UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ingeniería

SIMULACION DE CODIGOS CICLICOS CORRECTORES DE ERRORES Y ANALISIS DE SUS CARACTERISTICAS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A N

ISMAEL MAGAÑA MENDOZA

ROBERT HENRY MORELOS-ZARAGOZA ASCANIO

Director: Dr. Francisco J. García Ugalde

México, D. F. **1985**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

Página

PRESENTACION

PRESENTACION

Algunos desarrollos tecnológicos importantes han contribuido al rápido crecimiento del campo de los códigos correctores de errores en las últimas décadas.

El costo de los dispositivos electrónicos de estado sólido se ha decrementado casi tan dramáticamente como su tamaño. Esto ha estimulado el desarrollo de computadoras digitales y dispositivos periféricos, y esto a su vez, ha causado un incremento excepcional en el volumen de datos comunicados entre tales máquinas.

La intolerancia de los sistemas de computadoras a los errores y, en algunos casos, la naturaleza inherentemente crítica de los datos, demandan la facilidad, ya sea de errores nulos, o de algún tipo de código detector y/o corrector de errores en los dispositivos terminales.

Otros factores importantes que han contribuído en el desarrollo de la teoría y aplicaciones de los códigos correctores de errores son: el acelerado crecimiento de las comunicaciones digitales; la necesidad de obtener una transmisión confiable de señales analógicas codificadas digitalmente; la creciente digitalización de modems y equipo de comunicaciones; y la cada vez mayor complejidad de las necesidades de los usuarios y de la sociedad en general.

Desde la aparición de la teoría de la codificación, por Shannon (1948), se han producido importantes avances en el campo de los códigos correctores de errores. Se han diseñado diversos procedimientos de decodificación que pueden implementarse con una cantidad modesta de dispositivos electrónicos (hardware). Debido a lo anterior, el uso de códigos correctores de errores es bastante práctico en los sistemas de comunicación, y se espera que en un futuro cercano dichos códigos sean utilizados más ampliamente.

La teoría de la codificación tiene una historia única entre las disciplinas de la Ingeniería: los teoremas aparecieron primero, las aplicaciones prácticas después. Existen dos causas principales de lo anterior.

Primero, los teoremas de codificación de Shannon demostraron que, dentro de una clase mayor de familias de códigos, existían algunos esquemas, la mayoría implementados actualmente, que podrían proporcionar tasas de error arbitrariamente bajas para cierta tasa de información que no sobrepasara un valor crítico llamado: Capacidad del Canal. Estos teoremas no daban pista alguna para la construcción de dichos esquemas, y la búsqueda de técnicas de codificación capaces de lograr por lo menos una aproximación remota a la capacidad teórica fué muy difícil.

Segundo, los canales de interés práctico (líneas telefónicas, cables, microondas, enlaces troposféricos, radio HF, etc.) han probado no parecerse a la regularidad estadística asumida en la prueba de los teoremas. De hecho, la mayoría de dichos teoremas se basa en la suposición de la existencia de una independencia estadística en el ruido que afecta a cada símbolo que se transmite, mientras que en los canales citados el ruido tiende a manifestarse en paquetes que afectan muchos símbolos.

En la actualidad, el problema de encontrar esquemas funcionales de decodificación, se ha convertido en el de encontrar decodificadores de complejidad razonable. La solución más popular ha sido el uso de estructuras del álgebra, que generan clases de códigos buenos y decodificables. Otra solución ha sido el uso de circuitos lineales secuenciales, para generar una clase de códigos llamados secuenciales o convolucionales.

Este trabajo comprende el estudio de algunos esquemas de codificación y decodificación de códigos cíclicos, correctores y/o detectores de errores, buscando en lo posible su aplicación práctica en sistemas de comunicaciones; y la implementación de una familia particular de códigos cíclicos no binarios, usando un sistema microprogramado. El trabajo se dividió en siete capítulos cuyo contenido se resume a continuación.

En el capítulo 1 se describen de manera general los diversos tipos de códigos que existen, considerando especialmente los códigos cíclicos de bloques.

El capítulo 2 describe los códigos de Hamming, analizados tanto desde el enfoque de códigos lineales (representados por matrices), como de códigos cíclicos (representados por polinomios). Se muestran los esquemas de codifica

ción y de decodificación y la modificación que sufren estos al acortar o exten der la longitud del código. Finalmente, se presentan resultados de la simula ción de dichos códigos mediante programas en lenguaje FORTRAN, implementados en una minicomputadora PDP 11/40.

En el capítulo 3 se presentan los códigos de Bose-Chaudhuri-Hoequenghem (B.C.H.) mostrando sus propiedades principales y sus esquemas de codificación y decodificación más usados. Se describe el efecto del acortamiento de la lon gitud del código, y los resultados de la simulación en computadora.

El capítulo 4 trata de los códigos de Reed-Solomon, mencionándose las ca racterísticas de su polinomio generador y sus esquemas de codificación y deco dificación más usuales. También se describen algunos métodos alternativos de decodificación y se menciona el efecto del acortamiento y extensión de la lon gitud del código en el funcionamiento del esquema. Se presentan además resul tados de la simulación digital.

En el capítulo 5 se analiza el comportamiento de las tres familias de có digos cíclicos estudiadas (Hamming, B.C.H. y Reed-Solomon). En primer lugar se presentan las expresiones matemáticas de las probabilidades de los diferen tes eventos que pueden ocurrir en el decodificador: Corrección adecuada, no detección y decodificación incorrecta. Se grafican estas expresiones para ciertos casos de interés, y se muestran gráficas comparativas. Mas adelante, se analiza la complejidad de la implementación.

El capítulo 6 presenta la implementación de un código de Reed-Solomon parametrable (i.e. de longitud nominal, acortamiento y capacidad de correc ción variables), utilizando un sistema microprogramado. Primero, se estudian los principios básicos del funcionamiento de la tarjeta MEX68BKECB, que con tiene un microprocesador de 16 bits, en la cual se hizo la implementación. Se presentan las consideraciones teóricas y prácticas del diseño del esquema, y se reportan y analizan los resultados de las pruebas hechas al esquema.

En el capítulo 7 se presentan las conclusiones del trabajo y algunas ob servaciones sobre el trabajo a desarrollar en el futuro sobre los códigos cí clicos, particularmente el caso de los códigos no binarios.

En el apéndice A se hace una exposición general de los principios matemáticos de la teoría de códigos cíclicos de bloques.

El apéndice B es un listado del programa implementado en el microprocesador MC68000, en lenguaje ensamblador.

El apéndice C trata de una familia especial de los códigos cíclicos: Los códigos de Fire. Se analiza su capacidad de detección y/o corrección de paquetes de bits en error, sus esquemas de codificación y decodificación y los resultados de la simulación digital.

En el apéndice D se muestran los diagramas de flujo de los programas de simulación, en lenguaje FORTRAN, de las tres familias de códigos que se analizaron.

Finalmente, en el apéndice E se trata la aplicación de un código R.S. (62,56,7) para la protección de muestras de voz, almacenadas en disco en la minicomputadora PDP 11/40 del laboratorio CAD de la División de Estudios de Posgrado de la Facultad de Ingeniería.

CAPITULO 1

TIPOS DE CODIGOS

CAPITULO 1

TIPOS DE CODIGOS

1.1 INTRODUCCION

1.1.1 SISTEMAS DE COMUNICACION. [1,31]

El espectro electromagnético, empleado como soporte en la transmisión de información, de la mayoría de los sistemas de comunicación está limitado en su ancho de banda. El funcionamiento de cualquier sistema de comunicación está limitado por este factor y por la presencia de ruido.

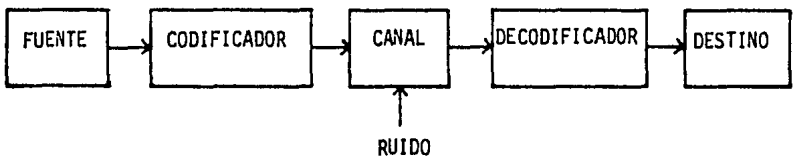


Figura 1.1 Diagrama de bloques de un sistema de comunicaciones.

Considérese el diagrama de bloques de un sistema de comunicación como el mostrado en la figura 1.1. El canal es un medio físico que une a la fuente con el destino. Ejemplos comunes son la línea telefónica, un patrón de propagación de energía electromagnética, o un cable en una computadora digital. Es común también considerar a un medio de almacenamiento de información como un canal. El codificador es un dispositivo que transforma la salida de la fuente en una forma de señal que pueda ser transmitida por el canal, de acuerdo a las limitaciones del mismo (ancho de banda, ruido, etc.). El decodificador por su parte convierte la salida del canal en una forma que pueda ser interpretada por el destino.

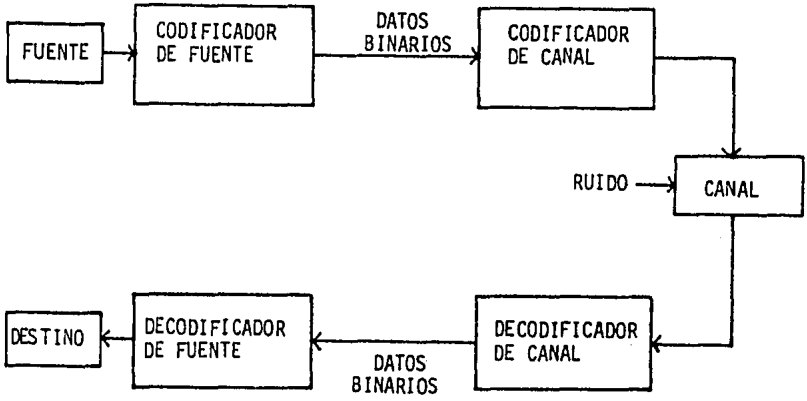


Figura 1.2 Diagrama de un canal de comunicación dividido en codificación de canal y codificación de fuente.

Un diagrama particularmente útil es el que se muestra en la figura 1.2, en el que se ha dividido al codificador y al decodificador en dos partes. Esta partición es particularmente conveniente, ya que permite que el diseño del esquema de codificación y decodificación de canal sea virtualmente independiente de la codificación de fuente, usando información binaria como interfaz. La principal ventaja de este modelo de sistema de comunicación es el poder utilizar diferentes fuentes sobre el mismo canal.

El propósito del codificador de fuente es representar la salida de la fuente por medio de una secuencia de dígitos binarios (bits). Aquí, una de las principales cuestiones es la determinación del número mínimo de bits por unidad de tiempo requerido para representar la salida de cualquier modelo de fuente.

Por otro lado, el propósito tanto del codificador como del decodificador de canal es permitir que las secuencias de datos binarios sean reproducidas fielmente a la salida del decodificador de canal.

Un canal discreto sin memoria constituye el tipo más simple de modelos de canal, la entrada es una secuencia de símbolos de un alfabeto finito $a_1, a_2, \ldots, a_k$ y la salida una secuencia de símbolos, también de un alfabeto finito $b_1, b_2, \ldots, b_j$. Cada símbolo en la secuencia de salida depende estadística

mente sólo del simbolo en la posición correspondiente de la secuencia de entrada, y está determinado por.una asignación fija de probabilidad condicional $P(b_j|a_k)$, definida para cada a_k en el alfabeto de entrada y cada b_j en el alfabeto de salida.

Para el caso de dos símbolos en cada alfabeto se tiene el llamado canal binario simétrico (CBS), mostrado en la figura 1.3.

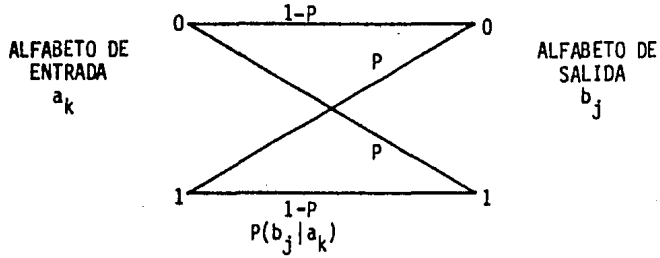


Figura 1.3 El canal binario simétrico.

En los canales discretos con memoria, la entrada y salida son también secuencias de símbolos de alfabetos finitos pero cada símbolo en la secuencia de salida puede depender estadísticamente de más de un símbolo correspondiente en la secuencia de entrada.

Otra clase de canales que se asemejan más a los canales físicos es aquélla en la que el conjunto de entradas y salidas es un conjunto de señales en el tiempo, y para cada señal de entrada la salida es un proceso aleatorio.

Un modelo de esta clase, de gran importancia teórica, es el canal con ruido blanco gaussiano (aditivo). El conjunto de entradas de este modelo es el conjunto de señales en el tiempo con un límite superior en potencia, y la salida es la suma de la entrada y ruido blanco Gaussiano.

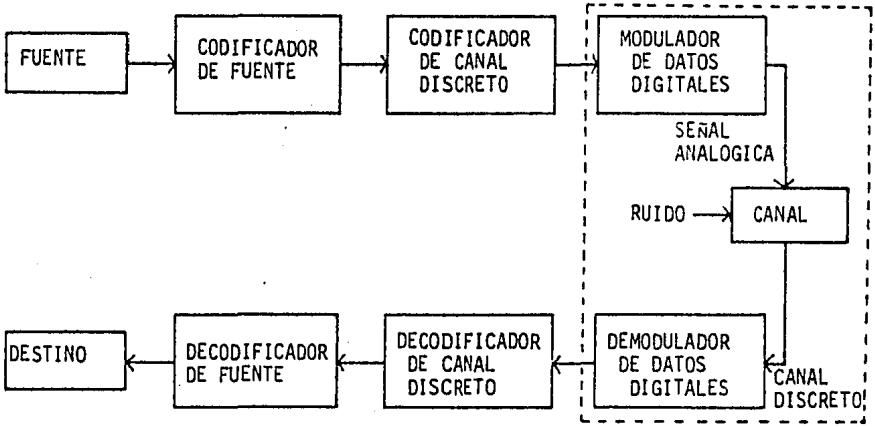


Figura 1.4 Sistema de comunicación con canal discreto.

Cuando se transmiten datos binarios sobre un canal de este tipo, es conveniente separar el esquema de codificación de canal en dos partes como se observa en la figura 1.4.

La salida del codificador de canal discreto es una secuencia de símbolos de un alfabeto finito $a_1, a_2, \ldots, a_k$. Estos símbolos son producidos a una tasa fija de tiempo, un símbolo cada τ_c segundos. En cada intervalo de τ_c segundos el modulador de datos digitales (MDD) produce una señal dada, dentro de un conjunto fijo de señales en el tiempo $S_1(t), S_2(t), \ldots, S_k(t)$, idealmente cada una de duración τ_c. La señal particular que se produzca estará determinada por el símbolo que entre al MDD en ese intervalo. Por lo tanto, la señal de entrada al canal tiene la forma

$$\sum_n S_n(t-n\,\tau_c) \qquad (1.1)$$

El demodulador de datos digitales (DDD) toma la señal recibida del canal y la convierte en una secuencia de símbolos de un alfabeto finito $b_1, b_2, \ldots, b_j$, produciendo símbolos a una tasa de un símbolo cada τ_c segundos. En el caso más sencillo, cada símbolo del DDD será una decisión (quizá incorrecta) sobre el símbolo que entró al MDD en el intervalo de tiempo correspondiente, y en este caso el alfabeto de salida $b_1, b_2, \ldots, b_j$ será el mismo que a la entrada del MDD. En casos más sofisticados, la salida del DDD contendrá tam

bién información acerca de la incertidumbre de la decisión, y el alfabeto de salida será mayor que el de entrada.

La capacidad del canal puede interpretarse como la cantidad máxima de información promedio (bits/seg) que puede transmitirse a través del canal. En términos generales el teorema de codificación para un canal discreto con ruido [38] establece que, si el canal tiene una capacidad de C bits por segundo y si los datos entran al codificador de canal a una tasa de R<C bits/seg, entonces mediante un apropiado diseño del esquema de codificación, es posible reproducir las secuencias de dígitos binarios a la salida del decodificador, con una probabilidad de error tan pequeña como se desee.

1.1.2 TECNICAS DE CODIFICACION. [1,30]

1.1.2.1 DETECCION Y CORRECCION DE ERRORES.

Frecuentemente los sistemas de comunicación utilizan canales en dos sentidos, en los cuales se utiliza comunmente un código detector de errores. De esta forma, cuando se detecta un error en el receptor (decodificador), se realiza una petición de repetición del mensaje y los errores se corrigen efectivamente. A este tipo de esquemas de codificación se le conoce como ARQ (Automatic Repeat Request).

En los canales de un solo sentido, la probabilidad de error puede ser disminuída utilizando códigos correctores de errores sin retransmisión. Por ejemplo, en las comunicaciones vía satélite, donde el equipo en el satélite es limitado, el uso de un esquema de codificación con corrección de errores es más práctico, en general, que el uso de retransmisión controlada remotamente. Este tipo de sistemas que usan códigos correctores de errores y no requieren realimentación son conocidos como FEC (Forward Error Correction).

Cuando el canal así lo permite, puede implementarse una combinación de ambos sistemas la que es más eficiente que la corrección o la detección separadamente.

1.1.2.2 CODIGOS DE BLOQUE

El codificador de un código de bloque parte la secuencia contínua de símbolos de información en bloques de k símbolos. Manipula entonces estos bloques de acuerdo a las reglas de codificación del código que se emplea. A cada bloque posible de información se le asocia un bloque de n símbolos, donde $n>k$. El bloque resultante, llamado también vector o palabra de código, es transmitido por el canal, perturbado por el ruido y finalmente decodificado, independientemente de los demás bloques o palabras de código. A la cantidad n se le conoce como longitud del código, la cantidad k se conoce a menudo como dimensión del código.

Considérese el sistema planteado en la figura 1.4, suponiendo que se desean transmitir datos sobre un canal discreto y que el canal acepta un símbolo de entrada cada τ_c segundos. Los datos binarios entran al codificador de canal a una tasa de R bits/seg.

El codificador acumula los bits en su entrada, durante un período fijo T, donde T es un parámetro de diseño del codificador. Durante ese período TR bits entran al codificador (por comodidad no se considera el caso en que TR no sea entero). El codificador puede ser visto como el conjunto de las 2^{TR} secuencias posibles de TR bits, a cada una de las cuales se le asocia una palabra de código de $n=T/\tau_c$ símbolos, para constituir la entrada al canal. Toma T segundos transmitir la palabra de código de n símbolos sobre el canal y en ese tiempo otra secuencia de TR bits está en curso de codificación, y se inicia la transmisión de la siguiente palabra de código.

El decodificador funciona de manera similar, acumula los n símbolos del canal correspondiente a la palabra de código transmitida y toma una decisión (quizá incorrecta) sobre los TR bits de entrada al codificador. Esta decisión puede basarse en una lista de todas las 2^n posibles secuencias de n bits, y el decodificador asigna a cada una de éstas la secuencia apropiada de TR bits.

Para un canal discreto dado y una tasa fija de R bits/seg, hay liber

tad en la elección, primero de T (o equivalentemente de $n=T/\tau_c$), segundo de el conjunto de 2^{TR} palabras de código, y finalmente de la regla de decisión. La probabilidad de error en los datos binarios, la complejidad del sistema y el retraso en la decodificación dependerán de estos parámetros.

1.1.2.3 CODIGOS DE ARBOL [3,16]

El otro tipo de código es el llamado código de árbol, que manipula la secuencia de símbolos de información sin separarla en bloques independientes. El codificador de un código de árbol procesa la información de forma continua y asocia a cada secuencia semi-infinita de información una secuencia de código que contiene más símbolos. El codificador parte a la secuencia de entrada en bloques de k_0 símbolos, donde k_0 es un número pequeño. Entonces, y en base al bloque de k_0 símbolos y a los símbolos de información que le han precedido, emite una porción de n_0 símbolos de la secuencia de código. El nombre de este tipo de código proviene del hecho de que las reglas de codificación son descritas mejor mediante una gráfica de árbol.

La familia de los códigos convolucionales forma parte del tipo de códigos de árbol. Estos códigos son de importancia práctica debido a su facilidad de implementación en comparación con otros códigos de árbol.

1.2 CODIGOS DE BLOQUE. [13,24,31]

1.2.1 CODIGOS LINEALES

1.2.1.1 INTRODUCCION. El conjunto de secuencias de n símbolos pertenecientes al campo de q elementos es un espacio vectorial (ver apéndice A). Un conjunto de estas secuencias de longitud n es llamado un código lineal de bloque si y solo si es un subespacio del espacio vectorial de todas las secuencias de n símbolos.

El peso de Hamming de un vector $\bar{v}$; denotado por $\omega(\bar{v})$, se define como el número de componentes diferentes de cero del vector.

La distancia de Hamming entre dos vectores $\bar{v}_1$ y $\bar{v}_2$ se define como el número de componentes en las que difieren. Claramente, la distancia entre $\bar{v}_1$ y $\bar{v}_2$ es igual a $\omega(\bar{v}_1-\bar{v}_2)$.

La distancia mínima de un código lineal es igual al peso mínimo de sus vectores no nulos.

1.2.1.2 CODIFICACION. Para un código lineal de bloque, la codificación se realiza de la siguiente manera: El vector de información $\bar{u}=u_1u_2\ldots u_k$ es codificado en un vector de código $\bar{x}=x_1x_2\ldots x_n$, donde $n>k$. La primera parte del vector de código consiste en la información $x_1=u_1$, $x_2=u_2,\ldots,x_k=u_k$, seguida por $n-k$ símbolos de redundancia $x_{k+1},\ldots,x_n$. Los símbolos de redundancia son elegidos de manera que los vectores de código satisfagan la ecuación:

$$H\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = H\bar{x}^T = 0 \qquad (1.2)$$

donde H es una matríz de $(n-k) \times n$ elementos, llamada matríz de paridad del código, dada por:

$$H = \left[\, A \mid I_{n-k} \,\right] \qquad (1.3)$$

donde A es una matriz de orden $(n-k) \times k$ con símbolos pertenecientes a un campo de q elementos e I_{n-k} es la matríz identidad.

Cuando los símbolos de información constituyen los primeros k símbolos del vector de código, se dice que el código lineal es sistemático. Entonces, cuando la matriz de paridad del código tiene la forma (1.3), el vector de código es de la forma:

$$\bar{x} = \underbrace{x_1x_2\ldots x_k}_{\substack{\text{símbolos de}\\ \text{información}}}\ \underbrace{x_{k+1}\ldots x_n}_{\substack{\text{símbolos de}\\ \text{redundancia}}} \qquad (1.4)$$

Cualquier conjunto de vectores base de un código lineal de bloque (Apéndice A) puede considerarse como los renglones de una matriz G conocida como la matriz generadora del código. Un vector pertenece al código si y solo si es una combinación lineal de los renglones de G.

Para obtener la expresión de G, se tiene primero que:
$x_i = u_i$, $i = 1, 2, \ldots, k$. Es decir:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = I_k \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix} \tag{1.5}$$

y de (1.2) y (1.3) se tiene que:

$$\left[A | I_{n-k}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = 0$$

$$A \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} + I_{n-k} \begin{bmatrix} x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = 0$$

es decir:

$$\begin{bmatrix} x_{k+1} \\ x_{k+2} \\ \vdots \\ x_n \end{bmatrix} = -A \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = -A \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix} \tag{1.6}$$

Ahora, de (1.4), (1.5) y (1.6), se obtiene:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} I_k \\ \hline -A \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix} \tag{1.7}$$

y transponiendo, se tiene:

$$\bar{x} = \left[I_k \mid -A^T \right] \bar{u} \tag{1.8}$$

Por lo tanto, la matriz generadora del código está dada por:

$$G = \left[I_k \mid -A^T \right] \tag{1.9}$$

De las ecuaciones dadas por (1.2) y (1.8) se deduce que: las matrices G y H satisfacen la relación:

$$GH^T = HG^T = 0 \tag{1.10}$$

De esta manera, se tiene un código lineal de bloque de dimensión k, longitud n, tasa o eficiencia R=k/n y distancia mínima d. En adelante se denotará el código por (n,k) o (n,k,d) indistintamente, de acuerdo a la familia de código de que se trate.

Para una mejor comprensión de los conceptos expuestos con anterioridad, considérese la matriz de paridad:

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

de símbolos pertenecientes a un campo finito de dos elementos, es decir a un

campo de Galois, CG(2), (Apéndice A). Esta matriz define a un código binario con k=3 y n=6.

Entonces, un vector de código correspondiente al mensaje $\bar{u} = u_1\ u_2\ u_3$ será aquel que cumple con las ecuaciones de paridad:

$$u_2 + u_3 + x_4 = 0$$
$$u_1 + u_3 + x_5 = 0$$
$$u_1 + u_2 + x_6 = 0$$

donde $x_4\ x_5\ x_6$ representan en este caso a los bits de redundancia. Ahora supóngase el mensaje $\bar{u} = 101$, los bits de paridad son entonces:

$$x_4 = - 0 - 1 = 1$$
$$x_5 = - 1 - 1 = 0$$
$$x_6 = - 1 - 0 = 1$$

La palabra de código correspondiente es entonces $\bar{x} = 101101$.

Debido a que existen 3 bits de información, se tienen en este caso 2^3 palabras de código:

000000
001110
010101
011011
100011
101101
110110
111000

En general, para un código (n,k) trabajando con símbolos de un campo finito de q elementos, existen q^k palabras o vectores de código.

