=!RAMZ[2][pos];
RAMZ[2][pos]=val;
RAMZ[6][pos]=val;
for (i=10;i<=12;i++)
    RAMZ[i][pos]=val;
pos=Compos[2][0];
val=!RAMZ[3][pos];
RAMZ[3][pos]=val;
RAMZ[7][pos]=val;
RAMZ[10][pos]=val;
RAMZ[13][pos]=val;
RAMZ[14][pos]=val;
pos=Compos[3][0];
val=!RAMZ[4][pos];
RAMZ[4][pos]=val;
RAMZ[8][pos]=val;
for (i=11;i<=q;i+=2)
    RAMZ[i][pos]=val;
pos=Compos[4][0];
val=!RAMZ[5][pos];
RAMZ[5][pos]=val;
RAMZ[9][pos]=val;
}
```

```

RAMZ[12][pos]=val;
RAMZ[14][pos]=val;
RAMZ[15][pos]=val;
for (i=0;i<=q;i++)
{Ct[i]=S1[i][0];
for(j=1;j<m;j++) Ct[i]=Ct[i]||S1[i][j];
Bit1[i][0]=ROM[BinaireBase10(S1[i],m)];
Bit1[i][1]=(2*sgnRprime[Bit1[i][0]]-1)*Rprime[Bit1[i][0]];
if (Bit1[i][0]!=N-1) RAMZ[i][Bit1[i][0]]!=RAMZ[i][Bit1[i][0]];
}
return;
}
/*-----*/
void Met10(int v,int valpar,int indice)
{
int V,ValPar;
V=v*Ct[indice];ValPar=valpar*(Ct[indice]^Dif_par);
Meq[indice]=V+ValPar;
RAMZ[indice][N-1]=(RAMZ[indice][N-1]^Ct[indice]);
return;
}
/*-----*/
void Met11(int v0[],int v[],int valpar,int indice)
{
int V0,V,ValPar,eg0;
eg0=(v0[0])!=(v[0]);V0=v0[1]*eg0;V=v[1]*eg0*Ct[indice];
ValPar=valpar*(Ct[indice]^Dif_par);Meq[indice]=V+V0+ValPar;
RAMZ[indice][N-1]=(RAMZ[indice][N-1]^Ct[indice]);
return;
}
/*-----*/
void Met12(int v0[],int v1[],int v[],int valpar,int indice)
{
int V0,V1,V,ValPar,eg0,eg1;
eg0=(v0[0])!=(v[0]);eg1=(v1[0])!=(v[0]);
V0=v0[1]*eg0;V1=v1[1]*eg1;V=v[1]*eg0*eg1*Ct[indice];
ValPar=valpar*(Ct[indice]^Dif_par);Meq[indice]=V0+V1+V+ValPar;
RAMZ[indice][N-1]=(RAMZ[indice][N-1]^Ct[indice]);
return;
}
/*-----*/
void Met13(int v0[],int v1[],int v2[],int v[],int valpar,int indice)
{
int V0,V1,V2,V,ValPar,eg0,eg1,eg2;
eg0=(v0[0])!=(v[0]);eg1=(v1[0])!=(v[0]);eg2=(v2[0])!=(v[0]);
V0=v0[1]*eg0;V1=v1[1]*eg1;V2=v2[1]*eg2;
V=v[1]*eg0*eg1*eg2*Ct[indice];ValPar=valpar*(Ct[indice]^Dif_par);
Meq[indice]=V0+V1+V2+V+ValPar;
RAMZ[indice][N-1]=(RAMZ[indice][N-1]^Ct[indice]);
return;
}
/*-----*/
void CalculDesDistances1()
{
int i,j,mod_parite;
mod_parite=(2*sgnRprime[N-1]-1)*Rprime[N-1];
Met10(Bit1[0][1],mod_parite,0);
for (i=0;i<5;i++) Met11(Compos[i],Bit1[i+1],mod_parite,i+1);
for (i=1;i<5;i++) Met12(Compos[0],Compos[i],Bit1[i+5],mod_parite,i+5);
Met13(Compos[0],Compos[1],Compos[2],Bit1[10],mod_parite,10);
Met13(Compos[0],Compos[1],Compos[3],Bit1[11],mod_parite,11);
Met13(Compos[0],Compos[1],Compos[4],Bit1[12],mod_parite,12);
Met13(Compos[0],Compos[2],Compos[3],Bit1[13],mod_parite,13);
Met13(Compos[0],Compos[2],Compos[4],Bit1[14],mod_parite,14);
Met13(Compos[0],Compos[3],Compos[4],Bit1[15],mod_parite,15);
return;
}

```

```

}
void DecisionEtConcurrent()
{
int i,j,k,command[4],dM1,dM2,dM3,Mc[2][4],c_prim,c_tier,non_conc;
MotDuCodeEntree=S1[0][0];
for (j=1;j<m;j++) MotDuCodeEntree=MotDuCodeEntree|S1[0][j];
Mc[0][0]=Meq[q];Mc[1][0]=q;
for (j=1;j<=3;j++) Mc[0][j]=63;
for (j=1;j<=3;j++) Mc[1][j]=16;
for (j=q-1;j>=0;j--)
{
/* non_conc=RAMZ[Mc[1][0]][0]^RAMZ[j][0];
for (i=0;i<N-1;i++) non_conc=non_conc|(RAMZ[j][i]^RAMZ[Mc[1][0]][i]);
non_conc=1;for (i=3;i>0;i--) if (Meq[j]==Mc[0][i]) non_conc=0;
if (non_conc==1)
{
*/
for (i=0;i<4;i++) command[i]=Compar1(Meq[j],Mc[0][i]);
for (i=3;i>0;i--)
{
if ((command[i-1]^command[i])==0)
{
for (k=0;k<2;k++) Mc[k][i]=command[i]*Mc[k][i-1]+!command[i]*Mc[k][i];
}
else {Mc[1][i]=j;Mc[0][i]=Meq[j];}
}
if (command[0]==1) {Mc[1][0]=j;Mc[0][0]=Meq[j];}
/* }*/
}
d=Mc[1][0];c=Mc[1][1];c_prim=Mc[1][2];c_tier=Mc[1][3];
dM1=Mc[0][1]-Mc[0][0];dM2=Mc[0][2]-Mc[0][0];dM3=Mc[0][3]-Mc[0][0];
for (i=0;i<N;i++)
{
McmoinsMd[i]=Compos[x+1][1]+Compos[x][1]+Compos[x+2][1]
+(2*sgnRprime[i-1])*Rprime[i]-Mc[0][0];
if (McmoinsMd[i]<0) McmoinsMd[i]=-McmoinsMd[i];
if (RAMZ[c_tier][i]^RAMZ[d][i]!=0) McmoinsMd[i]=dM3;
if (RAMZ[c_prim][i]^RAMZ[d][i]!=0) McmoinsMd[i]=dM2;
if (RAMZ[c][i]^RAMZ[d][i]!=0) McmoinsMd[i]=dM1;
if (MotDuCodeEntree==0) McmoinsMd[i]=McmoinsMd[i]+2;
}
/* for (i=0;i<q+1;i++) printf("%d ",Meq[i]);printf("\n");*/
return;
}
/*-----*/

void conversion_bin(int entier,int mot_bin[5])
{
int max=4;
int i;
int Nb;
Nb=abs(entier);
if (entier>=0) mot_bin[0]=1;
else mot_bin[0]=0;
for (i=1;i<=max;i++) {mot_bin[i]=Nb%2;Nb=(Nb-mot_bin[i])/2;}
}

```