Para el caso anterior, se observa que las palabras de código difieren entre ellas en por lo menos 3 posiciones, por lo tanto la distancia mínima del código es 3, y se trata de un código (6,3,3).

1.2.1.3. DECODIFICACION. Para un código lineal con distancia mínima d, pueden corregirse $\lfloor 1/2\ (d-1) \rfloor$ errores; si d es par, entonces además de corregir $\lfloor 1/2\ (d-1) \rfloor$ errores se pueden detectar d/2 errores. Si se denota a t como el número de errores que pueden corregirse, o capacidad de corrección del código, entonces la distancia mínima es

$$d = 2t+1 \qquad (1.11)$$

La decodificación de códigos lineales de bloque se realiza de la manera siguiente: Suponiendo que se transmitió la palabra de código $\bar{x} = x_1\ x_2\ \ldots\ x_n$ sobre el canal, y que debido al ruido la palabra recibida $\bar{y} = y_1\ y_2\ \ldots\ y_n$, puede diferir de $\bar{x}$, el vector o palabra de error es $\bar{e} = \bar{y}-\bar{x} = e_1\ e_2\ \ldots\ e_n$. La estrategia a seguir por el decodificador es elegir una $\bar{x}$, tal que el error $\bar{e}$ más probable haya sucedido, dado que se recibió $\bar{y}$. Lo anterior es óptimo en el sentido de minimizar la probabilidad del decodificador de cometer una equivocación. Esta estrategia es conocida como decodificación por máxima similitud.

Una manera bastante descriptiva de mostrar lo que hace el decodificador, es mediante el uso de una tabla llamada arreglo norma (del inglés standard). El primer renglón de la tabla está formado por todas las palabras de código, y los demás renglones se forman con los translados del código. Un translado es un arreglo de la forma $\bar{a} + C = \{\bar{a} + \bar{x} : \bar{x} \in C\}$ para cualquier $\bar{a}$, donde C es un código lineal de bloque (n,k) sobre un campo de q elementos. A la palabra de peso mínimo en un translado se le conoce como translado líder. En el caso de que haya más de una palabra, mínima en peso, dentro de un translado, puede tomarse a cualquiera de ellas como el translado líder. Cada translado del código tiene, por supuesto, q^k palabras de longitud n. La tabla consta de las q^n palabras posibles que se pueden recibir, y por lo tanto existen q^{n-k} renglones en el arreglo norma.

Para el ejemplo anterior, el código (6,3,3), el arreglo norma se presen

ta en la tabla 1.1.

Síndrome $\bar{s}$	Mensaje:	000	001	010	011	100	101	110	111
000	Código:	000000	001110	010101	011011	100011	101101	110110	111000
001	Translado:	000001	001111	010100	011010	100010	101100	110111	111001
010	Translado:	000010	001100	010111	011001	100001	101111	110100	111010
100	Translado:	000100	001010	010001	011111	100111	101001	110010	111100
110	Translado:	001000	000110	011101	010011	101011	100101	111110	110000
101	Translado:	010000	011110	000101	001011	110011	111101	100110	101000
011	Translado:	100000	101110	110101	111011	000011	001101	010110	011000
111	Translado:	100100	101010	110001	111111	000111	001001	010010	011100
		Translados líderes							

Tabla 1.1 Arreglo norma para el código lineal (6,3,3)

Nótese que en el arreglo aparecen todos los 64 vectores de longitud 6, divididos en 8 translados de 8 palabras cada uno y que los translados líderes aparecen a la izquierda. Una observación suplementaria debe hacerse en este caso, ya que en el último translado el líder tiene peso mínimo de 2. En este caso, también las palabras 001001 ó 010010 podrían haberse tomado como translados líderes.

La forma en la que se efectúa la decodificación es como sigue: Cuando se recibe $\bar{y}$, por ejemplo 110100, se localiza su posición en el arreglo. En

ta en la tabla 1.1.

Síndrome $\bar{S}$	Mensaje:	000	001	010	011	100	101	110	111
000	Código:	000000	001110	010101	011011	100011	101101	110110	111000
001	Translado:	000001	001111	010100	011010	100010	101100	110111	111001
010	Translado:	000010	001100	010111	011001	100001	101111	110100	111010
100	Translado:	000100	001010	010001	011111	100111	101001	110010	111100
110	Translado:	001000	000110	011101	010011	101011	100101	111110	110000
101	Translado:	010000	011110	000101	001011	110011	111101	100110	101000
011	Translado:	100000	101110	110101	111011	000011	001101	010110	011000
111	Translado:	100100	101010	110001	111111	000111	001001	010010	011100

Translados líderes

Tabla 1.1 Arreglo norma para el código lineal (6,3,3)

Nótese que en el arreglo aparecen todos los 64 vectores de longitud 6, divididos en 8 translados de 8 palabras cada uno y que los translados líderes aparecen a la izquierda. Una observación suplementaria debe hacerse en este caso, ya que en el último translado el líder tiene peso mínimo de 2. En este caso, también las palabras 001001 ó 010010 podrían haberse tomado como translados líderes.

La forma en la que se efectúa la decodificación es como sigue: Cuando se recibe $\bar{y}$, por ejemplo 110100, se localiza su posición en el arreglo. En

tonces, el decodificador decide que el vector o palabra de error $\bar{e}$ es la palabra de peso mínimo en el translado, es decir el translado líder 000010. $\bar{y}$ es decodificada como la palabra código $\bar{x} = \bar{y} - \bar{e} = 110110$ a la cabeza de la columna correspondiente a $\bar{y}$. Este tipo de decodificación es por máxima similitud.

Para conocer el translado en el que se encuentra $\bar{y}$, se calcula el vector columna, de longitud n-k:

$$\bar{S} = H\bar{y}^T \tag{1.12}$$

el cuál es conocido como síndrome de $\bar{y}$. El síndrome es nulo si y solo si $\bar{y}$ es una palabra de código.

En un código binario, el síndrome es igual a la suma de las columnas de H en las que ocurrieron los errores. De aquí el nombre de síndrome, ya que proporciona los síntomas de los errores.

Por ejemplo, si $\bar{y} = 110100$ el síndrome es:

$$\bar{S} = \begin{bmatrix} 011100 \\ 101010 \\ 110001 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

el cuál se localiza en la tabla a la izquierda del translado líder, que corresponde a $\bar{e}$, 000010.

1.2.2 CODIGOS CICLICOS

1.2.2.1 INTRODUCCION. [24]. Los códigos cíclicos de bloques son relativamente fáciles de implementar y poseen una estructura matemática que ha sido bastante estudiada.

Un código lineal es cíclico si cualquier corrimiento cíclico de una palabra de código $C_0\ C_1 \ldots C_{n-1}$, es también una palabra de código $C_{n-1}\ C_0 \ldots C_{n-2}$. Asociado a cada palabra de código $C_0\ C_1 \ldots C_{n-1}$ existe un polinomio de grado menor que n,

$C(x) = C_0 + C_1X + \ldots C_{n-1}x^{n-1}$ en F^n, donde F es un campo finito $CG(q)$.

Si a $C(x)$ se le multiplica por x, esto corresponde a un corrimiento cíclico. Definiendo a Rn como un anillo formado por el conjunto de polinomios en x con coeficientes en F, y de grado menor que n (i.e. módulo x^n-1), entonces:

$$xC(x) = C_0x + C_1x^2 + \ldots + C_{n-1}x^n$$

$$= C_{n-1} + C_0x + C_1x^2 + \ldots + C_{n-2}x^{n-1} \qquad (1.13)$$

ya que $x^n = 1$ en Rn.

Sea C un código cíclico de longitud n. Existe para C un polinomio único $g(x)$, de grado mínimo en C, y factor de x^n-1 conocido como polinomio generador del código.

Cualquier polinomio $C(x) \in C$ puede escribirse unívocamente como:

$$C(x) = i(x)\, g(x) \qquad (1.14)$$

donde $i(x)$ es de grado menor que $(n-m)$, siendo m el grado de $g(x)$. Entonces, la dimensión del código es $(n-m)$ y el mensaje $i(x)$ es convertido en la palabra de código $i(x)\, g(x)$.

Ya que $g(x)$ divide a x^n-1, existe un polinomio:

$$h(x) = (x^n-1)/g(x) \qquad (1.15)$$

llamado polinomio de paridad del código.

Su nombre proviene del hecho de que si $C(x)$ es un polinomio cualquiera del código C, entonces:

$$C(x)\, h(x) = i(x)\, g(x)\, h(x) = 0 \qquad (1.16)$$

en Rn. Es decir que los coeficientes de (1.16) cumplen con

$$\sum_{i=0}^{k} C_{k-i+j} h_i = 0, \qquad j = 0, 1, \ldots, n-k-1 \tag{1.17}$$

y en donde los subíndices son tomados módulo n. Por lo tanto, las ecuaciones (1.17) son las ecuaciones de paridad que satisface el conjunto de palabras del código. Esto implica que si la matriz de paridad es:

$$H = \begin{bmatrix} & & h_k \ldots h_1 h_0 \\ & h_k \ldots h_1 h_0 & \\ \ldots\ldots\ldots\ldots & & \\ h_k \ldots h_1 h_0 & & \end{bmatrix} \tag{1.18}$$

(1.17) establece que si $c \in C$, $Hc^T = 0$. Debido a que los renglones de H son linealmente independientes, la condición $Hc^T=0$, usada antes para los códigos lineales, es suficiente para que c sea una palabra de código de C.

1.2.2.2 CODIFICACION. [31]. Para codificar información usando un código cíclico existen dos métodos:

Método 1. Observando que de acuerdo a (1.17), si C(x) es una palabra de código, entonces debe satisfacer las ecuaciones de paridad:

$$\begin{aligned} &C_{n-k-1} h_k + C_{n-k} h_{k-1} + \ldots + C_{n-1} h_0 = 0 \\ &C_{n-k-2} h_k + C_{n-k-1} h_{k-1} + \ldots + C_{n-2} h_0 = 0 \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &C_0 h_k + C_1 h_{k-1} + \ldots + C_k h_0 = 0 \end{aligned} \tag{1.19}$$

esto es, C(x) satisface la recursión lineal con h(x),

$$C_l h_k + C_{l+1} h_{k-1} + \ldots + C_{l+k} h_0 = 0, \quad 0 \le l \le n-k-1 \tag{1.20}$$

Entonces, si $C_{n-1}\ C_{n-2} \cdots C_{n-k}$ se toman como los símbolos de información, las ecuaciones (1.20) definen sucesivamente los símbolos de redundancia $C_{n-k-1}\ C_{n-k-2} \cdots C_0$.

Como se dijo, una de las principales ventajas de los cõdigos cíclicos es su facilidad de implementación. Para efectuar la codificación por recurrencia lineal, se observa que las ecuaciones (1.20) pueden escribirse como:

$$\sum_{i=0}^{k} h_i\ C_{k-i+j}=0, \qquad j = 0, 1, \ldots, n-k-1 \tag{1.21}$$

o como:

$$C_j = -\sum_{i=0}^{k-1} h_i\ C_{k-i+j}, \quad j = 0, 1, \ldots, n-k-1 \tag{1.22}$$

donde $h_k=1$ y $h_0\neq 0$.

El circuito que implementa esta operación es:

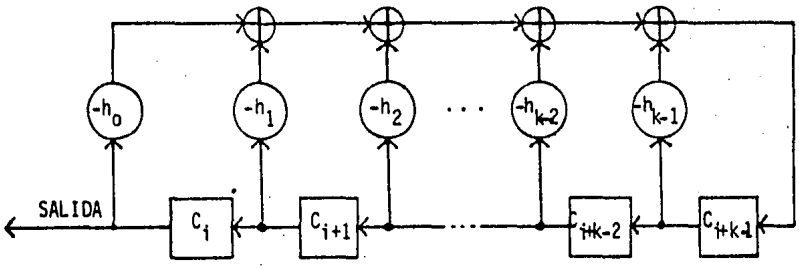


Figura 1.5 Circuito codificador por recursión lineal con h(x).

Los valores iniciales $C_0, C_1, \ldots, C_{k-1}$ son almacenados en los registros, y la salida después del i-ésimo corrimiento es C_{i-1}. Por lo tanto, los primeros k símbolos que salgan del codificador de la figura 1.5 serán los de información, y serán seguidos de los n-k símbolos de redundancia para formar la palabra de código de n símbolos.

Debe apuntarse aquí que las sumas y multiplicaciones efectuadas en el circuito de la figura 1.5, se hacen sobre un campo finito de q elementos. En el caso binario, q=2 y la suma equivale a la operación lógica OR exclusiva, mientras que la multiplicación es la operación AND.

El circuito consta de k registros de corrimiento y se recomienda cuando se tienen más símbolos de redundancia que de información.

Método 2. El polinomio C(x) es de código si y solo si es divisible por g(x). Este método es conocido como de división por g(x). Los k símbolos de información se multiplican por x^{n-k} y se dividen entre g(x).

$$\frac{x^{n-k}i(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \tag{1.23}$$

la palabra de código se obtiene mediante:

$$C(x) = q(x)\ g(x) = x^{n-k}i(x) - r(x) \tag{1.24}$$

donde r(x) es el residuo del algoritmo euclidiano de la división (1.23) y consta de los símbolos de redundancia.

El circuito que implementa la codificación por división entre g(x) es el que se muestra en la figura 1.6.

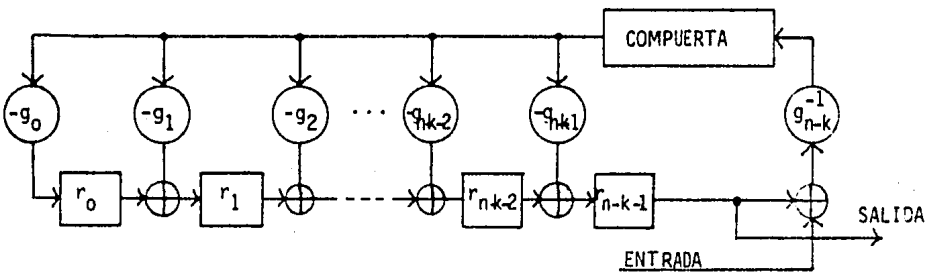


Figura 1.6 Circuito codificador de división por g(x)

La codificación se efectúa de la siguiente forma:

1. Alimentar los k símbolos de información al circuito y simultáneamente al canal de comunicación. En cuanto han entrado todos los símbolos de información, en el registro de corrimiento $(r_0, r_1, \ldots, r_{n-k-1})$ se encuentra el residuo de la división de $x^{n-k}i(x)$ por $g(x)$, i.e. los símbolos de redundancia negados.

2. Deshabilitar la compuerta de realimentación del circuito.

3. Recorrer el contenido del registro de corrimiento hacia la salida, cambiando los signos de los símbolos conforme éstos salen del codificador, y enviarlos al canal de comunicación.

Este método es recomendable para tasas de código $R>0.5$, es decir más símbolos de información que de redundancia, ya que el registro de corrimiento consta de $n-k$ etapas.

1.2.2.3 DECODIFICACION. [31]. En la decodificación de ciertos códigos cíclicos, un dispositivo muy popular es el llamado decodificador Meggitt (figura 1.7). Su funcionamiento se basa en el postulado de que si un decodificador puede corregir el primer símbolo en una palabra correctamente, para todos los patrones de error corregibles, entonces la palabra completa puede ser decodificada usando el mismo circuito.

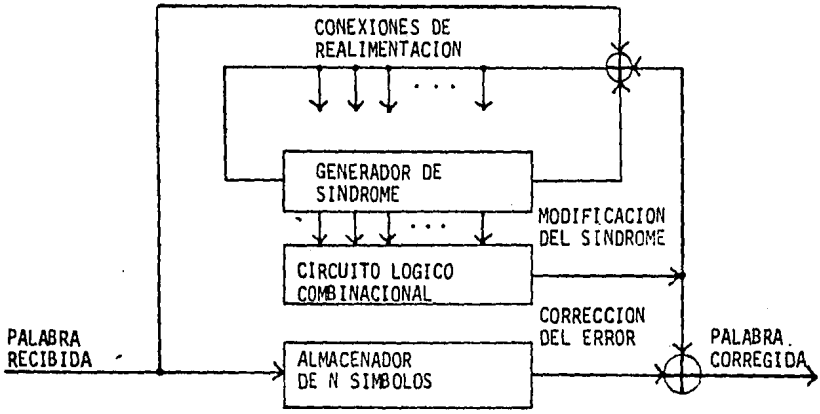


Figura 1.7. Circuito decodificador Meggit

La operación del circuito se describe a continuación:

1. Se supone que se eligen como translados líderes a los patrones de error más probable de los translados correspondientes. Además, cada corrimiento cíclico de un translado líder resulta en otro translado líder.

2. La palabra recibida es alimentada en el generador de síndrome y en el almacenador simultâneamente.

3. El circuito lógico combinacional esta diseñado de manera tal que produzca un elemento -a, del campo finito sobre el que trabaja el código, si y solo si el síndrome S(x) corresponde al patrón de error,de valor a, en el símbolo de mayor orden, i.e. el símbolo que está a punto de abandonar el decodificador.

4. Simultáneamente a la salida de símbolos del decodificador, el generador de síndrome es recorrido ciclicamente.

5. Los pasos 2 y 3 se repiten hasta que la palabra recibida ha abandonado completamente el decodificador. La salida del circuito lógico corresponde al valor del símbolo que ha de ser sumado al siguiente símbolo recibido, y al mismo tiempo corresponde a la modificación del síndrome.

6. Después de que la palabra ha sido decodificada, el síndrome consistirá de símbolos nulos en el caso de haberse corregido un patrón de error. Si el generador de síndrome contiene algún símbolo no nulo, entonces se ha detectado un patrón de error incorregible.

Con la posible excepción del circuito combinacional, el decodificador Meggitt es relativamente sencillo de implementar. Su aplicación en la decodificación dependerá del código utilizado, debido a la complejidad del circuito combinacional. En el caso de los códigos de Hamming, como se verá en el siguiente capítulo, el circuito es muy sencillo. Para los códigos correctores de paquetes de errores (Fire), analizados en el apéndice C, el circuito decodificador resulta también bastante sencillo. En el caso de los códigos

BCH y Reed-Solomon, el uso de este decodificador es, en la mayoría de los casos, prácticamente imposible debido a la gran complejidad del circuito combinacional requerido. Estas dos familias se verán en detalle en los siguientes capítulos.

CAPITULO 2

CODIGOS DE HAMMING

CAPITULO 2
CODIGOS DE HAMMING

2.1 CODIGOS LINEALES DE HAMMING. [6,24].

Los códigos de Hamming son una familia importante de códigos (cíclicos y lineales) que son fáciles de codificar y decodificar. En este trabajo se considerarán únicamente los códigos de Hamming binarios, debido a que son los que tienen un mayor interés práctico.

Recuérdese que, cuando se analizaron los códigos lineales, el síndrome de la palabra recibida corresponde a la suma de las columnas de la matriz de paridad H donde ocurrieron errores. Entonces, es posible diseñar un código corrector de errores simples (i.e. un error por palabra recibida). Para lograr esto, se hace que las columnas de H sean distintas entre sí y todas ellas diferentes de cero. Si la matriz H consta de m renglones (correspondiendo a m ecuaciones de paridad), existen entonces 2^m-1 columnas de ese tipo.

Por lo tanto, un código binario de Hamming de longitud 2^m-1, con $m \geq 2$, tiene una matriz de paridad H cuyas columnas son todos los vectores binarios de longitud m con excepción del vector nulo. Este código tiene entonces una dimensión $k=2^m-m-1$. En adelante para denotar un código de Hamming se usará $(2^m-1, 2^m-m-1)$, se sobreentiende que al corregir un error su distancia es $d=3$ y constante, por lo que no es necesario escribirla cada vez.

Para ilustrar la codificación y decodificación de códigos de Hamming, se usará el siguiente ejemplo. Considérese el caso $m=3$, entonces se tiene el código de Hamming (7,4) con matriz de paridad:

$$H = \left[\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \qquad (2.1)$$

Si el mensaje a transmitir es 0111, el codificador debe encontrar la palabra de código correspondiente, que satisfaga las ecuaciones de paridad dadas

por (1.2). Es decir:

$$H\bar{c}^T = 0$$

o explicitamente:

$$\begin{aligned} C_2 + C_3 + C_4 + C_5 &= 0 \\ C_1 + C_3 + C_4 + C_6 &= 0 \\ C_1 + C_2 + C_4 + C_7 &= 0 \end{aligned}$$

y haciendo $C_1 = 0$, $C_2 = C_3 = C_4 = 1$, se obtiene $C_5 = 1$ y $C_6 = C_7 = 0$. Por lo tanto, la palabra de código asociada al mensaje 0111 es:

$$\bar{C} = 0111100$$

Ahora, suponiendo que ha ocurrido un error en la cuarta posición de la palabra de código, el vector de error es:

$$\bar{e} = 0001000 .$$

y la palabra recibida es entonces:

$$\bar{y} = \bar{c} + \bar{e} = 0110100$$

Para decodificar la palabra recibida, primero se obtiene el síndrome (1.12):

$$\bar{S} = H\bar{y}^T = H\bar{c}^T + H\bar{e}^T = H\bar{e}^T$$

$$= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

y se localiza el síndrome en la matriz H, correspondiendo éste a la cuarta columna. Por lo tanto, ocurrió un error en la cuarta posición y la decodificación finaliza complementando el bit correspondiente.

2.1.1. CODIGOS LINEALES ACORTADOS DE HAMMING. [24].

Si i_{acor} bits del código original en la palabra de información son desechados, se obtiene un código de Hamming $(2^m-1-i_{acor}, 2^m-1-m-i_{acor})$. En la matriz de paridad, esto equivale a borrar i_{acor} columnas de H, de las primeras k columnas.

Por ejemplo, considérese m=2 e $i_{acor} = 1$, se obtiene un código de Hamming (6,3) con orden de acortamiento igual a 1. Su matriz de paridad es:

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

en la cual se desechó la cuarta columna de la matriz de paridad del código original (2.1).

El código acortado tiene una distancia mínima por lo menos igual a la del código original y el número de translados es el mismo.

Obsérvese que este código es el mismo que el analizado en el párrafo 1.2.1 y en el que se observa del arreglo norma que puede corregir todos los errores símples y además corrige el caso de dos errores en las posiciones 1 y 4.

Las operaciones de codificación y decodificación son idénticas a las vistas en la parte anterior, con la excepción de tener ahora tres bits de información con los mismos bits de redundancia.

2.1.2. CODIGOS LINEALES EXTENDIDOS DE HAMMING. [24,31].

Si además de las ecuaciones de paridad definidas por la matriz H, se agrega un bit de paridad global (i.e. $\sum_{i=1}^{n} C_i = 0$); se obtiene un código de Hamming $(2^m, 2^m-1-m)$ en el cual el peso mínimo se incrementa en la unidad, es decir d=4, este código corrige un error y detecta dos errores (1.11).

Nuevamente, considerando m=3, la matriz de paridad es:

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

Y suponiendo que el mensaje es 0111, las ecuaciones de paridad son:

$$\begin{aligned} 0 + 1 + 1 + 1 + C_5 + C_6 + C_7 + C_8 &= 0 \\ 0 + 1 + 1 + 1 + C_5 &= 0 \\ 0 + 0 + 1 + 1 + C_6 &= 0 \\ 0 + 1 + 0 + 1 + C_7 &= 0 \end{aligned}$$

de donde se obtiene $C_5=1$, $C_6=C_7=0$ y $C_8=0$, donde C_8 es el bit de paridad global. La palabra de código es por lo tanto:

$$\tilde{C} = 01111000$$

Si $\tilde{e} = 00010000$, el síndrome sería:

$$\tilde{S} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

en el cual el bit superior indica la presencia de un error simple, ya que la probabilidad de que ocurra un error simple es mayor que la probabilidad de que ocurran tres y esta a su vez mayor que la probabilidad de que ocurran cinco, etc. El síndrome corresponde a la cuarta columna de H, lugar donde efectivamente ocurrió el error.

Si ahora ocurren dos errores, e.g. $\tilde{e} = 00110000$, el síndrome es:

$$S = \begin{bmatrix}1\\1\\1\\0\end{bmatrix} + \begin{bmatrix}1\\1\\1\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

el bit superior es cero, lo que indica que han ocurrido 2 errores; debido a que la probabilidad de que ocurran dos errores es mayor que la probabilidad de que ocurran cuatro y ésta a su vez mayor que la probabilidad de que ocurran seis, etc. En general, ya que el síndrome es igual a la suma de las columnas de H donde ocurrieron los errores; si el bit superior es cero, se concluye que se presentó un número par de errores.

A continuación se presentan los resultados de la ejecución de los dos ejemplos anteriores, usando el programa HAMEXT y cuya descripción aparece en el apéndice D.1.2.

Se debe observar que la matriz H difiere de la matriz considerada en el ejemplo anterior en el orden de las columnas.

Sin embargo, ya que aparecen todas las combinaciones de vectores columna de 3 bits, con la adición de la columna de paridad global, dicha matriz es equivalente a la considerada en el ejemplo presentado en el texto.

El polinomio generador, g(x), que aparece en la ejecución del programa, se utiliza para generar todas las combinaciones de vectores de 3 bits mediante corrimientos cíclicos, como se verá en el capítulo siguiente.

Si el error ocurre en la 4a. posición, el síndrome está asociado efectivamente a la columna correspondiente en la matriz H. Además, también se presenta la ejecución para el caso en el cuál ocurren dos errores, y donde solamente es detectada la presencia de errores dentro de la palabra recibida.

```
RUN DK1:HAMEXT
 PROGRAMA SIMULADOR DE UN CODIGO DE HAMMING EXTENDIDO
 GRADO DEL POLINOMIO GENERADOR: 3
 POLINOMIO GENERADOR,G(X) ?
1101
   1 + 1*X** 1 + 0*X** 2 + 1*X** 3 +
 NO. DE BITS NO UTILIZADOS, I= 0
-----------------------------------------
 CODIGO DE HAMMING EXTENDIDO (  8,  4 )
-----------------------------------------
 MATRIZ DE PARIDAD, H:
   J=    1   2   3   4   5   6   7   8
 I= 1    1   1   1   1   1   1   1   1
 I= 2    0   1   0   0   1   0   1   1
 I= 3    0   0   1   0   1   1   1   0
 I= 4    0   0   0   1   0   1   1   1
 INFORMACION,I(X) ?
0111
   0 + 1*X** 1 + 1*X** 2 + 1*X** 3 +
 CODIGO, C(X):
00010111
   0 + 0*X** 1 + 0*X** 2 + 1*X** 3 + 0*X** 4 + 1*X** 5 + 1*X** 6 + 1*X** 7 +
 # DE OPERACIONES EN LA CODIFICACION =     5.7600000000E+02
 ERRORES, E(X) ?
00010000
 POLINOMIO CONTAMINADO, RECU(X):
00000111
   0 + 0*X** 1 + 0*X** 2 + 0*X** 3 + 0*X** 4 + 1*X** 5 + 1*X** 6 + 1*X** 7 +
```

Listado 2.1 Resultados de la simulación del código de Hamming extendido (8,4), cuando ocurre un error.

```
 SINDROME, SINDRO(X):
1001
   1 + 0*X** 1 + 0*X** 2 + 1*X** 3 +
    POLINOMIO REGENERADO, R(X):
00010111
   0 + 0*X** 1 + 0*X** 2 + 1*X** 3 + 0*X** 4 + 1*X** 5 + 1*X** 6 + 1*X** 7 +
 # DE OPERACIONES EN LA DECODIFICACION =      9.7400000000E+02
```

Listado 2.1 Resultados de la simulación del código de Hamming extendido (8.4), cuando ocurre un error (continuación).

```
RUN DK1:HAMEXT
 PROGRAMA SIMULADOR DE UN CODIGO DE HAMMING EXTENDIDO
 GRADO DEL POLINOMIO GENERADOR: 3
 POLINOMIO GENERADOR,G(X) ?
1101
   1 + 1*X** 1 + 0*X** 2 + 1*X** 3 +
 NO. DE BITS NO UTILIZADOS, I= 0
-----------------------------------------
 CODIGO DE HAMMING EXTENDIDO (  8,  4 )
-----------------------------------------
 MATRIZ DE PARIDAD, H:
    J=     1   2   3   4   5   6   7   8
 I= 1      1   1   1   1   1   1   1   1
 I= 2      0   1   0   0   1   0   1   1
 I= 3      0   0   1   0   1   1   1   0
 I= 4      0   0   0   1   0   1   1   1
```

Listado 2.2 Resultados de la simulación del código de Hamming extendido (8,4), cuando ocurren dos errores.

```
  INFORMACION,I(X) ?
0111

   0 + 1*X** 1 + 1*X** 2 + 1*X** 3 +

  CODIGO, C(X):

00010111

   0 + 0*X** 1 + 0*X** 2 + 1*X** 3 + 0*X** 4 + 1*X** 5 + 1*X** 6 + 1*X** 7 +

  # DE OPERACIONES EN LA CODIFICACION =      5.7600000000E+02

  ERRORES, E(X) ?
00110000

  POLINOMIO CONTAMINADO, RECU(X):

00100111

 . 0 + 0*X** 1 + 1*X** 2 + 0*X** 3 + 0*X** 4 + 1*X** 5 + 1*X** 6 + 1*X** 7 +

  SINDROME, SINDRO(X):

0011

   0 + 0*X** 1 + 1*X** 2 + 1*X** 3 +

  DETECCION DE ERRORES MULTIPLES, CORRECCION IMPOSIBLE


00100111

   0 + 0*X** 1 + 1*X** 2 + 0*X** 3 + 0*X** 4 + 1*X** 5 + 1*X** 6 + 1*X** 7 +

  # DE OPERACIONES EN LA DECODIFICACION =      7.2800000000E+02
```

Listado 2.2 Resultados de la simulación del código de Hamming extendido (8,4), cuando ocurren dos errores (cont.)

2.2. CODIGOS CICLICOS DE HAMMING. [24,31,37].

2.2.1. INTRODUCCION. El código binario de Hamming $(2^m-1, 2^m-m-1)$ es equivalente a un código cíclico. Si α es un elemento primitivo de $CG(2^m)$ (Apéndice A), entonces $1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}$ representan distintos vectores de m bits y pueden definir a la matriz de paridad:

$$H = \left[\alpha^{2^m-2} \ \ldots \ \alpha \ 1\right] \tag{2.2}$$

En un código cíclico, C(x) es una palabra de código si y solo si α es una raíz de C(x). Entonces, si la función mínima de α es un polinomio primitivo, el polinomio generador del código, g(x), es primitivo. En general, cualquier código generado por un polinomio primitivo es un código de Hamming.

Nuevamente considérese el código de Hamming (7,4) con m=3 y polinomio generador $g(x) = 1 + x + x^3$. Si el mensaje es 0111, éste tendrá asociado el polinomio $i(x) = 1 + x + x^2$.

2.2.2 CODIFICACION. La codificación puede realizarse de dos formas.

Método 1. Por recurrencia con h(x) .

Primero, se obtiene h(x).

$$h(x) = (x^7-1)/(x^3 + x + 1) = x^4 + x^2 + x + 1$$

las ecuaciones de recurrencia (1.20) son

$$C_1h_4 + C_{\ell+1}h_3 + C_{\ell+2}h_2 + C_{\ell+3}h_1 + C_{\ell+4}h_0 = 0, \quad 0 \leq \ell \leq 2$$

sustituyendo h_k, $k = 0, \ldots, 4$:

$$C_\ell + C_{\ell+2} + C_{\ell+3} + C_{\ell+4}, \quad 0 \leq \ell \leq 2$$

con $C_6=0$, $C_5=C_4=C_3=1$

$$\ell = 2: \quad C_2 = C_4 + C_5 + C_6 = 0$$

$$\ell = 1: \quad C_1 = C_3 + C_4 + C_5 = 1$$

$$\ell = 0: \quad C_0 = C_2 + C_3 + C_4 = 0$$

y el polinomio de código es:

$$C(x) = x + x^3 + x^4 + x^5$$

La implementación de este método de codificación se logra mediante el circuito de la figura 2.1, cuya operación está descrita en la tabla 2.1.

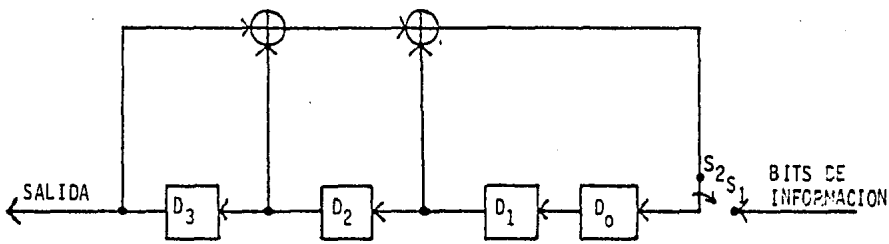


Figura 2.1 Circuito codificador, por recurrencia con $h(x)$, para el Código de Hamming (7,4).

SALIDA	D_3	D_2	D_1	D_0	CORRIMIENTO
0	1	1	1	0	1
1	1	1	0	1	2
1	1	0	1	0	3
1	0	1	0	0	4
0	1	0	0	1	5
1	0	0	1	1	6
0	0	1	1	1	7

Tabla 2.1. Operación del circuito de la figura 2.1.

El funcionamiento del codificador de la figura 2.1 es el siguiente:

1. Los bits de información se almacenan ya sea mediante 4 corrimientos (adquisición en serie , posición S_1) o directamente en los registros D_0-D_3 (adquisición en paralelo).

2. Se efectúan 7 corrimientos (Tabla 2.1). En el primer corrimiento, el primer bit de redundancia es obtenido mediante las ecuaciones de paridad y al mismo tiempo el primer bit de información sale del codificador.

Método 2. Por división por $g(x)$.

El polinomio de información es premultiplicado por x^{n-k}, y se divide por $g(x)$ usando el algoritmo euclidiano de la división, el residuo constituye la redundancia a añadir a $x^{n-k}\, i(x)$, para formar la palabra de código $C(x)$:

$$x^{n-k}\, i(x) = x^3(1 + x + x^2) = x^3 + x^4 + x^5$$

haciendo la división:

$$\begin{array}{r|l} x^5 + x^4 + x^3 & x^3 + x + 1 \\ \hline x^5 \quad + x^3 + x^2 & x^2 + x \\ \hline \quad x^4 + \quad + x^2 & \\ \quad x^4 \quad + x^2 + x & \\ \hline \qquad\qquad\quad x & \end{array}$$

y por lo tanto $r(x) = x$. La palabra de código es entonces (1.24)

$$\begin{aligned} c(x) &= x^{n-k}\, i(x) - r(x) \\ &= x + x^3 + x^4 + x^5 \end{aligned}$$

El circuito mostrado en la figura 2.2 realiza la operación anterior, y su operación aparece en la tabla 2.2.

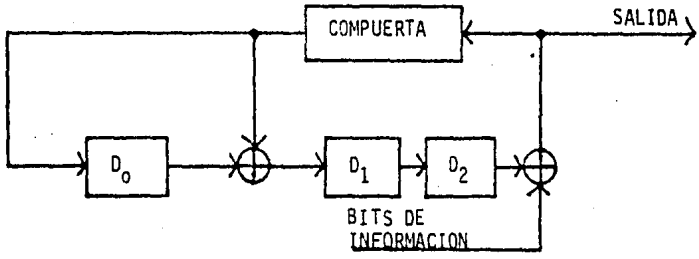


Figura 2.2 Circuito codificador, por división por g(x), para el código de Hamming (7,4).

SALIDA	D_0	D_1	D_2	CORRIMIENTO	
0	0	0	0	1	
1	1	1	0	2	
1	1	0	1	3	
1	0	1	0	4	
0	0	0	1	5	*Compuerta deshabilitada.
1	0	0	0	6	
0	0	0	0	7	

Tabla 2.2 Operación del circuito de la figura 2.2.

El funcionamiento de este codificador es de la siguiente manera:

1. Recorrer los 4 símbolos de información en el circuito y simultáneamente alimentarlos al canal de comunicación.

2. Deshabilitar la compuerta de realimentación del circuito.

3. Hacer el corrimiento de las etapas hasta obtener los 3 bits de redundancia.

Como se puede ver, el segundo método es más eficiente y rápido, ya que requiere menos etapas y tarda menos la codificación.

2.2.3 DECODIFICACION. Para ilustrar la decodificación de los códigos cíclicos de Hamming, se continuará con el ejemplo anterior. Suponiendo que ocurrió un error en la posición 4, esto es que el vector o patrón de error es 0001000, con su polinomio asociado $e(x) = x^3$, entonces el polinomio recibido es:

$$r(x) = C(x) + e(x) = x + x^4 + x^5$$

La decodificación se efectúa de la siguiente manera: Primero se calcula el síndrome del polinomio recibido:

$$S(x) = r(x) \text{ módulo } g(x) \qquad (2.3)$$

Recordando que, en el caso del código lineal, el síndrome se calculaba como $\bar{S} = H\bar{y}^T$ (1.12) y era igual a la suma de las columnas de H donde ocurrieron errores; en el código cíclico, la columna j de H es el vector de coeficientes del residuo obtenido por la división de x^{n-1-j} por $g(x)$.

Considerando la matriz de paridad del código de Hamming (7,4):

$$H = \left[\alpha^6 \ \alpha^5 \ \alpha^4 \ \alpha^3 \ \alpha^2 \ \alpha \ \ 1\right]$$

$$= \left[\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

la cuarta columna se obtiene como x^3 módulo $g(x) = 1 + x$, cuyo vector columna correspondiente es: $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$.

El síndrome para el ejemplo que se trata, es:

$$S(x) = (x + x^4 + x^5) \text{ módulo } (1 + x + x^3)$$

analizando la división:

$$
\begin{array}{r|l}
x^3 + x + 1 & x^2 + x + 1 \\
\hline
x^5 + x^4 + x & \\
x^5 + x^3 + x^2 & \\
\hline
x^4 + x^3 + x^2 + x & \\
x^4 + x^2 + x & \\
\hline
x^3 & \\
x^3 + x + 1 & \\
\hline
x + 1 &
\end{array}
$$

se obtiene: $S(x) = 1 + x$.

Como en el caso de los códigos cíclicos, en general la matriz H no se encuentra disponible en el decodificador, el siguiente paso es calcular x^{n-1-j} módulo $g(x)$, para $j = 0, \ldots, n-1$, hasta que el polinomio resultante sea igual al síndrome $S(x)$. En ese momento, j será igual a la posición del error en la palabra recibida.

Siguiendo con el ejemplo, para $j=4$ se obtiene x^3 módulo $(1+x+x^3) = 1 + x = S(x)$ y por lo tanto existe un error en la cuarta posición. La palabra corregida es entonces:

$$C(x) = r(x) + e(x) = (x+x^4+x^5) + x^3 = x + x^3 + x^4 + x^5$$

y de esta forma la decodificación se ha completado.

La decodificación puede ser realizada por medio del decodificador Meggitt discutido anteriormente. El circuito en este caso es el mostrado en la figura 2.3.

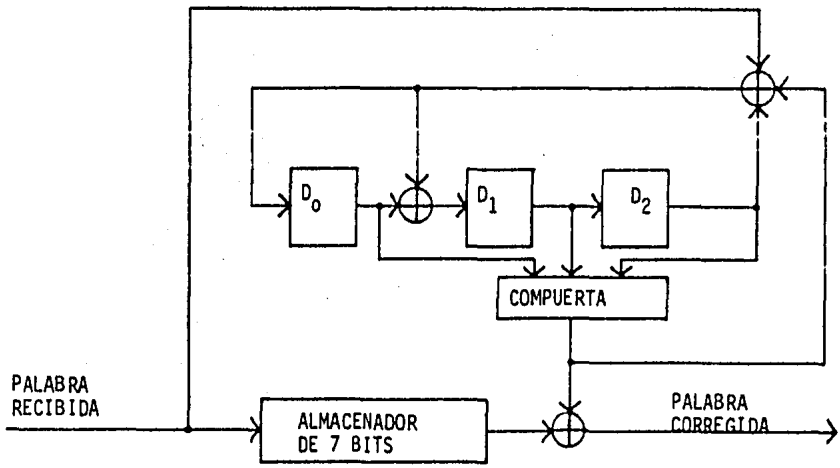


Figura 2.3 Circuito decodificador Meggitt para el código de Hamming (7,4).

Para explicar el diseño del decodificador, primero obsérvese que la matriz de paridad H de un código binario de Hamming tiene como primera columna a α^{n-1}, donde α es un elemento primitivo de CG (2^m) y $n=2^m-1$. Si ocurre un error simple en la posición más significativa, el síndrome de m bits corresponderá a α^{n-1}. Si el error ocurre en otra posición, el síndrome no será α^{n-1}.

Debido a que el decodificador Meggitt premultiplica al síndrome por $x^{n-k}=x^m$, entonces los registros contienen el vector binario correspondiente a α^{m-1}, el síndrome de x^{m-1}. El síndrome será el vector (000...1) de m bits. Por lo tanto, el circuito combinacional del decodificador consistirá simplemente en una compuerta 'AND' que proporcione una salida de 1 cuando se presente este síndrome.

Mas descriptivamente, el circuito de la figura 2.3 puede redibujarse para dar como resultado el circuito de la figura 2.4. La operación del circuito está mostrada en la tabla 2.3.

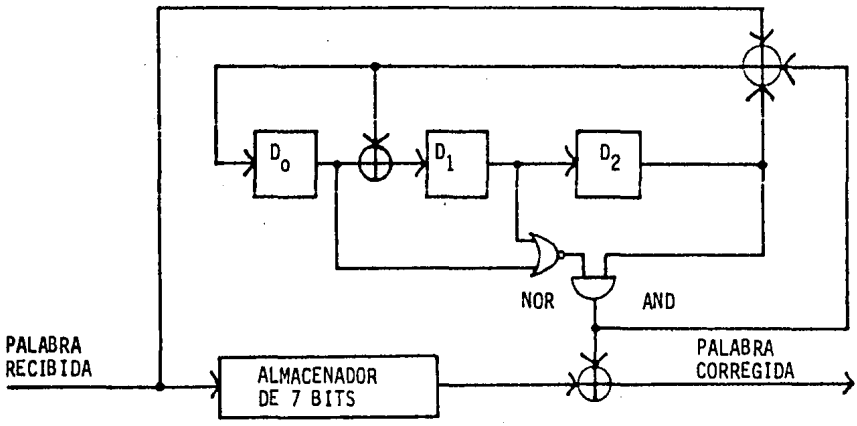


Figura 2.4. Circuito decodificador Meggitt, para el código de Hamming (7,4).

CORRIMIENTO	D_o	D_1	D_2	ALMACENADOR ENTRADA	ALMACENADOR SALIDA	SALIDA AND	SALIDA DECODIFICADOR
1	0	0	0	0	-	0	-
2	1	1	0	1	-	0	-
3	1	0	1	1	-	0	-
4	1	0	0	0	-	0	-
5	0	1	0	0	-	0	-
6	1	1	1	1	-	0	-
7	1	0	1	0	-	0	-
8	1	0	0	-	0	0	0
9	0	1	0	-	1	0	1
10	0	0	1	-	1	1*	1
11	0	0	0	-	0	0	1*
12	0	0	0	-	0	0	0
13	0	0	0	-	1	0	1
14	0	0	0	-	0	0	0

Tabla 2.3 Operación del circuito de la figura 2.4.

El ejemplo anterior fué simulado usando el programa HAMMIN, implementado en computadora, y cuyos resultados se presentan a continuación.

```
 RUN DK1:HAMMIN
  PROGRAMA SIMULADOR DE UN CODIGO DE HAMMING
  GRADO DEL POLINOMIO GENERADOR: 3
  POLINOMIO GENERADOR, G(X) ?
1101
   1 + 1*X** 1 + 0*X** 2 + 1*X** 3 +
  NO. DE BITS NO UTILIZADOS, I= 0
------------------------------
  CODIGO DE HAMMING  (  7,  4 )
------------------------------
  INFORMACION, I(X) ?
1110
   1 + 1*X** 1 + 1*X** 2 + 0*X** 3 +
  CODIGO, C(X)
0101110
   0 + 1*X** 1 + 0*X** 2 + 1*X** 3 + 1*X** 4 + 1*X** 5 + 0*X** 6 +
  # DE OPERACIONES EN LA CODIFICACION =      2.9800000000E+02
  POSICION DEL ERROR, POS= 4
  POLINOMIO CONTAMINADO, RECU(X)

0100110
   0 + 1*X** 1 + 0*X** 2 + 0*X** 3 + 1*X** 4 + 1*X** 5 + 0*X** 6 +
  POLINOMIO REGENERADO, R(X)

0101110
   0 + 1*X** 1 + 0*X** 2 + 1*X** 3 + 1*X** 4 + 1*X** 5 + 0*X** 6 +
  # DE OPERACIONES EN LA DECODIFICACION =      2.5260000000E+03
```

Listado 2.3 Resultados de la simulación del código de Hamming (7,4).

2.3 CODIGOS CICLICOS ACORTADOS DE HAMMING. [31].

2.3.1. INTRODUCCION. Al igual que en el caso de los códigos lineales, un código cíclico de Hamming puede acortarse i_{acor} bits para formar el nuevo código $(2^m-1-i_{acor}, 2^m-1-m-i_{acor})$. Las mismas técnicas del código original pueden usarse para codificar y decodificar, observando solamente que cada palabra de código tiene siempre i_{acor} ceros en las primeras posiciones. A este tipo de códigos se les conoce como códigos pseudocíclicos.

Debido a que la distancia del código acortado es por lo menos igual a la del original, los códigos acortados forman parte de los mejores códigos correctores de errores aleatorios.

Como ejemplo, se considera el código cíclico acortado de Hamming (6,3) con $m=3$, $i_{acor}=1$ y polinomio generador $g(x) = 1 + x + x^3$.

2.3.2. CODIFICACION. Si se tiene el mensaje 101, éste es codificado usando el mismo circuito de la figura 2.2. Su operación se muestra en la tabla 2.4.

SALIDA	D_0	D_1	D_2	CORRIMIENTO	
1	1	1	0	1	
0	0	1	1	2	
1	0	0	1	3	
1	0	0	0	4	* compuerta deshabilitada
0	0	0	0	5	
0	0	0	0	6	

Tabla 2.4 Operación del circuito de la figura 2.2, para el código de Hamming (6,3).

Algebráicamente, la codificación sería de la siguiente forma:

$$x^{n-k}i(x) = x^3 (1+x^2) = x^3 + x^5$$

$$
\begin{array}{r|l}
 & x^2 \\
\hline
x^3 + x + 1 & x^5 \quad + x^3 \\
 & \underline{x^5 \quad +x^3 + x^2} \\
 & \quad x^2
\end{array}
$$

y el polinomio de código es:

$$C(x) = x^2 + x^3 + x^5$$

2.3.3 DECODIFICACION. La decodificación se realiza de manera análoga a la del código original. Primero se obtiene el síndrome $S(x)$ y luego se localiza la posición del error. Suponiendo un patrón de error $e(x) = 1$, entonces:

$$r(x) = 1 + x^2 + x^3 + x^5$$

y el síndrome está dado por:

$$
\begin{array}{r|l}
 & x^2 \\
\hline
x^3 + x + 1 & x^5 + \quad + x^3 + x^2 + 1 \\
 & \underline{x^5 \quad + x^3 + x^2} \\
 & \qquad\qquad 1
\end{array}
$$

teniendo $S(x) = 1$, esto corresponde a x^0 módulo $g(x)$, por lo que $j=n-1$ la posición menos significativa en la palabra recibida. De esta manera la palabra que se obtiene después de la corrección es:

$$C(x) = (1+x^2+x^3+x^5) + 1 = x^2+x^3+x^5$$

El circuito decodificador Meggitt en este caso debe premultiplicar a la palabra recibida ya no por x^{n-k}, sino por $x^{n-k+i_{acor}}$ módulo $g(x)$. Si

$$
\begin{aligned}
f(x) &= x^{n-k+i_{acor}} \text{ módulo } g(x) \\
&= x^4 \text{ módulo } (x^3+x+1) \\
&= x + x^2
\end{aligned}
\qquad (2.4)
$$

la palabra recibida debe premultiplicarse por $f(x)$ y calcular el síndrome.

Para una mejor comprensión del circuito decodificador, se presenta a continuación el circuito que efectúa la multiplicación por $h(x)$ y la división por $g(x)$.

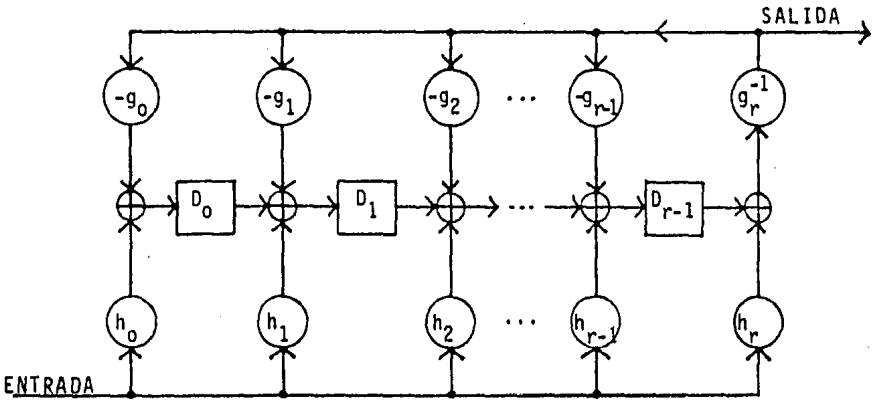


Figura 2.5 Circuito que multiplica por $h(x)$ y divide por $g(x)$.

El decodificador Meggitt para el código cíclico acortado de Hamming (6,3) es el mostrado en la figura 2.6. Y para el ejemplo considerado, su operación se presenta en la tabla 2.5.

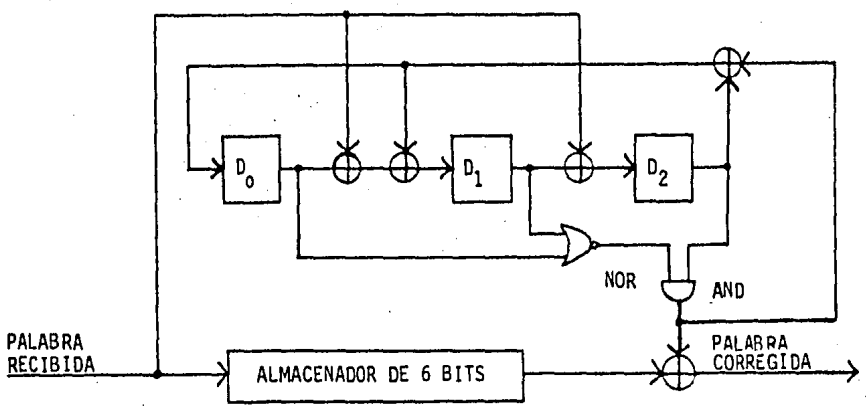


Figura 2.6 Circuito decodificador Meggitt, para el Código de Hamming (6,3).

CORRIMIENTO	D_o	D_1	D_2	ALMACENADOR ENTRADA	ALMACENADOR SALIDA	SALIDA AND	SALIDA DECODIFICADOR
1	0	1	1	1	-	0	-
2	1	0	1	0	-	0	-
3	1	1	1	1	-	0	-
4	1	0	0	1	-	0	-
5	0	0	0	0	-	0	-
6	0	1	1	1	-	0	-
7	1	1	1	-	1	0	1
8	1	0	1	-	0	0	0
9	1	0	0	-	1	0	1
10	0	1	0	-	1	0	1
11	0	0	1	-	0	1*	0
12	0	0	0	-	1	0	0*

Tabla 2.5 Operación del circuito de la figura 2.6.

El ejemplo anterior fué simulado por computadora y los resultados son:

```
RUN DK1:HAMMIN

 PROGRAMA SIMULADOR DE UN CODIGO DE HAMMING

 GRADO DEL POLINOMIO GENERADOR: 3

 POLINOMIO GENERADOR, G(X) ?
1101

  1 + 1*X** 1 + 0*X** 2 + 1*X** 3 +

 NO. DE BITS NO UTILIZADOS, I= 1

--------------------------------

 CODIGO DE HAMMING  (  6,  3 )

--------------------------------
```

Listado 2.4 Resultado de la simulación del código de Hamming (6,3)

```
  INFORMACION, I(X) ?
101

   1 + 0*X** 1 + 1*X** 2 +

  CODIGO, C(X)

001101

   0 + 0*X** 1 + 1*X** 2 + 1*X** 3 + 0*X** 4 + 1*X** 5 +

  # DE OPERACIONES EN LA CODIFICACION =      2.5000000000E+02

  POSICION DEL ERROR, POS= 1

  POLINOMIO CONTAMINADO, RECU(X)

101101

   1 + 0*X** 1 + 1*X** 2 + 1*X** 3 + 0*X** 4 + 1*X** 5 +

  POLINOMIO REGENERADO, R(X)

001101

   0 + 0*X** 1 + 1*X** 2 + 1*X** 3 + 0*X** 4 + 1*X** 5 +

  # DE OPERACIONES EN LA DECODIFICACION =      3.3200000000E+02
```

Listado 2.4 Resultados de la simulación del código de Hamming (6,3) (Cont.).

2.4 CODIGOS CICLICOS EXTENDIDOS DE HAMMING. [24,31].

El código de Hamming $(2^m, 2^m-m-1)$ formado mediante la adición de un bit de paridad global, tiene una distancia mínima de 4, como se vió anteriormente. La matriz de paridad en términos de α, la raíz primitiva de $CG(2^m)$, es:

$$H = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 & 1 \\ \alpha^{2^m-2} & \alpha^{2^m-3} & \dots & \alpha & 1 & 0 \end{bmatrix} \qquad (2.5)$$

este código no es cíclico.

Sin embargo, un código cíclico de Hamming con parámetros $(2^m-1, 2^m-m-2)$ con distancia mínima 4, logrado mediante el desecho de 1 bit de información, tiene un polinomio generado que está dado por:

$$g(x) = (1+x)\, p(x) \qquad (2.6)$$

donde $p(x)$ es un polinomio primitivo de grado m. La matriz de paridad para este código es:

$$H = \begin{bmatrix} 1 & 1 & \dots & 1 \\ \alpha^{2^m-2} & \alpha^{2^m-3} & \dots & 1 \end{bmatrix} \qquad (2.7)$$

la cuál difiere de la matriz (2.5) considerada inicialmente. En este caso una palabra de código $C(x)$ es válida si y solo si, α y 1 son raíces de $C(x)$.

CAPITULO 3

CODIGOS DE BCH

CAPITULO 3
CODIGOS DE BCH

3.1 INTRODUCCION. [2,24,31].

Un código cíclico de longitud n sobre un campo finito de q elementos, $CG(q)$, es un código de BCH de distancia diseñada d si, para algún entero $V \geq 0$, el polinomio generador del código es:

$$G(x) = MCM\left\{M^{(V)}(x),\ M^{(V+1)}(x),\ldots,M^{(V+d-2)}(x)\right\} \qquad (3.1)$$

donde $MCM\{\}$ indica el mínimo común múltiplo del argumento. Es decir, $G(x)$ es el polinomio de menor grado sobre $CG(q)$ cuyas raíces son $\alpha^{V}, \alpha^{V+1}, \ldots, \alpha^{V+d-2}$. Donde $M^{(i)}(x)$ indica el polinomio mínimo correspondiente a α^{i} (Apéndice A).

Entonces, una palabra $C(x)$ pertenece al código si y solo si:

$$C(\alpha^{V}) = C(\alpha^{V+i}) = \ldots = C(\alpha^{V+d-2}) = 0 \qquad (3.2)$$

De lo anterior se deduce que la matriz de paridad del código es:

$$H = \begin{bmatrix} 1 & \alpha^{V} & \alpha^{2V} & \ldots & \alpha^{(n-1)V} \\ 1 & \alpha^{V+1} & \alpha^{2(V+1)} & \ldots & \alpha^{(n-1)(V+1)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \alpha^{V+d-2} & & \ldots & \alpha^{(n-1)(V+d-2)} \end{bmatrix} \qquad (3.3)$$

Si $n=q^{m}-1$ y α es un elemento primitivo de $CG(q^{m})$, entonces el código BCH es primitivo.

Para el caso en que $q=2$, el código cíclico binario de BCH tiene un polinomio generador de menor grado, debido a que $M^{(2i)}(x) = M^{(i)}(x)$ (Apéndice A). Tomando $V=1$ y $d=2t-1$, se obtienen los más importantes códigos de BCH binarios, también conocidos como códigos BCH en el sentido justo (del inglés narrow

sence). El polinomio generador de estos códigos es de la forma:

$$g(x) = \text{MCM}\left\{M^{(1)}(x), M^{(3)}(x), \ldots, M^{(2t-1)}(x)\right\} \qquad (3.4)$$

y la matriz de paridad del código es ahora:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \ldots & \alpha^{3(n-1)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \alpha^{2t-1} & & \ldots & \alpha^{(2t-1)(n-1)} \end{bmatrix} \qquad (3.5)$$

donde cada elemento α^i de $CG(2^m)$ corresponde a un vector binario de m bits.

Los códigos de BCH son una generalización de los códigos de Hamming para corrección de t errores [30]. Esto puede verse claramente si se intenta construir un código de Hamming de longitud $n=2^m-1$ con capacidad para corregir 2 errores. Una manera de lograr esto es añadir m renglones a la matriz de paridad del código original. La matriz de paridad de un código de Hamming (2.2) puede escribirse como:

$$H = \begin{bmatrix} 1 & \alpha & \ldots & \alpha^{2^m-2} \end{bmatrix} \qquad (3.6)$$

añadir m renglones de elementos binarios a la matriz (3.6), equivale a incluir un renglón de elementos de $CG(2^m)$.

Por lo tanto, la matriz de paridad correspondiente a un código de Hamming que pudiera corregir 2 errores es:

$$H = \begin{bmatrix} 1 & \alpha & \ldots & \alpha^{2^m-2} \\ 1 & \alpha^3 & & \alpha^{(2^m-2)3} \end{bmatrix}$$

es decir, la matriz H de un código BCH corrector de 2 errores.

Para ilustrar lo anterior [24], sea $q=2$ y α un elemento primitivo de

$CG(2^4)$, i.e. $\alpha^{15}=1$. Considerando una distancia diseñada d=7, es decir con capacidad para corregir 3 errores; entonces se cumple que C(x) es una palabra de código, si y solo si, $\alpha, \alpha^2, \ldots, \alpha^5$ son sus raíces.

Sea $M^{(i)}(x)$ el polinomio mínimo de α^i. Entonces α, α^2, α^4 y α^8 son las raíces de $M^{(1)}(x)$, y además $M^{(1)}(x) = M^{(2)}(x) = M^{(4)}(x) = M^{(8)}(x)$. De manera similar, α^3, α^6, α^{12} y α^9 son raíces de $M^{(3)}(x)$. Lo anterior puede ser simplificado listando sólo los exponentes para formar lo que se conoce como arreglo cíclico:

1, 2, 4, 8, 16 = 1	$M^{(1)}(x)=M^{(2)}(x)=M^{(4)}(x)=M^{(8)}(x)$	grado 4
3, 6, 12, 24=9, 18 = 3	$M^{(3)}(x)=M^{(6)}(x)=M^{(12)}(x)=M^{(9)}(x)$	grado 4
5, 10, 20 = 5	$M^{(5)}(x)=M^{(10)}(x)$	grado 2

Por lo tanto $g(x) = M^{(1)}(x)M^{(3)}(x)M^{(5)}(x)$ es de grado 10, y n-k=10 bits de redundancia, obteniéndose el código de BCH (15,5,7) con capacidad de corregir 3 errores.

Si se elige a (x^4+x+1) como el polinomio primitivo generador de $CG(2^4)$, entonces:

$$M^{(1)}(x) = x^4 + x + 1$$
$$M^{(3)}(x) = x^4 + x^3 + x^2 + x + 1$$
$$M^{(5)}(x) = x^2 + x + 1$$

y los polinomios generador y de paridad (1.15) del código son:

$$g(x) = M^{(1)}(x)M^{(3)}(x)M^{(5)}(x) = x^{10}+x^8+x^5+x^4+x^2+x+1$$
$$h(x) = (x^{10}-1)/g(x) = x^5+x^3+x+1 \qquad (3.7)$$

la matriz de paridad es:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \\ 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} & 1 & \alpha^5 & \alpha^{10} \end{bmatrix} \qquad (3.8)$$

3.2 CODIFICACION.

La codificación para un código BCH es la misma que la analizada en el capítulo 1 para códigos cíclicos. Siguiendo con el ejemplo anterior, supóngase que la información a transmitir es 10110, cuyo polinomio asociado es:

$$i(x) = x + x^2 + x^4$$

Método 1. Por recurrencia con $h(x)$. Las ecuaciones de paridad (1.20) para este caso son:

$$C_\ell h_5 + C_{\ell+1} h_4 + \ldots + C_{\ell+5} h_0 = 0 \quad , \quad 0 \leq \ell \leq 9$$

donde:

$$C_{10}=0,\ C_{11}=C_{12}=1,\ C_{13}=0 \text{ y } C_{14}=1$$

para $\ell=9$:

$$C_9 = C_{10}h_4 + C_{11}h_3 + C_{12}h_2 + C_{13}h_1 + C_{14}h_0$$
$$= 0 + 1 + 0 + 0 + 1 = 0$$

de manera análoga se obtienen los demás bits de redundancia,

$$C_8 = 1,\ C_7 = C_6 = C_5 = 0,\ C_4 = C_3 = C_2 = C_1 = 1,\ C_0 = 0$$

y la palabra de código es:

$$C(x) = x + x^2 + x^3 + x^4 + x^8 + x^{11} + x^{12} + x^{14}$$

Método 2. Por división por $g(x)$

$$C(x) = x^{n-k}i(x) + x^{n-k}i(x) \quad \text{módulo } g(x)$$

$$x^{n-k}i(x) = x^{10}(x + x^2 + x^4) = x^{11} + x^{12} + x^{14} \tag{3.9}$$

$$\begin{array}{r|l}
x^{14}+x^{12}+x^{11} & x^4+x \\
\hline
x^{14}+x^{12}+x^9+x^8+x^6+x^5+x^4 & \\
\hline
x^{11}+x^9+x^8+x^6+x^5+x^4 & \\
\hline
x^{11}+x^9+x^6+x^5+x^3+x^2+x & \\
\hline
x^8+x^4+x^3+x^2+x &
\end{array}$$

$x^{10}+x^8+x^5+x^4+x^3+x^2+x+1$

de esta forma:

$$C(x) = x + x^2 + x^3 + x^4 + x^8 + x^{11} + x^{12} + x^{14}$$

3.3 DECODIFICACION. [11,24,26,31].

La decodificación de los códigos de BCH se divide en 4 etapas.

i) Cálculo del síndrome
ii) Cálculo del polinomio localizador de errores, $\sigma(x)$
iii) Evaluación de las raíces de $\sigma(x)$
iv) Evaluación del valor de los errores y corrección de la palabra recibida.

i) Cálculo del síndrome. [24,31].

Considerando al código de BCH como el código lineal que es, el síndrome se calcula mediante el producto siguiente:

$$\bar{S} = H\bar{r}^T = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(n-1)} \\ \vdots & & & & \\ 1 & \alpha^{d-2} & & \dots & \alpha^{(d-2)(n-1)} \end{bmatrix} \begin{bmatrix} r_o \\ r_1 \\ \vdots \\ r_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} \Sigma r_j \alpha^j \\ \Sigma r_j \alpha^{3j} \\ \vdots \\ \Sigma r_j \alpha^{(d-2)j} \end{bmatrix} = \begin{bmatrix} r(\alpha) \\ r(\alpha^3) \\ \vdots \\ r(\alpha^{d-2}) \end{bmatrix} = \begin{bmatrix} S_1 \\ S_3 \\ \vdots \\ S_{d-2} \end{bmatrix} \qquad (3.10)$$

donde $S_i = r(\alpha^i)$. Nótese que $S_{2i} = r(\alpha^{2i}) = r(\alpha^i)^2 = S_i^2$. Entonces, los t síndromes S_i, i = 1, 3, ..., 2t-1 se calculan a partir de la palabra recibida r(x) evaluada para α^i. Los demás síndromes $S_i = S_{i/2}^2$ para i = 2, 4, ..., 2t son obtenidos a partir de los calculados anteriormente. Entonces:

$$S_i = r(\alpha^i) = e(\alpha^i) = \sum_{j=0}^{n-1} e_j \alpha^{ij} = \sum_{j=1}^{e} Y_j X_j^i, \quad i=1,..,2t;\ e \leq t \qquad (3.11)$$

donde X e Y son la posición y el valor del error respectivamente.

Para el ejemplo anterior, si debido al ruido del canal se tiene el patrón de error $e(x) = 1 + x^6 + x^{12}$, entonces la palabra recibida por el decodificador es:

$$r(x) = C(x) + e(x) = 1+x+x^2+x^3+x^4+x^6+x^8+x^{11}+x^{14} \qquad (3.12)$$

y los síndromes son:

$$\begin{aligned} S_1 &= r(\alpha) = 1 + \alpha^6 + \alpha^{12} = \alpha \\ S_3 &= r(\alpha^3) = 1 + \alpha^3 + \alpha^6 = \alpha^8 \\ S_5 &= r(\alpha^5) = 1 + 1 + 1 = 1 \\ S_2 &= S_1^2 = \alpha^2 \end{aligned} \qquad (3.13)$$

$$S_4 = S_2^2 = \alpha^4$$
$$S_6 = S_3^2 = \alpha$$

ii) Cálculo del polinomio localizador de errores, $\sigma(x)$. [6,31,32].

El cálculo de $\sigma(x)$ se implementa mediante el algoritmo de Berlekamp-Massey, que consiste en un método iterativo, a partir de ciertas condiciones iniciales, que utiliza los componentes del síndrome, S_i, $i \in [1,2t]$.

El polinomio localizador de errores está definido por la siguiente expresión:

$$\sigma(x) = \prod_{j=1}^{e} (x-X_j) = \sum_{j=0}^{e} \sigma_j x^{e-j} \qquad (3.14)$$

donde $\sigma_0=1$ y el grado de $\sigma(x)$ es menor o igual que t.

Para entender lo que realiza el algoritmo, primero se multiplica la expresión (3.14) por $X_j^i Y_j$ y se sustituye $x=X_j$, quedando:

$$\prod_{i=1}^{e} (X_j-X_j)\, X_j^i Y_j = \sum_{j=0}^{e} \sigma_j X_j^{e-j+i} Y_j, \quad i \in [1,2t]. \qquad (3.15)$$

y ya que $S_i = \sum_{j=1}^{e} X_j^i Y_j$ (3.11), entonces la ecuación anterior queda como:

$$\sum_{j=0}^{e} \sigma_j S_{i+e-j} = 0$$

es decir:

$$S_{i+e} + S_{i+e-1}\,\sigma_1 + \ldots + S_{i+1}\,\sigma_{e-1} + S_i\,\sigma_e = 0, \quad i \in [1,2t-e]. \qquad (3.16)$$

En la primera etapa del algoritmo los l_n primeros valores de $\sigma_j^{(n)}$ son determinados a partir de los primeros n componentes del síndrome $(S_1, S_2, \ldots, S_n)$, $n \in [1,2t]$, explícitamente las ecuaciones son:

$$
\begin{aligned}
&S_n + S_{n-1}\,\sigma_1^{(n)} + \ldots + S_{n-l_n}\,\sigma_{l_n}^{(n)} = 0\\
&S_{n-1} + S_{n-2}\,\sigma_1^{(n)} + \ldots + S_{n-l_n-1}\,\sigma_{l_n}^{(n)} = 0\\
&S_{n-2} + S_{n-3}\,\sigma_1^{(n)} + \ldots + S_{n-l_n-2}\,\sigma_{l_n}^{(n)} = 0\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&S_{l_n+1} + S_{l_n}\,\sigma_1^{(n)} + \ldots + S_1\,\sigma_{l_n}^{(n)} = 0
\end{aligned}
\qquad (3.17)
$$

donde $\sigma_i^{(n)}$, $i \in [1, l_n]$ son los coeficientes del polinomio localizador de errores en la etapa n, que está dado por:

$$\sigma(x)^{(n)} = \sigma_0^{(n)} + \sigma_1^{(n)} x + \ldots + \sigma_{l_n}^{(n)} x^{l_n}, \quad \sigma_0^{(n)} = 1 \qquad (3.18)$$

La siguiente etapa consiste en encontrar el polinomio $\sigma(x)^{(n+1)}$ de grado mínimo que cumple con las ecuaciones:

$$\sum_{j=0}^{l_{n+1}} S_{i-j}\,\sigma_j^{(n+1)} = 0, \qquad i = l_{n+1}+1, \ldots, n+1 \qquad (3.19)$$

Se define la discrepancia en la etapa n como:

$$d_n = S_{n+1} + S_n\,\sigma_1^{(n)} + \ldots + S_{n-l_n+1}\,\sigma_{l_n}^{(n)} \qquad (3.20)$$

y sirve para determinar el valor del polinomio de la siguiente etapa. Se tienen dos casos:

a) Si $d_n=0$, entonces las ecuaciones (3.19) se cumplen para:

$$
\begin{aligned}
&\sigma(x)^{(n+1)} = \sigma(x)^{(n)}\\
&l_{n+1} = l_n
\end{aligned}
\qquad (3.21)
$$

es decir, la solución de la etapa siguiente es la misma que la de la anterior.

b) Si $d_n \neq 0$, el cálculo de $\sigma(x)^{(n+1)}$ a partir de $\sigma(x)^{(n)}$ se efectúa de la siguiente forma: Si $\sigma(x)^{(n)}$ es la solución mínima de la etapa n y $\sigma(x)^{(m)}$ es la solución de la etapa m, con $-1 \leq m < n$ para $d_m \neq 0$, tal que $(m-l_m)$ sea máxima; entonces la solución mínima de la etapa n+1 es:

$$\sigma(x)^{(n+1)} = \sigma(x)^{(n)} - d_n d_m^{-1} x^{(n-m)} \sigma(x)^{(m)}$$

$$l_{n+1} = \max \left\{ l_n,\ l_m + n - m \right\} \qquad (3.22)$$

con $-1 \leq m < n$, $d_m \neq 0$ y $(m-l_m)$ máximo

El cálculo de $\sigma(x)$ continúa hasta cumplir la condición:

$$n \geq l_{n+1} + t - 1 \qquad (3.23)$$

Las condiciones iniciales del algoritmo son:

$$\sigma(x)^{(-1)} = 1 \qquad l_{-1} = 0 \qquad d_{-1} = 1$$
$$\sigma(x)^{(0)} = 1 \qquad l_0 = 0 \qquad d_0 = S_1 \qquad (3.24)$$

Para ilustrar el algoritmo de Berlekamp-Massey, considerando el ejemplo planteado, se tienen los síndromes:

$$S_1 = \alpha \qquad S_2 = \alpha^2$$
$$S_3 = \alpha^8 \qquad S_4 = \alpha^4$$
$$S_5 = 1 \qquad S_6 = \alpha$$

Las condiciones iniciales son:

$$\sigma(x)^{(-1)} = 1 \qquad l_{-1} = 0 \qquad d_{-1} = 1$$
$$\sigma(x)^{(0)} = 1 \qquad l_0 = 0 \qquad d_0 = S_1 = \alpha$$

ETAPA 1. $n = 0$ $m = -1$ que maximiza $(-1+0) = -1$ para $d_{-1} \neq 0$.

$$\sigma(x)^{(1)} = \sigma(x)^{(0)} - d_0 d_{-1}^{-1} x^{[0-(-1)]} \sigma(x)^{(-1)}$$

$$= 1 - \alpha(1)^{-1} x(1)$$

$$= 1 + \alpha x$$

$$l_1 = \max \{l_0, l_{-1} + 0 - (-1)\} = 1$$

$$l_1 + 3 - 1 \leq 0 \quad ? \text{ No.}$$

$$d_1 = S_2 + S_1 \sigma_1^{(1)} = \alpha^2 + \alpha(\alpha) = 0$$

ETAPA 2. $n = 1, \quad d_1 = 0$

$$\sigma(x)^{(2)} = \sigma(x)^{(1)} = 1 + \alpha x$$

$$l_2 = l_1 = 1$$

$$l_2 + 3 - 1 \leq 1 \quad ? \text{ No.}$$

$$d_2 = S_3 + S_2 \sigma_1^{(1)} = \alpha^8 + \alpha^2(\alpha) = \alpha^{13}$$

ETAPA 3. $n = 2, \quad m = 0$ maximiza $(0-0) = 0$ para $d_0 \neq 0$.

$$\sigma(x)^{(3)} = \sigma(x)^{(2)} - d_2 d_0^{-1} x^{(2-0)} \sigma(x)^{(0)}$$

$$= (1+\alpha x) - \alpha^{13}(\alpha)^{-1} x^2(1)$$

$$= 1 + \alpha x + \alpha^{12} x^2$$

$$l_3 = \max \{l_2, l_0 + 2 - 0\} = 2$$

$$l_3 + 3-1 \leq 2 \quad ? \text{ No.}$$

$$d_3 = S_4 + S_3 \sigma_1^{(3)} + S_2 \sigma_2^{(3)}$$

$$= \alpha^4 + \alpha^8(\alpha) + \alpha^2(\alpha^{12}) = 0$$

ETAPA 4. $n = 3$, $d_3 = 0$

$$\sigma(x)^{(4)} = \sigma(x)^{(3)} = 1 + \alpha x + \alpha^{12} x^2$$

$$l_4 = l_3 = 2$$

$l_4 + 3 - 1 \leq 3$? No.

$$d_4 = S_5 + S_4 \sigma_1^{(4)} + S_3 \sigma_2^{(4)} = 1 + \alpha^4(\alpha) + \alpha^8(\alpha^{12})$$

$$= 1$$

ETAPA 5. $n = 4$, $m = 2$ maximiza $(2-1) = 1$ para $d_2 \neq 0$.

$$\sigma(x)^{(5)} = \sigma(x)^{(4)} - d_4 d_2^{-1} x^{(4-2)} \sigma(x)^{(2)}$$

$$= (1+\alpha x+\alpha^{12}x^2) - (1)(\alpha^{13})^{-1} x^2 (1+\alpha x)$$

$$= 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$$

$$l_5 = \max \{l_4, l_2 + 4 - 2\} = 3$$

$l_5 + 3 - 1 \leq 4$? No.

$$d_5 = S_6 + S_5 \sigma_1^{(5)} + S_4 \sigma_2^{(5)} + S_3 \sigma_3^{(5)}$$

$$= \alpha + 1(\alpha) + \alpha^4(\alpha^7) + \alpha^8(\alpha^3) = 0$$

ETAPA 6. $n = 5$, $d_5 = 0$

$$\sigma(x)^{(6)} = \sigma(x)^{(5)} = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$$

$$l_6 = l_5 = 3$$

$l_6 + 3 - 1 \leq 5$? Sí : Termina el algoritmo.

Por lo tanto, el polinomio localizador de errores es:

$$\sigma(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$$

Sin embargo, en el caso binario se pueden efectuar la mitad de las etapas usadas en el caso general. Como puede observarse del ejemplo anterior, las discrepancias para n impar fueron nulas; por lo tanto puede incrementarse a n en dos unidades de la vez. En este caso, el algoritmo se termina cuando se cumple la condición $n \geq l_{n+2} + t - 2$.

Para el ejemplo en cuestión se tienen entonces las siguientes etapas:

ETAPA 1. $n=0$, $m=-1$

$$\sigma(x)^{(2)} = \sigma(x)^{(0)} - d_0 d_{-1}^{-1} x^{(0-(-1))} \sigma(x)^{(-1)}$$
$$= 1 + \alpha x$$
$$l_2 = \max\{l_0, l_1+0-(-1)\} = 1$$
$$l_2 + 3 - 2 \leq 0 \ ? \ \text{No.}$$
$$d_2 = S_3 + S_2 \sigma_1^{(2)} = \alpha^8 + \alpha^2(\alpha) = \alpha^{13}$$

ETAPA 2. $n=2$, $m=0$

$$\sigma(x)^{(4)} = \sigma(x)^{(2)} - d_2 d_0^{-1} x^{(2-0)} \sigma(x)^{(0)}$$
$$= 1 + \alpha x + \alpha^{12} x^2$$
$$l_4 = \max\{l_2, l_0+2-0\} = 2$$
$$l_4 + 3 - 2 \leq 2 \ ? \ \text{No.}$$
$$d_4 = S_5 + S_4 \sigma_1^{(4)} + S_3 \sigma_2^{(4)} = 1 + \alpha^4(\alpha) + \alpha^8(\alpha^{12}) = 1$$

ETAPA 3. $n=4$, $m=2$

$$\sigma(x)^{(6)} = \sigma(x)^{(4)} - d_4 d_2^{-1} x^{(4-2)} \sigma(x)^{(2)}$$
$$= 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$$
$$l_6 = \max\{l_4, l_2 + 4 - 2\} = 3$$
$$l_6 + 3 - 2 \leq 4 \ ? \ \text{Si : Termina el algoritmo.}$$

iii) Evaluación de las raíces de $\sigma(x)$. [10,24,31].

Para encontrar la posición de los errores en la palabra recibida, es utilizado generalmente el método de la búsqueda de Chien. Este consiste en evaluar el polinomio localizador de errores, $\sigma(x)$, para $x=\alpha^{-i}$, donde $i=0, 1, , \ldots, n-1$. Si $\sigma(\alpha^{-i})=0$ entonces i representa la posición del error en la palabra recibida de longitud n.

Mediante este método es posible la detección del evento: imposibilidad de decodificación. Este es el caso cuando el número de raíces es inferior al grado de $\sigma(x)$, i.e. cuando las raíces son repetidas o cuando no pertenecen al campo finito CG(q).

En el ejemplo se tiene que:

$$\sigma(\alpha^{\circ}) = 1 + \alpha + \alpha^{7} + \alpha^{3} = 0$$
$$\sigma(\alpha^{-6}) = 1 + \alpha^{-5} + \alpha^{-5} + \alpha^{-15} = 0$$
$$\sigma(\alpha^{-12}) = 1 + \alpha^{-11} + \alpha^{-17} + \alpha^{-33} = 0$$

y por lo tanto $e(x) = 1 + x^{6} + x^{12}$.

iv) Evaluación del valor de los errores.

En general el valor de los errores se obtiene mediante el método de Forney, usando la fórmula [11,31].

$$Y_j = \frac{\sum_{i=0}^{e-1} (-1)^i \sigma_{ji} S_{e-i}}{\sum_{i=0}^{e-1} (-1)^i \sigma_{ji} X_j^{e-i}} \quad , \quad j = 1, \ldots, e, \quad e \leq t \qquad (3.25)$$

siendo e el número de errores.

v) Corrección de la palabra recibida.

En el caso binario, $Y_j = 1$, $j=1,\ldots, e$. Así, para finalizar el ejemplo,

el decodificador simplemente sumaría el patrón de error a la palabra recibida obteniendo la palabra corregida $C(x)$, esto es:

$$C(x) = r(x) + e(x) = (1 + x + x^2 + x^3 + x^4 + x^6 + x^8 + x^{11} + x^{14}) + (1 + x^6 + x^{12})$$
$$= x + x^2 + x^3 + x^4 + x^8 + x^{11} + x^{12} + x^{14}.$$

Este ejemplo fué simulado en computadora, presentándose los resultados a continuación.

```
.RUN DK1:BCH

  PROGRAMA SIMULADOR DE UN CODIGO BCH.

  ORDEN MULTIPLICATIVO DE CG(2**M1), M1= 4

  LONGUITUD DEL CODIGO, N= 15

  POLINOMIO GENERADOR DE CG(2**M1), G(X)?
11001

   1+ 1*X** 1+ 0*X** 2+ 0*X** 3+ 1*X** 4+

  CAPACIDAD DEL CODIGO, T= 3

  NUMERO NOMINAL DE BITS DE INFORMACION, KNA= 5

  POLINOMIO GENERADOR DEL CODIGO, GP(X)?
11101100101

   1+ 1*X** 1+ 1*X** 2+ 0*X** 3+ 1*X** 4+ 1*X** 5+ 0*X** 6+ 0*X** 7+ 1*X** 8+ 0*X** 9+ 1*X**10


  ---------------------
  CODIGO BCH (15, 5, 7)
  ---------------------

  INFORMACION A CODIFICAR, INFO(X)?
01101

   0+ 1*X** 1+ 1*X** 2+ 0*X** 3+ 1*X** 4+

  PALABRA CODIFICADA, CODE(X):

011110001001101

   0+ 1*X** 1+ 1*X** 2+ 1*X** 3+ 1*X** 4+ 0*X** 5+ 0*X** 6+ 0*X** 7+ 1*X** 8+ 0*X** 9+ 0*X**10
 1*X**11+ 1*X**12+ 0*X**13+ 1*X**14+
```

Listado 3.1 Resultados de la simulación del código BCH (15,5,7).

```
 # DE OPERACIONES EN LA CODIFICACION =      6.2200000000E+02

 ERRORES, ERROR(X)?
100000100000100

  1+ 0*X** 1+ 0*X** 2+ 0*X** 3+ 0*X** 4+ 0*X** 5+ 1*X** 6+ 0*X** 7+ 0*X** 8+ 0*X** 9+ 0*X**10
0*X**11+ 1*X**12+ 0*X**13+ 0*X**14+

 PALABRA RECIBIDA, RECU(X):

111110101001001

  1+ 1*X** 1+ 1*X** 2+ 1*X** 3+ 1*X** 4+ 0*X** 5+ 1*X** 6+ 0*X** 7+ 1*X** 8+ 0*X** 9+ 0*X**10
1*X**11+ 0*X**12+ 0*X**13+ 1*X**14+

 SINDROMES DE LA PALABRA RECIBIDA:

 SINDROME     RENGLON

      1           3
      2           4
      3          10
      4           6
      5           2
      6           3

 ITERACION #  1

 POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):

   2    3    1    1

   2+  3*X** 1+  1*X** 2+  1*X** 3+

 GRADO=   1   PARA NI=   2

 DISCREPANCIA=  15


 ITERACION #  2
```

Listado 3.1 Resultados de la simulación del código BCH (15,5,7) (cont.).

```
  POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):

    2     3    14     1

    2+  3*X** 1+ 14*X** 2+  1*X** 3+

  GRADO=    2    PARA NI=    4

  DISCREPANCIA=    2

  ITERACION #  3

  POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):

    2     3     9     5

    2+  3*X** 1+  9*X** 2+  5*X** 3+

  GRADO=    3    PARA NI=    6

  RAICES DE SIGMA(X):

  RAIZ( 1)=  1
  RAIZ( 2)=  7
  RAIZ( 3)= 13

  DECODIFICACION EXITOSA:

011110001001101

   0+ 1*X** 1+ 1*X** 2+ 1*X** 3+ 1*X** 4+ 0*X** 5+ 0*X** 6+ 0*X** 7+ 1*X** 8+ 0*X** 9+ 0*X**10
 1*X**11+ 1*X**12+ 0*X**13+ 1*X**14+

  # DE OPERACIONES EN LA DECODIFICACION =      8.9944000000E+04
```

Listado 3.1 Resultados de la simulación del código BCH (15,5,7) (Cont.).

La decodificación de errores, en el caso binario [24], puede efectuarse mediante registros de corrimiento y una unidad de cálculo como lo muestra el diagrama de la figura 3.1. En la primera etapa del decodificador se evalúan los componentes del síndrome, S_i, al dividir la palabra recibida por $M^{(i)}(x)$, para $i=1, 3, \ldots, 2t-1$. La evaluación de $S_{2i}=S_i^2$ se logra calculando el residuo de $r(\alpha^{2i})$, al haber dividido la palabra recibida entre $M^{(i)}(x)$.

En la segunda etapa, la más compleja de todas, se requiere de una unidad aritmética que trabaje en $CG(2^m)$, para la implementación del algoritmo de Berlekamp-Massey. La última etapa se logra con $t+1$ registros de m bits, inicialmente cargados con los valores de $1, \sigma_1, \ldots, \sigma_{t_2}$ respectivamente. Además los registros $2, 3, \ldots, t+1$ multiplican por $\alpha, \alpha^2, \ldots, \alpha^t$ e incrementan las potencias en cada ciclo, esta mecánica se lleva a cabo sincronizadamente a la salida de los bits de la palabra recibida. Al salir un cero de esta etapa, significa que se obtuvo una raíz del polinomio $\sigma(x)$, la posición del bit corresponde a un error, y la palabra es corregida.

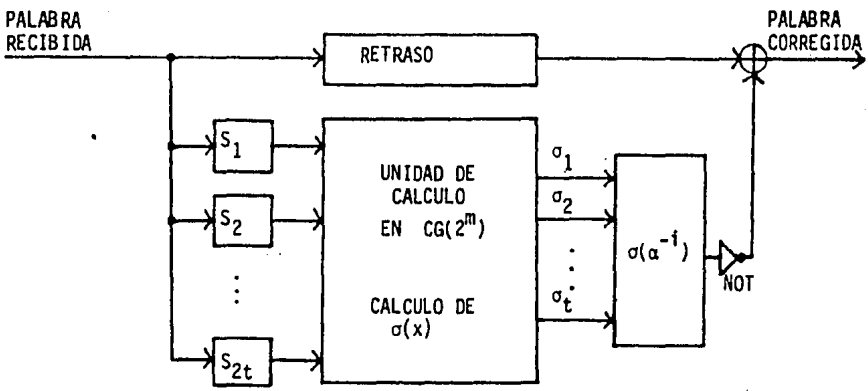


Figura 3.1 Diagrama de bloques del decodificador de Códigos de B.C.H.

3.3.1 DECODIFICACION DE ERRORES Y BORRADOS.

Hasta este punto, se ha tratado sobre la corrección de errores, sin embargo, el método de decodificación anterior puede modificarse para corregir borra

dos. Si el código tiene una distancia mínima d, entonces se cumple

$$d \geq 2e + b + 1 \qquad (3.26)$$

donde e es el número de errores y b el número de borrados. El código puede corregir e errores y b borrados si su distancia mínima cumple con (3.26).

Antes de continuar es necesario definir el concepto de borrado. Si el decodificador cuenta con una regla de decisión que le indique que recibe un nivel lógico alto si el nivel de voltaje de la señal es superior a cierto umbral V_1, y que recibe un nivel lógico bajo si el nivel es inferior a otro umbral V_0, entonces si la señal tiene un nivel V_b, $V_0 \leq V_b \leq V_1$, el demodulador detecta la presencia de un borrado y transfiere esta información al decodificador.

Si en el canal binario simétrico son considerados los borrados, éste luce como se muestra en la figura 3.2. Para este canal, las posiciones de los símbolos borrados son conocidas, y este hecho facilita su corrección. De hecho, de acuerdo a 3.26 , el código proporciona el doble de redundancia para la corrección de errores que para la de borrados.

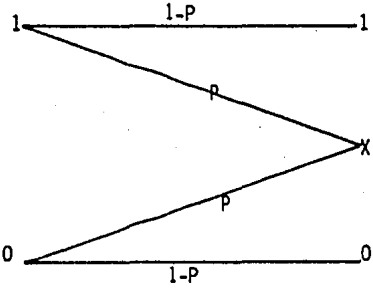


Figura 3.2 Canal binario simétrico con borrados.

Para incluir la capacidad de corrección de borrados en el esquema de decodificación, [24,30,11], es necesario incluir un polinomio localizador de borrados definido como:

$$\tau(x) = \prod_{j=1}^{b} (x-X_j) = \sum_{j=0}^{b} (-1)^j \tau_j x^{b-j}, \quad \tau_o=1 \tag{3.27}$$

una vez calculado este polinomio, se calcula un síndrome modificado, conocido como el síndrome de Forney y definido como:

$$T(x) = [(1+ S(x))\,\tau'(x)] + 1 \quad \text{módulo} \quad x^d \tag{3.28}$$

$$= \sum_{i=1}^{d-1} T_i x^i$$

Donde:
$$S(x) = \sum_{j=1}^{2t} S_j x^j \tag{3.29}$$

y
$$\tau'(x) = \sum_{j=0}^{b} \tau_j x^j \tag{3.30}$$

debe observarse que $\tau'(x)$ consta de los mismos coeficientes de $\tau(x)$ con el orden invertido.

El algoritmo de Berlekamp-Massey por su parte, sufre las siguientes modificaciones: La discrepancia de la etapa n es ahora:

$$d_n = T_{n+b+1} + T_{n+b}\,\sigma_1^{(n)} + \ldots + T_{n+b+1-l_n}\,\sigma_{l_n}^{(n)} \tag{3.31}$$

(con $d_0 = T_{b+1}$).

y el algoritmo termina cuando:

$$n \geq l_{n+1} + t - 1 - b/2 \tag{3.32}$$

Una vez que se ha calculado $\sigma(x)$, se procede a calcular el polinomio evaluador de erratas (i.e. errores y borrados), definido como:

$$\Omega(x) = [1 + T(x)]\,\sigma(x) \quad \text{módulo} \quad x^d \tag{3.33}$$

así, mediante los componentes Ω_i, $i=0, 1, \ldots, e+b-1$, se calcula el valor de

las erratas como:

$$Y_j = \frac{X_j^{(e+b-1)}\Omega(X_j^{-1})}{\prod_{j\neq i}(X_j - X_i)}$$

$$= \frac{X_j^{(e+b-1)} \sum_{i=0}^{e+b-1} \Omega_i X_j^{-i}}{\prod_{j\neq i}(X_j - X_i)}, \quad j=1, 2, \ldots, e+b \qquad (3.34)$$

Para el caso binario (q=2) el valor de las erratas es $Y_j=1$ ó $Y_j=0$. Una estrategia empleada [31] es suponer que los b borrados ocurridos tienen valor nulo y decodificar así la palabra recibida. Posteriormente, se complementan los valores y se vuelve a decodificar.

Suponiendo que S $(0 \leq S \leq b)$ de los valores iniciales supuestos fueron correctos, entonces b-S de los complementos serán incorrectos. Debido a que $\min\{S, b-S\} \leq b/2$, en uno de los dos casos el número total de errores no será mayor que $(d-1)/2$, y por lo tanto el patrón de error podrá ser corregido. En el otro caso, un total de $e + \max\{S, b-S\}$ errores deberán corregirse. Si ocurre que este número no excede de $(d-1)/2$, entonces se tendrá una decodificación exitosa. De otra manera el número total de errores sobrepasará la capacidad de corrección del código; y ya que $d > 2t+b$, en la decodificación no se obtendrá una palabra de código válida.

En resumen, la decodificación de errores y borrados usando un código de BCH se lleva a cabo ejecutando las siguientes etapas:

i) Calcular el síndrome, S_i, $i = 1, 2t$.

ii) Si hubo borrados, calcular el polinomio localizador de borrados, $\tau(x)$, y el síndrome de Forney, $T(x)$. En caso contrario, el síndrome de Forney $T(x)$ es igual al síndrome calculado en i).

iii) Calcular el polinomio localizador de errores, $\sigma(x)$, mediante el algoritmo de Berlekamp-Massey.

iv) Obtener las raíces de $\sigma(x)$.

v) Calcular el polinomio evaluador de erratas, $\Omega(x)$.

vi) Calcular el valor de las erratas, Y_j, $j = 1, \ldots, e+b$.

vii) Corregir la palabra recibida.

En el siguiente capítulo se verá que existen otros esquemas de decodificación para códigos de BCH, que ofrecen algunas ventajas con respecto a lo visto hasta aquí.

3.4 CODIGOS ACORTADOS DE B.C.H. [2,24,31].

Como en los códigos analizados anteriormente, si en un código BCH los primeros i_{acor} símbolos son desechados, se obtiene un código acortado de BCH. Los parámetros del nuevo código son (n', k', d'), donde $n'=q^m-1-i_{acor}$, $k'=n'-r$ y $d'\geq d$, con $r\leq mt$ y d es la distancia diseñada del código original. Donde $r=n-k$.

Para codificar información mediante un código acortado de BCH, utilizando el codificador original, se hacen nulos los primeros i_{acor} símbolos.

En la decodificación, los síndromes se calculan considerando n' símbolos, ya que i_{acor} de los n símbolos originales son nulos.

El algoritmo de Berlekamp-Massey no sufre ninguna modificación, ya que depende únicamente de la distancia diseñada del código original. Sin embargo, debido a que la distancia del código acortado, es por lo menos igual a la del código original, es posible que la capacidad de corrección de errores del código sea mayor que la original, en cuyo caso el algoritmo deberá sufrir modificaciones.

La localización de errores, búsqueda de Chien, puede efectuarse con n' elementos de $CG(q^m)$, ya que i_{acor} elementos son nulos.

Resumiendo, los códigos acortados presentan la ventaja de tener una distancia por lo menos igual a la del código del que provienen. Además, en gene

ral, el tiempo de decodificación se reduce. Si se emplea el esquema del código original, los códigos acortados tienen la ventaja de poder implementar varios códigos de parámetros diferentes mediante el mismo esquema.

CAPITULO 4

CODIGOS DE REED-SOLOMON

CAPITULO 4
CODIGOS DE REED-SOLOMON

4.1 INTRODUCCION. [2,24,31].

Un código de Reed-Solomon (o código RS) sobre un CG(q) es un código BCH de longitud $n=q-1$, donde $q=p^m$, p es un número primo y m un entero.

Si se tiene una distancia diseñada d, el código RS tendrá como raíces $\alpha^v, \alpha^{v+1}, \ldots, \alpha^{v+d-2}$, donde α es un elemento del CG(q) de orden n, i.e. $\alpha^n=1$. Puesto que el polinomio mínimo $M^{(i)}(x)$ que contiene a α^i como raíz es simplemente $x-\alpha^i$, el polinomio generador del código G(x) y el polinomio de paridad H(x) son respectivamente:

$$G(x) = \prod_{i=v}^{v+d-2} (x-\alpha^i), \quad H(x) = \prod_{i=v+d-1}^{n+v-1} (x-\alpha^i) \qquad (4.1)$$

donde: $G(x)\,H(x) = x^n-1$

Para v=1, que es el caso generalmente empleado, se observa claramente que el grado de G(x) es d-1, es decir, habrá n-k=d-1 símbolos de redundancia en la palabra de código, de donde se obtiene que los códigos RS son de distancia máxima de separación.

La importancia de los códigos de Reed-Solomon puede resumirse en los siguientes puntos:

i) Son muy útiles en la corrección de paquetes múltiples de error.
ii) Son los códigos propiamente empleados cuando se requiere un código de longitud menor que el tamaño del campo.
iii) Son de distancia máxima de separación, i.e. d=n-k+1
iv) Pueden ser mapeados en códigos binarios obteniéndose una distancia mínima sorprendentemente alta.
v) Se emplean para construir códigos concatenados.
vi) Para una longitud n y una capacidad de corrección de error t fijas, un código RS requiere de un campo de Galois más pequeño que

un código BCH con las mismas n y t. Se ha demostrado que en muchas situaciones las versiones binarias de los códigos RS tienen una probabilidad de error menor que los códigos BCH con parámetros equivalentes.

4.2 CODIFICACION. [31].

Por ser los códigos de Reed-Solomon un caso particular de los BCH, su codificación y su decodificación pueden realizarse empleando los mismos algoritmos que en el capítulo anterior.

A continuación se verá un ejemplo de corrección de errores y borrados en el cual los símbolos de información pueden considerarse como paquetes de m símbolos binarios cada uno.

Sea un $CG(2^4)$ con α un elemento primitivo de orden $n = 2^4-1=15$, y sea $g(x) = x^4+x^3+1$ el polinomio primitivo que genera dicho campo.

Para una distancia $d=7$, como en el caso de los códigos BCH, párrafo 3.3.1, se podrán corregir e errores y b borrados tales que $d \geq 2e + b + 1$.

De la ecuación (4.1) para v=1 se tiene:

$$H(x) = \prod_{i=d}^{n} (x-\alpha^i) = \prod_{i=7}^{15} (x-\alpha^i) = (x-\alpha^7)(x-\alpha^8) \ldots (x-\alpha^{15})$$

$$H(x) = x^9+\alpha^{12}x^8+\alpha^2x^7+\alpha^5x^6+\alpha^5x^5+\alpha x^4+\alpha^8x^3+\alpha^{12}x^2+\alpha^{14}x+\alpha^9$$

En este caso $n-k=d-1=7-1=6$, por lo tanto se trata de un código RS(15,9,7).

Nota: Puesto que el $CG(2^4)$ es una extensión del CG(2) con característica p=2, se pueden emplear indiferentemente los signos "+" y "-" en las operaciones dentro del campo.

Supongamos el siguiente polinomio de información:

$$i(x) = \alpha+\alpha^4x+\alpha^6x^2+\alpha^3x^4+\alpha^5x^5+\alpha^6x^6+\alpha^{13}x^7+\alpha^{10}x^8$$

empleando la codificación por recurrencia con $H(x)$, (ecuaciones de paridad (1.22)):

$$C_j = -\sum_{i=0}^{k=1} h_i\, C_{k-i+j} \quad j = 0, 1, \ldots, n-k-1$$

se obtiene:

$$C(x) = \alpha^5+\alpha^3x+\alpha^{13}x^2+\alpha x^3+\alpha^7x^4+\alpha^4x^5+\alpha x^6+$$

$$+ \alpha^4x^7+\alpha^6x^8+\alpha^3x^{10}+\alpha^5x^{11}+\alpha^6x^{12}+\alpha^{13}x^{13}+\alpha^{10}x^{14}$$

4.3 DECODIFICACION. [5,17,24,31,32].

Se desarrollará a continuación la decodificación del ejemplo anterior empleando el esquema descrito en la sección 3.3 para el caso de errores y borrados.

Supóngase que ocurrieron dos errores y dos borrados en $C(x)$, dados por:

polinomio de errores: $E(x) = \alpha^8x^3+\alpha x^{12}$

polinomio de borrados: $B(x) = \alpha^{14}+\alpha^5x^5$

que juntos definen el polinomio de erratas:

$$u(x) = E(x) + B(x) = \alpha^{14} + \alpha^8x^3 + \alpha^5x^5 + \alpha x^{12}$$

de ésta forma la palabra recibida queda dada por:

$$r(x) = C(x) + u(x) = \alpha^7+\alpha^3x+\alpha^{13}x^2+\alpha^{14}x^3+\alpha^7x^4+\alpha x^5+\alpha x^6+$$

$$+ \alpha^4x^7+\alpha^6x^8+\alpha^3x^{10}+\alpha^5x^{11}+\alpha^{11}x^{12}+\alpha^{13}x^{13}+\alpha^{10}x^{14}$$

de la cual sólo se conoce en el decodificador el número de borrados ocurridos y su posición.

En el ejemplo planteado siendo 2 el número de borrados, en principio es posible realizar la corrección durante la decodificación.

Ahora, siguiendo las etapas planteadas en los párrados 3.3. y 3.3.1 se tiene:

i) Cálculo de los síndromes

$$S_i = r(\alpha^i) \qquad \text{para } i = 1, 2, \ldots, d-1$$

al evaluar el polinomio recibido se obtiene:

$$S_1 = r(\alpha) = \alpha^{11} \qquad S_4 = r(\alpha^4) = \alpha^2$$

$$S_2 = r(\alpha^2) = \alpha^5 \qquad S_5 = r(\alpha^5) = \alpha^0 = 1$$

$$S_3 = r(\alpha^3) = \alpha^2 \qquad S_6 = r(\alpha^6) = \alpha^{14}$$

con éstos síndromes se define la función generadora:

$$S(x) = \sum_{j=1}^{d-1} S_j x^j = \alpha^{11}x + \alpha^5 x^2 + \alpha^2 x^3 + \alpha^2 x^4 + x^5 + \alpha^{14} x^6$$

ii) Síndromes de Forney

Se calculan mediante la ecuación (3.28):

$$T(x) = [1+S(x))\,\tau'(x)] + 1 \quad \text{módulo } x^d$$

Puesto que las posiciones X_j de los borrados son conocidas, dicho polinomio está dado por (3.27):

$$\tau(x) = (x-1)(x-\alpha^5) = x^2 + \alpha^{10}x + \alpha^5$$

y de la ecuación (3.30):

$$\tau'(x) = 1 + \alpha^{10}x + \alpha^5 x^2$$

en consecuencia:

$$T(x) = \left[1 + \alpha^{11}x + \alpha^{5}x^{2} + \alpha^{2}x^{3} + \alpha^{2}x^{4} + x^{5} + \alpha^{14}x^{6}\right]\left[1 + \alpha^{10}x + \alpha^{5}x^{2}\right] + 1 \text{ módulo } x^{7}$$

$$= \alpha^{7}x + \alpha^{6}x^{2} + \alpha^{7}x^{3} + \alpha^{11}x^{4} + \alpha^{9}x^{5} + x^{6} = \sum_{i=1}^{6} T_i x^i \qquad (4.2)$$

iii) Cálculo del polinomio localizador de errores $\sigma(x)$.

Como se explicó en el capítulo anterior, $\sigma(x)$ se puede calcular mediante el algoritmo de Berlekamp-Massey.

Para éste ejemplo la discrepancia enésima (ecuación 3.31) está dada por:

$$d_n = T_{n+3} + T_{n+2}\,\sigma_1^{(n)} + \ldots + T_{n+3-ln}\,\sigma_{ln}^{(n)} \qquad (4.3)$$

y las condiciones iniciales son:

$$\sigma(x)^{(-1)} = 1 \qquad l_{-1}=0 \qquad d_{-1}=1$$

$$\sigma(x)^{(0)} = 1 \qquad l_0=0 \qquad d_0=T_3 \qquad n=0$$

Paso 1. Se calcula:

para $d_n=0$ $\qquad \sigma(x)^{(n+1)} = \sigma(x)^{(n)}$

$$l_{n+1} = l_n$$

y para $d_n\neq 0$ $\qquad \sigma(x)^{(n+1)} = \sigma(x)^{(n)} - dn\, d_m^{-1}\, x^{(n-m)}\, \sigma(x)^{(m)}$

$$l_{n+1} = \text{máx}\,[l_n,\ l_m + n-m].$$

Paso 2. Se pregunta si para éste caso $n \geq l_{n+1} + 1$, si se cumple, el algoritmo termina. De otra forma se calcula d_{n+1} mediante (4.3), se incrementa n por uno y se regresa al Paso 1. Esta última desigualdad se ha obtenido sustituyendo los valores de t y b en la ecuación 3.32.

De la ecuación (4.2) tenemos:

$T_1=\alpha^7$ $\quad T_4=\alpha^{11}$

$T_2=\alpha^6$ $\quad T_5=\alpha^9$

$T_3=\alpha^7$ $\quad T_6=1$ $\quad$ y $\quad d_o=T_3=\alpha^7$

ETAPA 1. n=0, m=-1

$$\sigma(x)^{(1)} = \sigma(x)^{(0)} + d_o\, d_{-1}^{-1}\, x^{(0-(-1))}\, \sigma(x)^{(-1)}$$
$$= 1 + \alpha^7 x$$

$l_1 = \text{máx}\ \{0,0+0-(-1)\} = 1$

? es $n \geq l_{n+1} + 1$? No.

$d_1 = T_4 + T_3\, \sigma_1^{(1)} = \alpha^{11} + \alpha^7\alpha^7 = 1$

ETAPA 2. n=1, m=0

$$\sigma(x)^{(2)} = \sigma(x)^{(1)} + d_1\, d_o^{-1}\, \sigma(x)^{(0)}$$
$$= 1 + \alpha^4 x$$

$l_2 = \text{máx}\ \{1,0+1-0\} = 1$

? es $n \geq l_{n+1} + 1$? No.

$d_2 = T_5 + T_4\, \sigma_1^{(2)} = \alpha^9 + \alpha^{11}\alpha^4 = \alpha^2$

ETAPA 3. n=2, m=0

$$\sigma(x)^{(3)} = \sigma(x)^{(2)} + d_2\, d_n^{-1}\, x^{(2-0)}\, \sigma(x)^{(0)}$$
$$= 1 + \alpha^4 x + \alpha^{10} x^2$$

$l_3 = \text{máx}\ \{1,0+2-0\} = 2$

? es $n \geq l_{n+1} + 1$? No.

$d_3 = T_6 + T_5\, \sigma_1^{(3)} + T_4\sigma_2^{(3)} = 1 + \alpha^9\alpha^4 + \alpha^{11}\alpha^{10} = \alpha^3$

ETAPA 4. n=3, m=2

$$\sigma(x)^{(4)} = \sigma(x)^{(3)} + d_3\, d_2^{-1}\, x^{(3-2)}\, \sigma(x)^{(2)}$$
$$= 1 + \alpha^5 x + x^2$$

$l_4 = \max\{2, 1+3-2\} = 2$

? es $n \geq l_{n+1} + 1$? Sí.

el algoritmo termina y se tiene:

$$\sigma(x) = 1 + \alpha^5 x + x^2 \qquad (4.4)$$

iv) Obtención de las raíces de $\sigma(x)$.

Por medio de la busqueda de Chien, i.e. evaluando $\sigma(\alpha^{-i})$ para $i = 0, 1, \ldots, n-1 = 14$ se obtiene que:

$$\sigma(\alpha^{-3}) = 0 \qquad y \qquad \sigma(\alpha^{-12}) = 0$$

Por lo tanto las posiciones de los errores son:

$$X_1 = \alpha^3 \quad y \quad X_2 = \alpha^{12}$$

Las posiciones de los borrados, conocidas desde un principio, son:

$$X_3 = \alpha^0 = 1 \quad y \quad X_4 = \alpha^5$$

v) Cálculo del polinomio evaluador de erratas $\Omega(x)$

De la ecuación (3.33) tenemos:

$$\Omega(x) = [1 + T(x)]\ \sigma(x) \quad \text{módulo } x^7 \qquad (4.5)$$

sustituyendo (4.2) y (4.4) en (4.5) y realizando la multiplicación módulo x^7 se obtiene finalmente:

$$\Omega(x) = 1 + \alpha^{14}x + \alpha^{11}x^2 + \alpha^{11}x^3 + x^4$$

vi) Cálculo del valor de las erratas

Para este ejemplo el número de erratas es $e + b = 2 + 2 = 4$, por lo que

la ecuación (3.34) resulta:

$$Y_j = \frac{X_j^3 \, \Omega \, (X_j^{-1})}{\prod_{i \neq j} (X_j - X_i)} \qquad , \quad j = 1, 2, \ldots, 4$$

de donde:

$$Y_1 = \frac{\alpha^9 \, \Omega \, (\alpha^{-3})}{(\alpha^{12}+1)(\alpha^{12}+\alpha^5)(\alpha^5+1)} = \alpha^8$$

$$Y_2 = \frac{\alpha^6 \, \Omega \, (\alpha^{-12})}{(\alpha^3+1)(\alpha^3+\alpha^5)(\alpha^5+1)} = \alpha$$

$$Y_3 = \frac{\alpha^0 \, \Omega \, (\alpha^{-0})}{(\alpha^3+\alpha^{12})(\alpha^3+\alpha^5)(\alpha^5+\alpha^{12})} = \alpha^{14}$$

$$Y_4 = \frac{\alpha^0 \, \Omega \, (\alpha^{-5})}{(\alpha^3+\alpha^{12})(\alpha^3+1)(\alpha^{12}+1)} = \alpha^5$$

vii) Corrección de la palabra recibida

A partir de las parejas de valores (X_j, Y_j) $j = 1, \ldots, 4$, formamos el polinomio de erratas $\mu(x)$:

$$\mu(x) = \alpha^{14} + \alpha^8 x^3 + \alpha^5 x^5 + \alpha x^{12}$$

y la palabra de código es:

$$C(x) = r(x) + \mu(x) = \alpha^5 + \alpha^3 x + \alpha^{13} x^2 + \alpha x^3 + \alpha^7 x^4 + \alpha^4 x^5 +$$
$$+ \alpha x^6 + \alpha^4 x^7 + \alpha^6 x^8 + \alpha^3 x^{10} + \alpha^5 x^{11} + \alpha^6 x^{12} + \alpha^{13} x^{13} + \alpha^{10} x^{14}$$

Podemos observar que es la misma palabra de código que se tenía originalmente.

A continuación se presenta la simulación del ejemplo anterior realizada mediante el programa REEDS.

```
.RUN DK1:REEDS1

 PROGRAMA SIMULADOR DE UN CODIGO REED-SOLOMON.

 ORDEN MULTIPLICATIVO DE CG(2**M1), M1= 4

 LONGUITUD DEL CODIGO, N= 15

 POLINOMIO GENERADOR DE CG(2**M1), G(X)= 10011

  1+ 0*X** 1+ 0*X** 2+ 1*X** 3+ 1*X** 4+

 CAPACIDAD DEL CODIGO, T= 3

 POLINOMIO GENERADOR DEL CODIGO, GP(X):

    8    13     9     4     2    14     2

    8+ 13*X** 1+  9*X** 2+  4*X** 3+  2*X** 4+ 14*X** 5+  2*X** 6+

 POLINOMIO DE PARIDAD DEL CODIGO, H(X):

  11    16    14    10     3     7     7     4    14     2

  11+ 16*X** 1+ 14*X** 2+ 10*X** 3+  3*X** 4+  7*X** 5+  7*X** 6+  4*X** 7+ 14*X** 8+  2*X** 9+

 ---------------------------------
 CODIGO DE REED-SOLOMON (15, 9, 7)
 ---------------------------------

 INFORMACION A CODIFICAR, INFO(X)=  3 6 8 1 5 7 81512

    3+  6*X** 1+  8*X** 2+  1*X** 3+  5*X** 4+  7*X** 5+  8*X** 6+ 15*X** 7+ 12*X** 8+

 PALABRA CODIFICADA, CODE(X):

    7     5    15     3     9     6     3     6     8     1     5     7     8    15    12

    7+  5*X** 1+ 15*X** 2+  3*X** 3+  9*X** 4+  6*X** 5+  3*X** 6+  6*X** 7+  8*X** 8+  1*X** 9+  5*X**10
 7*X**11+  8*X**12+ 15*X**13+ 12*X**14+

 # DE OPERACIONES EN LA CODIFICACION =      3.7686000000E+04
```

Listado 4.1 Resultados de la simulación del código de Reed-Solomon (15,9,7).

```
NUMERO DE BORRADOS, B= 2
POSICION= 1
VALOR= 16
POSICION= 6
VALOR= 7

NUMERO DE ERRORES, E= 2
POSICION= 4
VALOR= 10
POSICION= 13
VALOR= 3

PALABRA RECIBIDA, RECU(X):

  9    5   15   16    9    3    3    6    8    1    5    7   13   15   12

  9+  5*X** 1+ 15*X** 2+ 16*X** 3+  9*X** 4+  3*X** 5+  3*X** 6+  6*X** 7+  8*X** 8+  1*X** 9+  5*X**10
7*X**11+ 13*X**12+ 15*X**13+ 12*X**14+

SINDROMES DE LA PALABRA RECIBIDA:

SINDROME      RENGLON

     1           13
     2            7
     3            4
     4            4
     5            2
     6           16

POLINOMIO LOCALIZADOR DE BORRADOS, TAO(X):

  2   12    7

  2+ 12*X** 1+  7*X** 2+

SINDROMES DE FORNEY, FORNEY(X):

  1    9    8    9   13   11    2

  1+  9*X** 1+  8*X** 2+  9*X** 3+ 13*X** 4+ 11*X** 5+  2*X** 6+
```

Listado 4.1 Resultados de la simulación del código de Reed-Solomon (15,9,7) (Cont.).

```
ITERACION #  1

POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):
  2    9    1    1
 2+  9*X** 1+  1*X** 2+  1*X** 3+

GRADO =   1   PARA NI=   2
DISCREPANCIA=   2

ITERACION #  2

POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):
  2    6    1    1
 2+  6*X** 1+  1*X** 2+  1*X** 3+

GRADO =   1   PARA NI=   3
DISCREPANCIA=   4

ITERACION #  3

POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):
  2    6   12    1
 2+  6*X** 1+ 12*X** 2+  1*X** 3+

GRADO =   2   PARA NI=   4
DISCREPANCIA=   5

ITERACION #  4

POLINOMIO LOCALIZADOR DE ERRORES, SIGMA(X):
  2    7    2    1
 2+  7*X** 1+  2*X** 2+  1*X** 3+

GRADO =   2   PARA NI=   5

RAICES DE SIGMA(X):

RAIZ( 1)=  4
RAIZ( 2)= 13

POLINOMIO LOCALIZADOR DE ERRATAS, OMEGA(X):

  2   16   13   13    2    1    1
```

Listado 4.1 Resultados de la simulación del código de Reed-Solomon (15,9,7) (Cont.)

```
  2+ 16*X** 1+ 13*X** 2+ 13*X** 3+  2*X** 4+  1*X** 5+  1*X** 6+

ERRATA   VALOR

   4       10
  13        3
   1       16
   6        7

DECODIFICACION EXITOSA!

  7    5   15    3    9    6    3    6    8    1    5    7    8   15   12

  7+  5*X** 1+ 15*X** 2+  3*X** 3+  9*X** 4+  6*X** 5+  3*X** 6+  6*X** 7+  8*X** 8+  1*X** 9+  5*X**10
7*X**11+  8*X**12+ 15*X**13+ 12*X**14+

# DE OPERACIONES EN LA DECODIFICACION =     1.9635800000E+05
```

Listado 4.1 Resultados de la simulación del código de Reed-Solomon (15,9,7) (ont.).

4.4 METODOS ALTERNATIVOS DE DECODIFICACION.

4.4.1 INTRODUCCION. [20,23,25,36,39].

Recientemente la conexión que se ha establecido entre la transformada sobre un campo de Galois y los códigos lineales de bloque, ha permitido simplificar significativamente la complejidad de los algoritmos de decodificación para los códigos BCH y RS. Más aún se ha mejorado la funcionalidad de los decodificadores con técnicas adicionales como lo son el uso de los algoritmos de la transformada rápida de Fourier (TRF) y de las fracciones continuadas, así como el empleo del teorema chino del residuo en combinación con el algoritmo de Winograd para desarrollar una técnica más rápida en el cálculo de los síndromes. La desventaja de los métodos de decodificación basados en la transformada es que la longitud n del código no puede ser escogida arbitrariamente, limitando así el empleo de éstas poderosas técnicas: la longitud del código debe ser tal, que el uso de los algoritmos de la TRF sean óptimos.

En la decodificación de códigos RS también pueden emplearse las fracciones continuadas y el algoritmo de la TRF sobre un campo finito $CG(F_n)$, donde F_n es un número primo de Fermat. Justesen propuso que éstas transformadas pueden ser empleadas para definir códigos RS y mejorar la eficiencia de su decodificación cuando $F_n = 2^{2^n} + 1$ para $n = 1, 2, 3$ y 4. Más tarde se extendió la transformada a campos finitos del tipo $I_{F_n}(\sqrt[8]{2})$, donde $\sqrt[8]{2}$ es una raiz del polinomio $P(x) = x^8 - 2$ sobre $CG(F_n)$ e I_{F_n} es el conjunto de enteros módulo F_n. Cuando se emplea ésta técnica el código RS tendrá una longitud de 2^{n+4}, para $n = 3$ y 4, y la aritmética empleada para realizar ésta transformada requiere únicamente de sumas enteras, corrimientos circulares y un pequeño número de multiplicaciones por potencias de $\sqrt[8]{2}$.

A continuación se dará la definición de transformada de campo finito haciendo una analogía con la transformada discreta de Fourier:

Sea $e = [e_0, e_1, \ldots, e_{n-1}]$ un vector en el "tiempo" sobre un $CG(q)$, donde n divide a q^m-1 para alguna m, y sea α un elemento de orden n sobre $CG(q^m)$.

La transformada de Fourier de campo finito del vector e, es el vector $E = [E_0, E_1, \ldots, E_{n-1}]$ sobre un $CG(q^m)$ donde E_j está dada por:

$$E_j = \sum_{i=0}^{n-1} e_i \alpha^{ij} \qquad j = 0, 1, \ldots, n-1 \tag{4.6}$$

y la transformada inversa está dada por:

$$e_i = \frac{1}{n \text{ módulo } p} \sum_{j=0}^{n-1} E_j \alpha^{-ij} \qquad i = 0, 1, \ldots, n-1 \tag{4.7}$$

: p es la característica del campo.

Nota: Para los códigos RS tenemos que m=1, de manera que los vectores e y E tienen componentes que están en el mismo campo.

Otro concepto necesario antes de describir los métodos alternativos de de codificación es la implementación del algoritmo de Berlekamp mediante el uso de fracciones continuadas en un campo finito [33,34].

Se parte de un polinomio T(x) del cual solamente son conocidos d-1-b coeficientes y se quiere encontrar su representación en función de dos polinomios $\Omega(x)$ y $\sigma(x)$ de manera que:

$$T(x) = \frac{\Omega(x)}{\sigma(x)}$$

La fórmula recursiva está dada por

$$T_n(x) = \frac{q_n(x)\,\Omega_{n-1}(x) - \Omega_{n-2}(x)}{q_n(x)\,\sigma_{n-1}(x) + \sigma_{n-1}(x)} = \frac{\Omega_n(x)}{\sigma_n(x)} \tag{4.8a}$$

donde

$\Omega_0(x) = 1$ $\qquad$ $\sigma_0(x) = 0$

$\Omega_{-1}(x) = 0$ $\qquad$ $\sigma_{-1}(x) = 1$ $\qquad$ e inicialmente n=1.

los cocientes parciales $q_n(x)$ pueden ser calculados mediante la siguiente fórmula recursiva:

$$r_{n-2}(x) = q_s(x)\, r_{n-1}(x) + r_n(x)$$

donde $r_{-1}(x) = T(x)$, y $r_o(x) = 1$, y donde $q_n(x)$ se obtiene como la "parte entera" del cociente $r_{n-2}(x)/r_{n-1}(x)$, siendo $r_n(x)$ el "residuo" dado por:

$$(-1)^n\, r_n(x) = \Omega_n(x) = \sigma_n(x)\, T(x).$$

$$q_n(x) = \left[r_{n-2}(x)/r_{n-1}(x) \right] \qquad (4.8b)$$

El algoritmo se termina cuando $r_n(x) = 0$, teniéndose finalmente:

$$T(x) = \frac{\Omega_n(x)}{\sigma_n(x)} = \frac{\Omega(x)}{\sigma(x)}$$

4.4.2 METODOS DE DECODIFICACION POR TRANSFORMADA PARA CODIGOS BCH Y RS EMPLEANDO FRACCIONES CONTINUADAS PARA ERRORES Y BORRADOS. [7,21,22,28,39].

Se presentarán dos enfoques diferentes. Primeramente sea n la longitud de un código RS de distancia diseñada d, sobre un $CG(q)$. Defínanse las siguientes palabras o vectores:

$C = C_o, C_1, \ldots, C_{n-1}$ = palabra de código.

$r = r_o, r_1, \ldots, r_{n-1}$ = palabra recibida.

$\mu = \mu_o, \mu_1, \ldots, \mu_{n-1}$ = vector de erratas

que se relacionan mediante:

$$C = r + \mu$$

Supóngase que ocurren e errores y b borrados en la palabra recibida r y que $b + 2e \leq d-1$. Si α^k es una raíz del polinomio generador del códi

go, entonces los síndromes están dados por:

$$S_k = \sum_{i=0}^{n-1} r_i \alpha^{ki} = \sum_{j=1}^{b+e} X_j Y_j^k \qquad k = 1, 2, \ldots, d-1 \tag{4.9}$$

donde si X_j es la posición de un borrado, Y_j es la diferencia entre el símbolo asignado y el símbolo transmitido. Si X_j es la posición de un error, Y_j es la diferencia entre el símbolo recibido y el símbolo transmitido.

La función generativa de la secuencia S_k está definida como la serie formal de potencias:

$$S(x) = \sum_{k=1}^{\infty} S_k x^{-k} \tag{4.10}$$

sustituyendo (4.9) en (4.10), se obtiene:

$$S(x) = \sum_{k=1}^{\infty} \sum_{j=1}^{b+e} X_j Y_j^k x^{-k} = \sum_{j=1}^{b+e} Y_j \sum_{k=1}^{\infty} (X_j x^{-1})^k =$$

$$= \sum_{j=1}^{b+e} Y_j \frac{X_j x^{-1}}{1-X_j x^{-1}} = \sum_{j=1}^{b+e} \frac{Y_j X_j}{x - X_j} =$$

$$= \frac{\sum_{j=1}^{b+e} X_j Y_j \left(\prod_{i \neq j} (x - X_i) \right)}{\prod_{j=1}^{b+e} (x - X_j)} \tag{4.11}$$

Defínanse los siguientes conjuntos:

$$\tau = \{X_j / X_j \text{ es la posición de un borrado}\}$$

y $$\sigma = \{X_j / X_j \text{ es la posición de un error}\}.$$

Defínanse también los siguientes polinomios:

polinomio localizador de errores:

$$\sigma(x) = \prod_{X_j \in \sigma} (x - X_j) \tag{4.12}$$

polinomio localizador de borrados:

$$\tau(x) = \prod_{X_j \in \tau} (x - X_j) \tag{4.13}$$

polinomio localizador de erratas (errores y borrados):

$$\rho(x) = \tau(x) \cdot \sigma(x) = \prod_{j=1}^{e+b} (x - X_j) \tag{4.14}$$

y el polinomio evaluador de erratas:

$$\Omega(x) = \sum_{j=1}^{e+b} X_j Y_j \left(\prod_{i \neq j} (x - X_i) \right) \tag{4.15}$$

Entonces la ecuación (4.11) la podemos escribir como:

$$S(x) = \frac{\sum_{j=1}^{e+b} X_j Y_j \left(\prod_{i \neq j} (x - X_i) \right)}{\prod_{j=1}^{e+b} (x - X_j)} = \frac{\sum_{j=1}^{e+b} X_j Y_j \left(\prod_{i \neq j} (x - X_i) \right)}{\prod_{X_j \in \tau} (x - X_j) \prod_{X_j \in \sigma} (x - X_j)} = \frac{\Omega(x)}{\tau(x)\ \sigma(x)} \tag{4.16}$$

de donde: $$S(x)\ \tau(x) = \frac{\Omega(x)}{\sigma(x)} \tag{4.17}$$

El grado de $\tau(x)$ es b, por lo que (4.17) se puede representar como:

$$S(x)\ \tau(x) = \sum_{k=-b+1}^{\infty} T_{-k} x^{-k} \tag{4.18}$$

En la ecuación (4.18) se conocen $d-1-b$ coeficientes, donde $t \leq (d-1-b)/2$ por lo que $\Omega(x)$ y $\sigma(x)$ pueden obtenerse mediante las fracciones continuadas (ecuaciones (4.8)). Las raíces de $\sigma(x)$ se encuentran mediante la busqueda de Chien.

Sea X_k la posición de una errata. De la ecuación (4.15) tenemos:

$$\Omega(X_k) = Y_k X_k \prod_{i \neq k} (X_k - X_i) \tag{4.19}$$

de donde:

$$Y_k = \frac{\Omega(X_k)}{X_k \prod_{i \neq k} (X_k - X_i)} \tag{4.20}$$

derivando $\rho(x)$ con respecto a x en (4.14) obtenemos:

$$\rho'(x) = \sum_{j=1}^{e+b} \prod_{i \neq j} (x - X_i)$$

por lo tanto:

$$\rho'(x) = \sum_{j=1}^{e+b} \prod_{i \neq j} (X_k - X_i) = \prod_{i \neq k} (X_k - X_i) \quad k=1,2,\ldots,e+b$$

y finalmente la ecuación (4.20) resulta ser:

$$Y_k = \frac{\Omega(X_k)}{X_k \, \rho'(X_k)}$$

En resumen éste algoritmo de decodificación consta de las siguientes etapas:

i) Cálculo de la transformada de la palabra recibida a partir de la ecuación (4.9).

ii) Cálculo del polinomio localizador de borrados $\tau(x)$ mediante (4.13).

iii) Calculo de los coeficientes T_{-k}, k=-b+1.,...d-1-b, mediante (4.18).

iv) Si $b < d-1$ emplear las fracciones continuadas para determinar $\sigma(x)$ y $\Omega(x)$ a partir de los coeficientes T_k conocidos, de otra forma $\sigma(x) = 1$.

v) Hallar las posiciones de los errores (raíces de $\sigma(x)$) mediante la búsqueda de Chien.

vi) Calcular $\rho(x)$ mediante la ecuación (4.14) y calcular los valores de las erratas con la ecuación (4.21).

vii) Restar el vector de erratas μ, obtenido mediante las parejas de valores (X_j, Y_j), a la palabra recibida r para obtener la palabra de código C.

EJEMPLO. Sea un código RS sobre CG (8) con α un elemento de orden $n=8-1=7$, con una distancia $d=5$, por lo que se trata de un código RS(7,3,5). Sea $g(x) = x^3 + x + 1$ el polinomio generador del campo.

Supóngase la siguiente palabra de código:

$$C = [\alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha] .$$

y $\mu = [\alpha^4, 0, 0, 0, 0, 0, \alpha^2]$ con un error y un borrado (el borrado en $X_j = \alpha^6$), entonces: $r = C + \mu = [\alpha^2, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha^4]$.

Siguiendo el algoritmo anterior se tiene:

i) $S_k = \sum_{i=0}^{7-1} r_i \alpha^{ik}$ $\qquad k = 1, 2, \ldots, 4$

$S_1 = \alpha^2$ $\qquad S_3 = \alpha^3$

$S_2 = \alpha^5$ $\qquad S_4 = 1$

de (4.10): $S(x) = \alpha^2 x^{-1} + \alpha^5 x^{-2} + \alpha^3 x^{-3} + x^{-4} + ? \, x^{-5} + \ldots$

(solamente se conocen los primeros 4 componentes).

ii) de (4.13): $\tau(x) = (x-\alpha^6)$

iii) de (4.18), para $b-1$, al hacer el producto se obtiene:

$$S(x) \cdot \tau(x) = \alpha^2 + \alpha^6 x^{-1} + \alpha^6 x^{-2} + \alpha^6 x^{-3} + ? \, x^4 + \ldots = \sum_{k=0}^{\infty} T_{-k} x^{-k}$$

iv) en éste caso tenemos: $T(x) = \alpha^2 + \alpha^6 x^{-1} + \alpha^6 x^{-2} + \alpha^6 x^{-3} + \ldots$
de acuerdo a las ecuaciones (4.8):

$\sigma_{-1}(x) = 1 \qquad \Omega_{-1}(x) = 0 \qquad r_{-1}(x) = T(x)$
$\sigma_0(x) = 0 \qquad \Omega_0(x) = 1 \qquad r_0(x) = 1 \qquad n=1$ inicialmente.

$$\sigma_n(x) = q_n(x)\,\sigma_{n-1}(x) + \sigma_{n-2}(x)$$
$$\Omega_n(x) = q_n(x)\,\Omega_{n-1}(x) - \Omega_{n-2}(x)$$

con: $q_n(x) = \left[r_{n-2}(x)/r_{n-1}(x)\right]$

ETAPA 1. n=1

$$q_1(x) = [T(x)/1] = \alpha^2$$
$$r_1(x) = \alpha^6 x^{-1} + \alpha^6 x^{-2} + \alpha^6 x^{-3} + \ldots$$
$$\sigma_1(x) = \alpha^2(0) + 1 = 1$$
$$\Omega_1(x) = \alpha^2(1) - 0 = \alpha^2$$

ETAPA 2. n=2

$$q_2(x) = \left[1/(\alpha^6 x^{-1} + \alpha^6 x^{-2} + \alpha^6 x^{-3} + \ldots)\right]$$

$$\begin{array}{l|l} \alpha^6 x^{-1} + \alpha^6 x^{-2} + \alpha^6 x^{-3} + \ldots & \alpha x + \alpha \\ \hline & 1 + x^{-1} + x^{-2} + \ldots \\ \hline & x^{-1} + x^{-2} + \ldots \\ & x^{-1} + x^{-2} + \ldots \\ \hline & 0 + \ldots \end{array}$$

entonces: $q_2(x) = \alpha x + \alpha$

y $\qquad r_2(x) = 0 + \ldots$

$$\sigma_2(x) = (\alpha x + \alpha)(1) + 0 = \alpha x + \alpha$$
$$\Omega_2(x) = (\alpha x + \alpha)\,\alpha^2 + 1 = \alpha^3 x + \alpha$$

Terminando el algoritmo con:

$$T(x) = \frac{\Omega(x)}{\sigma(x)} = \frac{\alpha^3 x + \alpha}{\alpha x + \alpha} = \frac{\alpha^2 x + 1}{x + 1}$$

v) fácilmente vemos que el único error ocurrido está en $x_2=1$, no siendo necesaria en éste caso la búsqueda de Chien.

vi) de (4.14):

$$\rho(x) = \tau(x)\,\sigma(x) = (x+\alpha^6)(x+1) = x^2 + \alpha^2 x + \alpha^6$$

de donde:

$$\rho'(x) = 2x + \alpha^2 = x + x + \alpha^2 = \alpha^2$$

evaluando (4.21) para $X_1 = \alpha^6$:

$$Y_1 = \frac{\Omega(\alpha^6)}{\alpha^6\,\rho'(\alpha^6)} = \frac{\alpha^2\alpha^6 + 1}{\alpha^6\alpha^2} = \alpha^2$$

y para $X_2 = 1$:

$$Y_2 = \frac{\Omega(1)}{(1)\,\rho'(1)} = \frac{\alpha^2 + 1}{\alpha^2} = \alpha^4$$

vii) mediante las parejas $(1,\alpha^4)$ y (α^6,α^2) formamos el vector de erratas:

$$\mu = [\alpha^4,\ 0,\ 0,\ 0,\ 0,\ 0,\ \alpha^2]$$

de donde:

$$C = r - \mu = [\alpha,\ \alpha,\ \alpha,\ \alpha,\ \alpha,\ \alpha,\ \alpha]$$

Para analizar otro método de decodificación considérese lo definido en esta sección (4.4.2) hasta la obtención del polinomio localizador de errores $\sigma(x)$, por medio de las fracciones continuadas. [29,33,35,41].

Definanse los coeficientes ρ_k de la ecuación (4.14) como:

$$\rho(x) = \tau(x)\,\sigma(x) = \sum_{k=0}^{e+b} (-1)^k \rho_k\, x^{e+b-k} \qquad (4.22)$$

Si X_j es la posición de una errata:

$$\rho(X_j) = \sum_{k=0}^{e+b} (-1)^k \rho_k\, X_j^{e+b-k} = 0 \qquad j = 1, 2, \ldots, e+b \qquad (4.23)$$

y multiplicando (4.9) por (4.23) obtenemos:

$$\sum_{j=1}^{e+b} X_j Y_j^k \sum_{l=0}^{e+b} (-1)^l \rho_l\, X_j^{e+b-l} = 0$$

$$= \sum_{l=0}^{e+b} (-1)^l \rho_l \sum_{j=1}^{e+b} Y_j X_j^{2+b-l+k} = 0$$

$$= \sum_{l=0}^{e+b} (-1)^l \rho_l\, S_{e+b-l+k} = 0$$

$$= S_{e+b+k} + \sum_{l=1}^{e+b} (-1)^l \rho_l\, S_{e+b-l+k} = 0$$

de donde: $$S_i = -\sum_{l=1}^{e+b} (-1)^l \rho_l\, S_{i-l} \qquad i = d, \ldots, n \qquad (4.24)$$

La ecuación (4.24) nos permite calcular los componentes restantes transformada del vector de erratas. De manera que al obtener la transformada inversa del vector $S_0, S_1, \ldots, S_{n-1}$ hallamos μ, es decir

$$\mu_i = \sum_{k=0}^{n-1} S_k\, \alpha^{-ik} \qquad i = 0, 1, \ldots, n-1 \qquad (4.25)$$

Resumiendo, éste algoritmo consta de las etapas i), ii), iii) y iv) idénticas que el método anterior, teniéndose a continuación las etapas:

v) Cálculo de las componentes de la transformada del vector de erratas mediante (4.24), donde los ρ_l están definidas por (4.22).

vi) Cálculo de la transformada inversa del vector de síndromes, empleando la ecuación (4.25) para obtener μ.

vii) restar μ de r y obtener la palabra de código C.

EJEMPLO. Considérese el ejemplo anterior, del cual se tenían:

$$S_1 = \alpha^2 \qquad S_3 = \alpha^3$$

$$S_2 = \alpha^5 \qquad S_4 = 1$$

$$\sigma(x) = x + 1 \quad y \quad \tau(x) = x + \alpha^6$$

v) de donde: $\rho(x) = \sigma(x)\,\tau(x) = x^2 + \alpha^2 x + \alpha^6$

con $\rho_0 = 1$

$\rho_1 = \alpha^2$

$\rho_2 = \alpha^6$

para e+b=2 de (4.24) se tiene:

$$S_i = \sum_{l=1}^{2} \rho_l\, S_{i-l} \qquad i = 5, \ldots, 7$$

$$S_5 = \alpha^2 + \alpha^6\alpha^3 = 0$$

$$S_6 = 0 + \alpha^6(1) = \alpha^6$$

$$S_7 = S_0 = \alpha^2\alpha^6 + 0 = \alpha$$

vi) para el ejemplo planteado (4.25) resulta:

$$\mu_i = \sum_{k=0}^{6} S_k \alpha^{-ik} \qquad i = 0, 1, \ldots, 6$$

de donde:

$$\mu_0 = \alpha + \alpha^2 + \alpha^5 + \alpha^3 + 1 + \alpha^6 = \alpha^4$$

$$\mu_1 = \alpha + \alpha^2\alpha^{-1} + \alpha^5\alpha^{-2} + \alpha^3\alpha^{-3} + \alpha^4 + \alpha^6\alpha^{-6} = 0$$

$$\mu_2 = \alpha + \alpha^2\alpha^{-2} + \alpha^5\alpha^{-4} + \alpha^3\alpha^{-6} + \alpha^{-1} + \alpha^6\alpha^{-12} = 0$$

$$\mu_3 = \alpha + \alpha^2\alpha^{-3} + \alpha^5\alpha^{-6} + \alpha^3\alpha^{-9} + \alpha^{-12} + \alpha^6\alpha^{-18} = 0$$

$$\mu_4 = \alpha + \alpha^2\alpha^{-4} + \alpha^5\alpha^{-8} + \alpha^3\alpha^{-12} + \alpha^{-16} + \alpha^6\alpha^{-24} = 0$$

$$\mu_5 = \alpha + \alpha^2\alpha^{-5} + \alpha^5\alpha^{-10} + \alpha^3\alpha^{-15} + \alpha^{-20} + \alpha^6\alpha^{-30} = 0$$

$$\mu_6 = \alpha + \alpha^2\alpha^{-6} + \alpha^5\alpha^{-12} + \alpha^3\alpha^{-18} + \alpha^{-24} + \alpha^6\alpha^{-36} = \alpha^2$$

$$\mu = [\alpha^4, 0, 0, 0, 0, 0, \alpha^2]$$

vii) finalmente: $C = r - \mu = \alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha$

4.5 CODIGOS DE REDD-SOLOMON ACORTADOS Y EXTENDIDOS. 24,31 .

4.5.1. INTRODUCCION. Como se ha visto en capítulos anteriores, en ocasiones el tamaño del código, i.e. n=q-1 no corresponden a las necesidades planteadas en un caso particular. Es necesario entonces ajustar el código, ya sea a nuestro canal de información o a nuestra fuente de información, mediante la extensión o el acortamiento de dicho código.

4.5.2 CODIGOS RS ACORTADOS. El acortamiento se realiza mediante la disminución de cierto número de símbolos de información, manteniendo el mismo número de símbolos de redundancia. La nueva distancia mínima del código acortado es mayor o igual que la del código original.

Si se tiene originalmente un código RS(n,k,d), el acortado será $RS(n-i_{acor}, k-i_{acor}, \geq d)$

Puesto que los códigos RS son un caso particular de los BCH, los conceptos tratados en las secciones 3.3 y 3.4 siguen vigentes también aquí.

4.5.3. CODIGOS RS EXTENDIDOS. El añadir un símbolo de chequeo global de paridad a un código no siempre incrementa su distancia mínima, sin embargo en los códigos RS sí ocurre ésto:

Si tenemos originalmente el código RS(n,k,d) el añadir a la palabra de código $C = C_0, C_1, \ldots, C_{n-1}$ un nuevo símbolo

$$C_n = - \sum_{i=0}^{n=1} C_i$$

producirá un código RS(n+1, k, d+1), de manera que se incrementa su capacidad de corrección.

CAPITULO 5

COMPORTAMIENTO DE LOS CODIGOS CICLICOS

CAPITULO 5

COMPORTAMIENTO DE LOS CODIGOS CICLICOS. [2,24,31].

5.1 INTRODUCCION.

En las primeras etapas del diseño de un sistema digital de comunicaciones es importante conocer las capacidades así como las limitaciones de los códigos de bloque. Esta información aunada con el conocimiento de la complejidad de los esquemas de codificación, el retraso que introduce el código, el incremento de ancho de banda, la ganancia del código y sus medidas probabilísticas de comportamiento, entre otros parámetros, nos permiten la elección del código más apropiado a nuestro sistema.

5.2 MEDIDAS PROBABILISTICAS DE COMPORTAMIENTO. [15].

5.2.1 INTRODUCCION

En la selección de una técnica de codificación para el control de errores en un sistema digital de comunicaciones, se comparan varios diseños alternativos de códigos típicamente en base a diferentes medidas probabilísticas de comportamiento.

Como se mencionó en el capítulo 1 (párrafo 1.1.2.1) existen dos métodos básicos para el control de errores: FEC (Forward Error Correction) y ARQ (Automatic Repeat Request). En un sistema de comunicaciones con FEC se emplea un código correcto de errores para detectar, localizar y corregir errores; en éste sistema un parámetro típico de comparación entre diferentes códigos correctores de errores es la probabilidad de decodificación correcta (P_{CD}). Sin embargo, en muchas aplicaciones P_{CD} no nos suministra suficiente información acerca del comportamiento del código, teniéndose entonces que hacer comparaciones adicionales en base a la probabilidad de decodificación incorrecta (P_{ICD}). Otros parámetros de interés, que pueden ser útiles después de la decodificación son la tasa de error por símbolo después de la decodificación (P'_{CE}) y la distribución de los errores después de la decodificación ($P(\lambda)$).

Cuando se tiene un sistema de comunicaciones con ARQ se emplea un código

detector de error, teniéndose entonces como parámetro de comparación entre diferentes códigos la probabilidad de no detección (P_{UD}).

En éste capítulo se hablará de manera general de los códigos no binarios con símbolos pertenecientes a un $CG(q=2^m)$. Sin embargo los resultados pueden aplicarse a códigos binarios ($q=2$), en éste caso la tasa de símbolos en error P_{CE} es igual a la tasa de bits en error P_B. Si se supone independencia entre símbolos en un canal no binario en el que cada símbolo está representado por m bits, se tiene la siguiente relación:

$$P_B = \frac{2^{m-1}}{2^m - 1} P_{CE}$$

5.2.2 PROBABILIDAD DE DECODIFICACION CORRECTA

Un código lineal de largo n sobre un campo de q elementos y una distancia mínima d, puede corregir hasta t errores, donde $2t + 1 \leq d$. En un canal sin memoria, los errores en los símbolos son independientes e igualmente probables en cada una de las n posiciones de la palabra de código, por lo tanto los errores à la entrada del decodificador está distribuidos binomialmente y la probabilidad de una decodificación correcta es:

$$P_{CD} = \sum_{k=0}^{t} \binom{n}{k} P_{CE}^{k} (1 - P_{CE})^{n-k} \qquad (5.1)$$

donde P_{CE} es la tasa de símbolos en error del canal.

Cuando el número de errores a la entrada del decodificador es mayor que la capacidad de corrección del código pueden ocurrir dos eventos, el decodificador detecta que ocurrieron más de t errores y no decodifica (decodificación imposible) ó decodifica erroneámente con probabilidad P_{ICD}.

5.2.3 PROBABILIDAD DE DECODIFICACION INCORRECTA.

Para la evaluación de la probabilidad de decodificación incorrecta de los códigos lineales de bloque en un canal simétrico, se puede suponer que la pala

bra con todos sus símbolos igual a cero (palabra nula) se transmite. Una decodificación incorrecta ocurre si un patrón de error en la palabra recibida es decodificado en una palabra diferente a la palabra nula.

Sea $P_{ICD}(h)$ la probabilidad de decodificación incorrecta a una palabra de peso h. La probabilidad total de una decodificación incorrecta está dada por:

$$P_{ICD} = \sum_{h=d}^{n} P_{ICD}(h) \tag{5.2}$$

y la tasa de símbolos en error después de la decodificación P'_{CE} se define como el número esperado de errores después de la decodificación:

$$P'_{CE} = \frac{1}{n} \sum_{h=d}^{n} h\, P_{ICD}(h) \tag{5.3}$$

Para determinar $P_{ICD}(h)$, sea $n(h,k,s)$ el número de patrones de error de peso k que están a una distancia s de una palabra de código de peso h, desde luego que $n(h,k,s) \neq 0$ únicamente para $h - s \leq k \leq h + s$.

Los patrones para los cuales $s \leq t$ serán decodificados como la palabra de peso h y puesto que los errores de entrada están distribuidos binomialmente, la probabilidad de ocurrencia de alguno de los $n(h,k;s)$ patrones de error es:

$$P(k) = \frac{P_{CE}^{k}\,(1-P_{CE})^{N-k}}{(q-1)^{k}} \tag{5.4}$$

Puesto que los errores son independientes, $n(h,k;s)$, es la misma para todas las palabras de código de peso h, podemos entonces escribir:

$$P_{ICD}(h) = W(h) \sum_{S=0}^{t} \sum_{k=h-s}^{h+s} n(h,k;s)\, P(k), \quad h \geq d \tag{5.5}$$

donde $W(h)$ es el número de palabras de código de peso h.

Para evaluar $n(h,k;s)$, sea C_i el símbolo en la iésima posición de una palabra de código de peso h, y sea E_i el símbolo en la iésima posición del

patrón de error de peso K a la entrada del decodificador.

Se empleará μ para representar el número de elementos de un conjunto condicionado a los valores que toman C_i y E_i, entonces:

$$k = \mu\{i : E_i \neq 0\}$$

$$h = \mu\{i : C_i \neq 0\}$$

Sea:

$$m = \mu\{i : C_i = E_i \neq 0\}$$

$$j = \mu\{i : C_i \neq E_i,\ E_i \neq 0,\ C_i \neq 0\}$$

$$v = \mu\{i : C_i \neq 0,\ E_i = 0\}$$

$$r = \mu\{i : C_i = 0,\ E_i \neq 0\}$$

puesto que los conjuntos anteriores son exclusivos entre si, tenemos que:

$$m + j + r = k \tag{5.6}$$

y

$$m + j + v = h \tag{5.7}$$

La distancia s es el número de posiciones en que difieren la palabra de código y el patrón de error. Por lo tanto:

$$s = j + v + r \tag{5.8}$$

$$n(h,k,m,j,r) = \binom{h}{m}\binom{h-m}{j}(q-2)^j\binom{n-h}{r}(q-1)^r, \tag{5.9}$$

De las ecuaciones (5.6), (5.7) y (5.8) podemos poner m y j en términos de r:

$$m = h - s + r$$

$$j = k - h + s - 2r$$

y la ecuación (5.9) se puede escribir como:

$$n(h,k;s) = \sum_{r=r_1}^{r_2}\binom{h}{h-s+r}\binom{s-r}{k-h+s-2r}\binom{n-h}{r}(q-2)^{k-h+s-2r}(q-1)^r \tag{5.10}$$

donde $r_1 = \max\{0, k-h\}$ y $r_2 = [(k-h+s)/2]$, donde $[x]$ representa el entero más grande menor o igual que x.

Al sustituir (5.4) y (5.10) en (5.5) podemos finalmente evaluar $P_{ICD}(h)$ para los códigos en que su distribución de pesos $W(h)$ es conocida. Por ejemplo para los códigos Reed-Solomon:

$$W(h) = \binom{n}{h} (q-1) \sum_{i=0}^{h-d} (-1)^i \binom{h-1}{i} q^{h-d-i} \quad d \leq h \leq n \tag{5.11}$$

Por medio de $P_{ICD}(h)$ evaluamos la probabilidad total de decodificación incorrecta P_{ICD} en (5.2) y la tasa de símbolos en error después de la decodificación P_{CE} en (5.3).

Cuando tenemos un código de Reed-Solomon acortado en M símbolos de información, el número de palabras de código de peso h se obtiene sustituyendo la longitud del código acortado $n-M$ en lugar de n en la ecuación (5.11). Igualmente modificamos (5.4) y (5.10) para obtener $P_{ICD}(h)$ en un código acortado. Por lo tanto la probabilidad total de una decodificación incorrecta cuando no existen errores en los primeros M símbolos es:

$$P_{ICD} = \sum_{h=d}^{n-M} P_{ICD}(h) \tag{5.12}$$

y la tasa de símbolos en error después de la decodificación para un código de Reed-Solomon acortado es:

$$P'_{CE} = \frac{1}{n-M} \sum_{h=d}^{n-M} h\, P_{ICD}(h) \tag{5.13}$$

Los parámetros obtenidos en ésta sección se pueden extender para incluir la decodificación de errores y borrados para aquellos códigos en los que su distribución de pesos, modificada por el borrado de ℓ posiciones, se conoce.

La ocurrencia de ℓ borrados en un código de largo n puede interpretarse como un acortamiento de la longitud del código a $n-\ell$. Por lo tanto:

$$P_{ICD}(h) = \sum_{\ell=0}^{d-1} W_{n-\ell}(h) \sum_{S=0}^{\lfloor (d-\ell-1)/2 \rfloor} \sum_{k=h-s}^{h+s} n_{n-\ell}(h,k,s)\, P(k,\ell) \qquad (5.14)$$

donde $W_{n-\ell}(h)$ es la distribución de pesos del código "acortado" y está dada por (5.11) para los códigos de Reed-Solomon sustituyendo n por n-ℓ y d por d-ℓ. Similarmente $n_{n-\ell}(h,k,s)$, se obtiene de (5.10) sustituyendo n por n-ℓ.

$P(k,\ell)$ está dada por:

$$P(k,\ell) = \frac{\binom{n-k}{\ell} P_{CE}^{k} P_{ER}^{\ell} (1-P_{CE}-P_{ER})^{n-k-\ell}}{(q-1)^{k}} \qquad (5.15)$$

donde P_{ER} es la tasa de caracteres borrados.

5.2.4 PROBABILIDAD DE NO DETECCION. [42].

En un código lineal de bloque empleado para detección de errores, no se detectará un error cuando el patrón de error sea exactamente una palabra de código diferente de la nula, pues la palabra recibida será otra palabra de código y el decodificador la tomará por correcta.

Para un código lineal de bloque de longitud N con símbolos de un CG(q) y distancia mínima d, la probabilidad de no detección de error P_{UD} está dada por:

$$P_{UD} = \sum_{h=d}^{N} W(h) \frac{P_{CE}^{h}(1 - P_{CE})^{n-h}}{(q-1)^{h}} \qquad (5.16)$$

donde P_{CE} es la tasa de símbolos en error del canal y W(h) es el número de palabras de código de peso h que hay en el código.

Para la mayoría de los códigos la distribución de pesos W(h) se desconoce (para los códigos de máxima distancia de separación W(h) está dada por (5.11)), por lo que se han obtenido fronteras del comportamiento de P_{UD} que no requieren del conocimiento de W(h). [19].

5.2.5 GRAFICAS COMPARATIVAS

En este párrafo se muestran algunas curvas de códigos de Reed-Solomon para observar su comportamiento al variar su longitud, su distancia y la tasa de símbolos borrados.

En la figura 5.1 se ha graficado la tasa de símbolos en error después de la decodificación P'_{CE} contra la tasa de símbolos en error P_{CE} a la entrada del decodificador, para un código de longitud n=31, en donde se han tomado diferentes valores de la distancia. Se puede apreciar como al aumentar la distancia mínima d, P'_{CE} disminuye para un mismo valor de P_{CE}. Al incrementar d y mantener fija la longitud N estamos aumentado los símbolos de redundancia y por lo tanto la protección de la información. Nótese que la tasa del código k/n disminuye, pues n se mantiene constante, por lo tanto para transmitir la misma cantidad de información en el mismo intervalo de tiempo habrá que incrementar la velocidad de símbolos por unidad de tiempo transmitidos.

Lo mismo ocurre con un código de Reed Solomon empleado para detección de errores, su probabilidad de no detección P_{UD} decrece al incrementar su distancia cuando se mantiene fija su longitud. Fig. 5.2.

Como se puede ver en la figura 5.3 al acortar un código corrector de errores, disminuyendo el número de símbolos de información, y manteniendo fija su distancia mínima también disminuye P'_{CE} para un valor dado de P_{CE}.

Si comparamos algunos códigos de Reed Solomon en los cuales su tasa es aproximadamente constante para diferentes longitudes, fig. 5.4, vemos como al aumentar la longitud del código, por tanto al incrementar la complejidad del decodificador y el retardo que introduce el código, se puede disminuir apreciablemente la tasa de símbolos en error después de la decodificación. En la figura 5.4 se emplea como referencia la tasa de bits en error P_B suponiendo independencia entre símbolos:

$$P_B = \frac{2^{m-1}}{2^m-1} P_{CE} \quad , \quad \text{con} \quad n = 2^m-1.$$

Por último en la figura 5.5 vemos como al decodificar errores y borrados, con un código de Reed-Solomon, P'_{CE} disminuye al aumentar la tasa de borrados

P_{ER} para un mismo código. P_{ER} puede controlarse aumentando o disminuyendo el umbral en donde se declaran los símbolos recibidos como borrados.

5.2.6 OTROS PARAMETROS DE COMPARACION. [15,43,44].

La ganancia de un código se define como: la mejora equivalente de la relación señal a ruido E/N_0 obtenida para un esquema dado de corrección de errores. Esta ganancia es muy práctica por ser compatible y comparable con parámetros tales como la ganancia de una antena o el factor de ruido de los amplificadores y puede ser muy útil cuando se consideran compromisos entre potencia transmitida, tamaño de la antena, ganancia del amplificador, etc. [43].

La ganancia g de un código de bloque de largo n con k símbolos de información es el cociente:

$$g = \frac{k\,\rho_0^2}{n\,\rho_1^2} \tag{5.17}$$

donde ρ_0^2 es la relación señal a ruido sin codificación y ρ_1^2 es la relación señal a ruido después de la codificación.

Otra medida de comportamiento útil para comparar sistemas de comunicación que emplean diferentes códigos es la tasa de símbolos de información en error P_{IS}. Este parámetro es función de la distribución de pesos del código, pero cuando se usa una decodificación "hard-decision", esto es cuando no se toman en cuenta ciertas medidas estadísticas de canal de comunicaciones, se puede aproximar mediante la siguiente expresión:

$$P_{IS} \simeq \frac{d}{n} \sum_{i=t+1}^{d} \binom{n}{i} P_{CE}^{i} (1 - P_{CE})^{n-i} + \frac{1}{n} \sum_{i=d+1}^{n} i \binom{n}{i} P_{CE}^{i} (1 - P_{CE})^{n-i} \tag{5.18}$$

Otro parámetro de interés es la distribución de los errores después de la decodificación $P(\lambda)$ definida por:

$$P(\lambda) = \sum_{h=\lambda+1}^{n} P_{ICD}(h) = P_{ICD} - \sum_{h=d}^{\lambda} P_{ICD}(h) \tag{5.19}$$

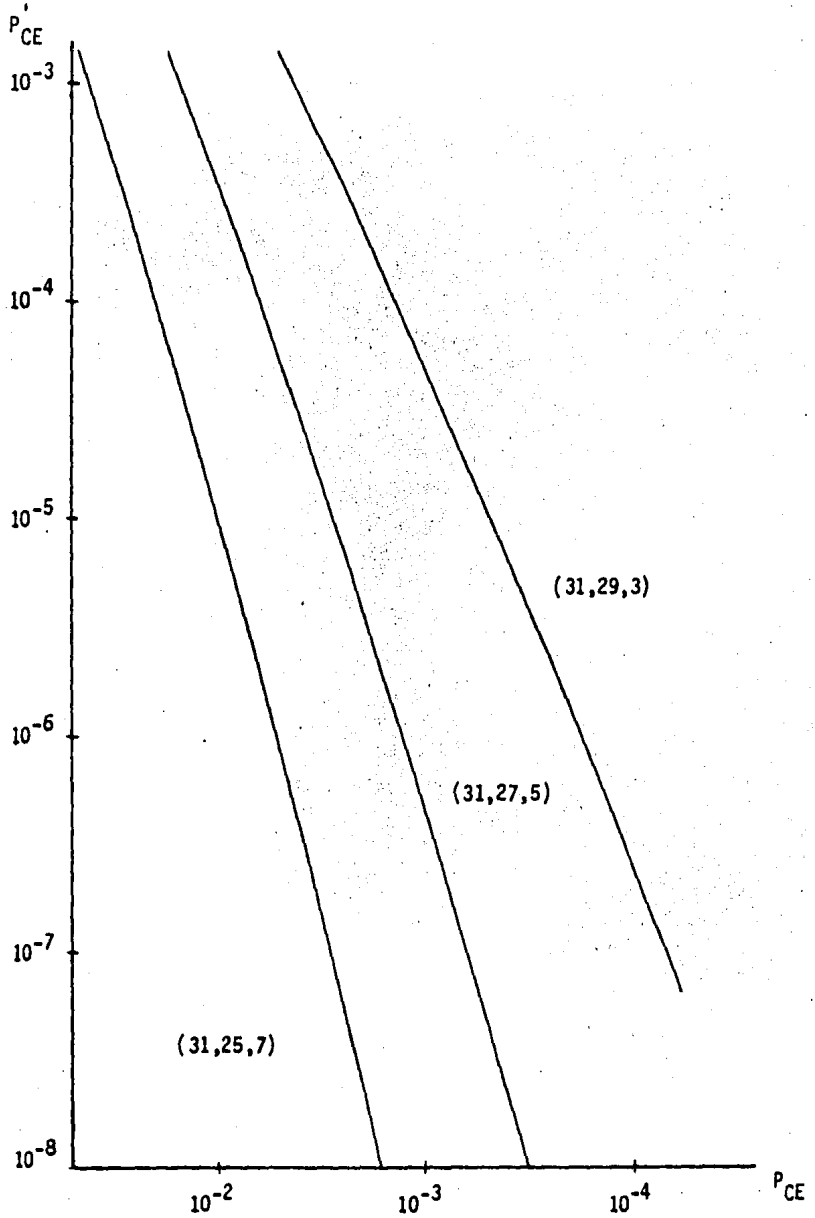


Fig. 5.1 Tasa de símbolos en error después de la decodificación P'_{CE} vs tasa de símbolos en error a la entrada del decodificador P_{CE} para códigos de R.S. de longitud n=31 y distancias 3, 5 y 7.

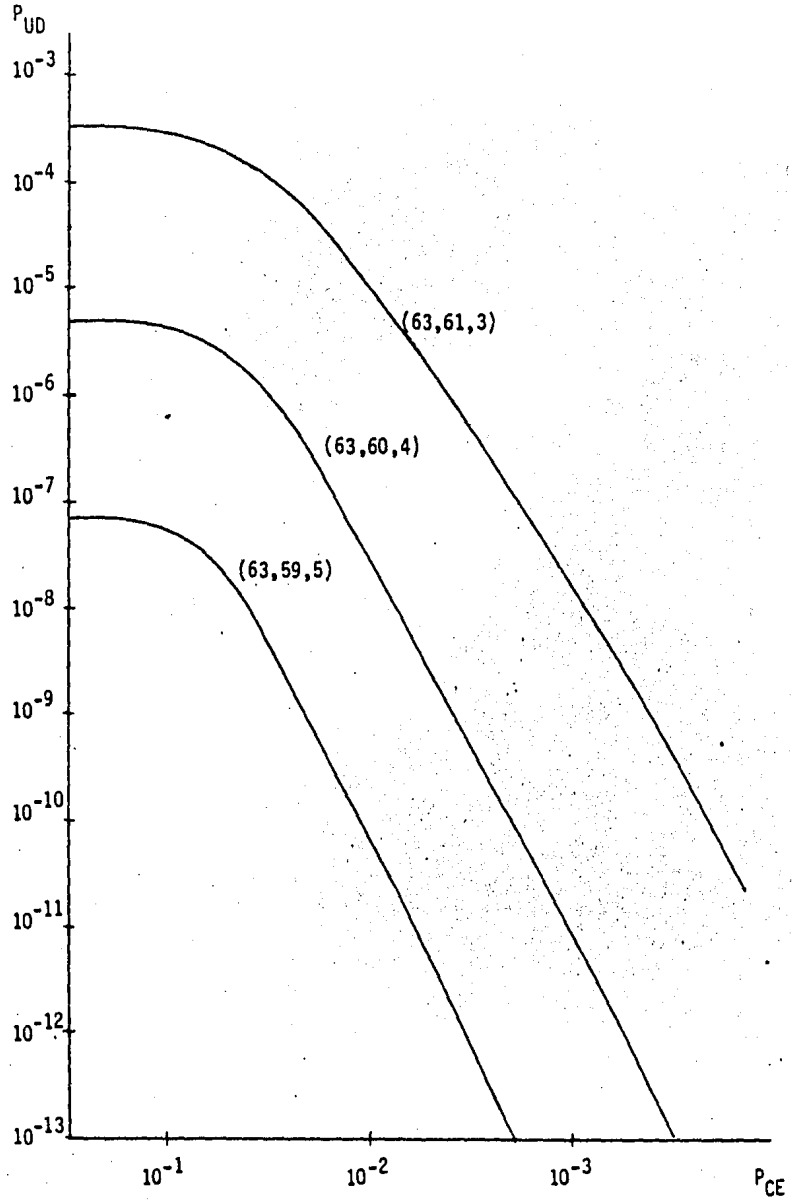


Fig. 5.2 Probabilidad de no detección P_{UD} vs P_{CE} para códigos de R.S. con n=63 y distancias 3, 4 y 5.

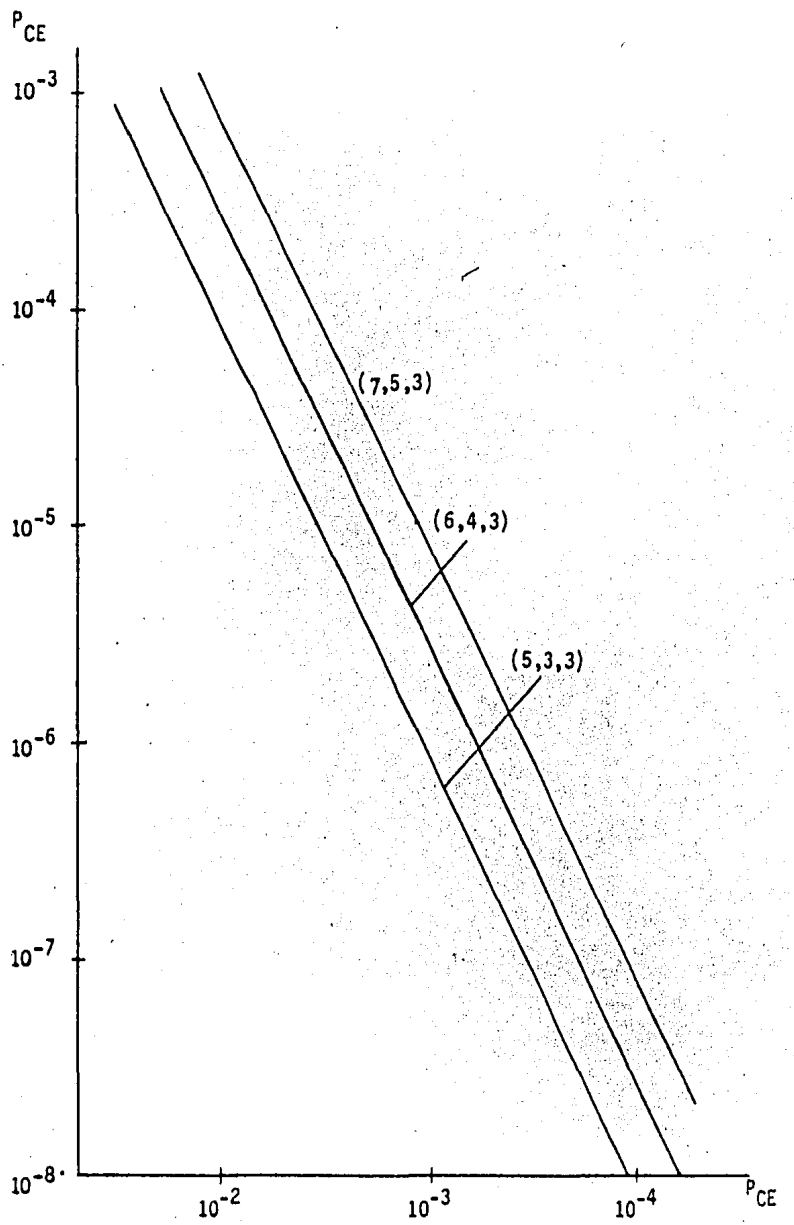


Fig. 5.3 P'_{CE} vs P_{CE} para el código de R.S. (7,5,3) y cuando éste es acortado a (6,4,3) y (5,3,3).

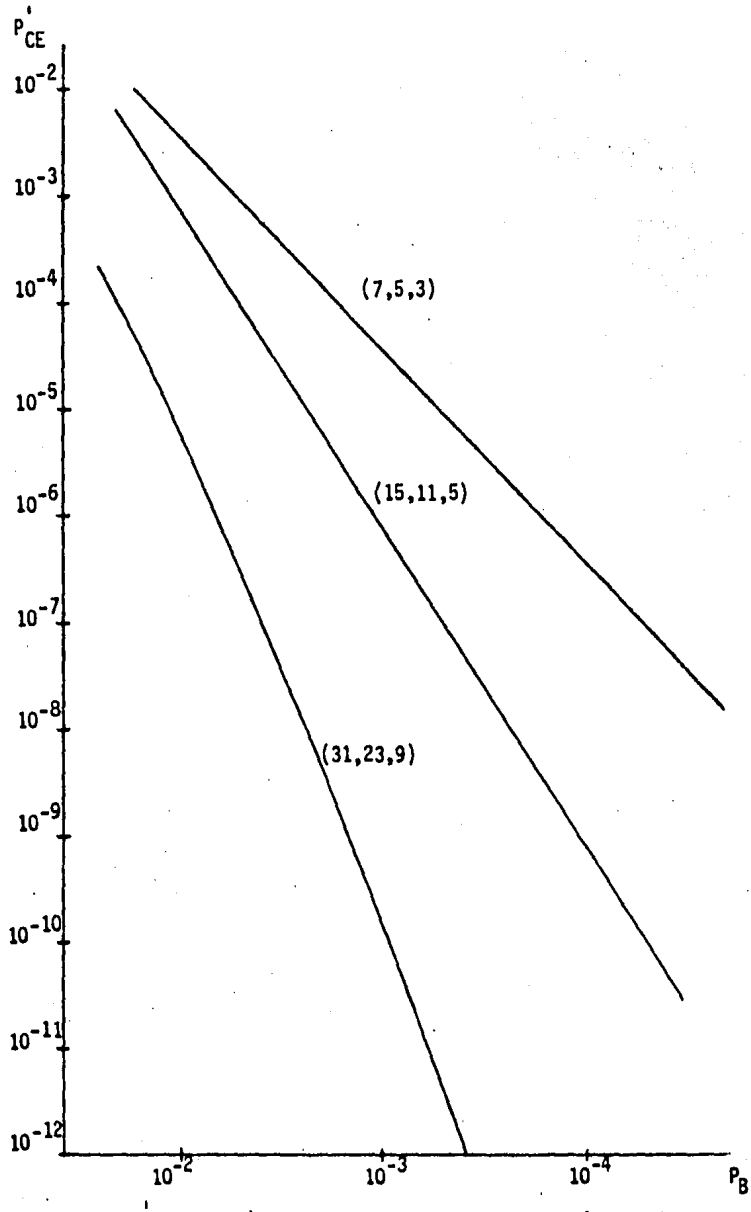


Fig. 5.4 P'_{CE} vs tasa de bits en error P_B para códigos de R.S. de longitud 7,15 y 31 con una tasa k/n aproximadamente constante.

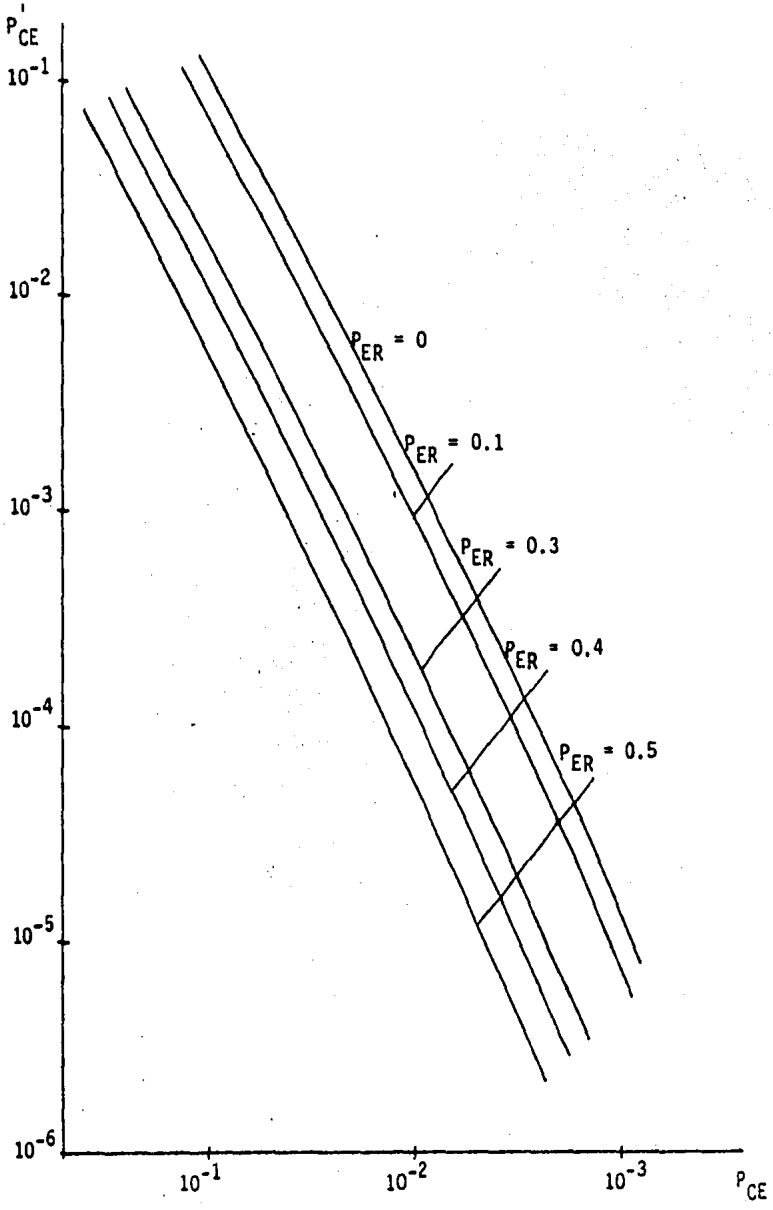


Fig. 5.5 P'_{CE} vs P_{CE} para el código RS(15,9,7), considerando diferentes tasas de borrados P_{ER}.

es decir, $P(\lambda)$ es la probabilidad de que ocurran más de λ errores después de la decodificación. P_{ICD} y $P_{ICD}(h)$ están dadas por (5.2) y (5.5) respectivamente.

5.3 COMPLEJIDAD DE LOS ESQUEMAS DE CODIFICACION Y DECODIFICACION DE CODIGOS CICLICOS.

5.3.1 CODIGOS CICLICOS BINARIOS DE HAMMING Y FIRE.

Una medida de la complejidad de la implementación de un esquema de codificación y decodificación, cuando se emplea un código de Hamming o de Fire, es el número de componentes de la lógica secuencial y combinacional. Aquí se hará referencia a tres parámetros: El número de registros de corrimiento (e.g. Flip-Flop tipo D), la cantidad de compuertas OR-exclusivas (XOR) y el número de compuertas lógicas empleadas en la implementación de la lógica combinacional.

Para un código binario cíclico (n,k) (de Hamming o de Fire), el número de registros de corrimiento (NR) necesarios en la implementación está dado por la siguiente expresión:

$$NR = 2(n-k) + n \qquad (5.20)$$

donde:

n = Longitud del código
k = Dimensión del código

En efecto, se requieren (n-k) registros de corrimiento en la codificación (considerando que, en general, los códigos de interés práctico tienen en una tasa $k/n>0.5$, y por lo tanto el método empleado de codificación es el de división por $g(x)$). En la decodificación, utilizando el circuito decodificador Meggitt, se requieren (n-k) registros para el generador de síndrome y n registros para el almacenador de la palabra recibida.

Nótese, sin embargo, que el almacenador de la palabra recibida puede constar únicamente de k registros de corrimiento. Para lograr esto, el circuito decodificador deberá incluir un interruptor digital que inicialmente permita la lle

gada de los k bits de información, tanto al almacenador como al generador de síndrome. Después de k ciclos, el interruptor abre la trayectoria habia el almacenador, y los (n-k) bits de redundancia son canalizados únicamente hacia el generador de síndrome. Al completarse la llegada de la palabra recibida, en n ciclos, la corrección de ésta se realiza en k ciclos, en lugar de los n ciclos que demoraría el circuito decodificador sin el interruptor. Por lo tanto, existe un compromiso entre el costo de la implementación y su rapidez. Dependerá del diseñador la elección del esquema adecuado de codificación, de acuerdo a las características y necesidades particulares del sistema en el que se implante.

Por otra parte, la cantidad de compuertas OR-exclusivas (NC) necesarias para la implementación es:

$$NC = 2NG + NM - 1 \qquad (5.21)$$

donde:

NG = Número de coeficientes del polinomio generador del código, g(x), distintos de cero.
NM = Número de coeficientes no nulos del polinomio x^{n-k+i} módulo g(x).
i = Acortamiento del código (i.e. número de bits no utilizados).

En la codificación son necesarias (NG-1) compuertas XOR correspondientes a las conexiones de realimentación, tanto de la división por g(x) como de la premultiplicación de los bits de información por x^{n-k}. El circuito decodificador Meggitt requiere de (NG-1) compuertas XOR en el generador de síndrome y NM compuertas para implementar la premultiplicación de la palabra recibida. Además, es necesaria una compuerta adicional para la corrección de la palabra recibida.

Finalmente, se requieren NL compuertas lógicas varias para la implementación de la lógica combinacional del circuito decodificador Meggitt. Para los códigos cíclicos binarios de Hamming, la lógica debe detectar la ocurrencia de la condición (000..01) sobre los (n-k) bits en el generador de síndrome. Entonces, son necesarias (n-k-3) compuertas OR, una compuerta NOR y una compuerta AND para la implementación. En el caso de los códigos de

Fire, la lógica debe detectar la presencia de ceros en los (n-k-b) bits menos significativos del generador de síndrome, para lo cual son necesarias (n-k-b-1) compuertas OR. Para efectuar la modificación del síndrome y la corrección de la palabra recibida, se requiere además de dos compuertas AND y de un inversor (NOT). Por lo tanto, el número de compuertas requeridas, para cada familia de código, es:

$$NL_{Hamm} = n-k-1$$
$$NL_{Fire} = n-k-b+2 \qquad (5.22)$$

5.3.2 CODIGOS CICLICOS BINARIOS DE BCH.

Aunque los códigos cíclicos de BCH, analizados en este trabajo, son binarios, es necesario considerarlos aisladamente de los códigos cíclicos binarios del párrafo anterior. Esto se debe a que como se describió en el capítulo 3, en la decodificación de esta familia de códigos se realizan operaciones sobre un campo finito, CG(q). En adelante, para hacer referencia a las operaciones sobre CG(q), se emplearán los siguientes parámetros:

NS = Número de sumas sobre CG(q).
NM = Número de multiplicaciones sobre CG(q). (5.23)
NE = Número de exponenciaciones sobre CG(q).

5.3.2.1 CODIFICACION

Al igual que para los códigos del párrafo anterior, la complejidad de la implementación de la codificación se mide por el número de registros de corrimiento (NR) y de compuertas XOR (NC). Para los códigos cíclicos binarios de BCH, estos parámetros son:

$$NR = n-k$$
$$NC = NG-1 \qquad (5.24)$$

donde:

n = Longitud del código.
k = Dimensión del código.

NG = Número de coeficientes del polinomio generador, g(x), distintos de cero.

5.3.2.2 DECODIFICACION.

i) Cálculo del síndrome.

Como se expuso anteriormente (3.11), los componentes del síndrome de la palabra recibida son calculados mediante:

$$S_i = r(\alpha^i) \quad , i = 1, 3, \ldots, 2t-1$$

$$S_i = S_{i/2}^2 \quad , i = 2, 4, \ldots, 2t$$

Estos cálculos pueden efectuarse evaluando el residuo de la división de la palabra recibida entre los polinomios mínimos, factores del polinomio generador del código (Fig. 3.1). En este caso, los valores exactos de los parámetros NR y NC dependen del código particular que se quiera implementar. Sin embargo, es posible acotar dichos valores para obtener una medida aproximada de la complejidad de esta etapa:

$$\begin{aligned} NR &\leq mt \\ NC &\leq mt + m^2t \end{aligned} \qquad (5.25)$$

ya que el grado máximo de los polinomios mínimos es m, y se tienen t de éstos como factores del polinomio generador del código (3.4). La cota del número de compuertas XOR se obtiene considerando que, en el peor de los casos, todos los coeficientes de los polinomios mínimos son distintos de cero y de grado máximo. Se considera además que cada bit, de los m que constituyen cada síndrome, en promedio se obtiene mediante la combinación de m/2 bits del residuo de la división entre el polinomio mínimo correspondiente. El valor exacto de los parámetros dependerá del código particular a implementar.

ii) Algoritmo de Berlekamp-Massey.

En esta etapa se determina el polinomio localizador de errores, cuyos coeficientes se encuentran al resolver el sistema de ecuaciones 3.16 :

$$
\begin{aligned}
&S_{e+1} + S_e\sigma_1 + \ldots + S_1\sigma_e = 0 \\
&S_{e+2} + S_{e+1}\sigma_1 + \ldots + S_2\sigma_e = 0 \\
&\ldots\ldots\ldots\ldots\ldots\ldots \\
&S_{2t} + S_{2t-1}\sigma_1 + \ldots + S_{2t-e}\sigma_e = 0
\end{aligned}
\qquad (5.26)
$$

Para resolver este sistema de ecuaciones, es necesaria una unidad de cálculo sobre CG(q), y por lo tanto la complejidad se mide por el número de operaciones sobre CG(q), además del hecho de contar con una unidad especializada de este tipo. Cuando se emplea el algoritmo de Berlekamp-Massey para la evaluación de $\sigma(x)$, la solución se encuentra después de:

$$
\begin{aligned}
NS &= 2e^2 + e \\
NM &= 3e^2 + e
\end{aligned}
\qquad (5.27)
$$

operaciones sobre CG(q).

iii) Búsqueda de Chien.

La evaluación de $\sigma(x)$ para $x = \alpha^{-i}$, $i=0,1,\ldots,n-1$ se logra mediante los circuitos de la Fig. 3.1, inicialmente cargados con los coeficientes de $\sigma(x)$ y que multiplican a éstos por $x, x^2, \ldots, x^t$. Esto es equivalente a evaluar $\sigma(x)$ mediante:

$$
\sigma(\alpha^{-i}) = \sum_{j=0}^{e} \sigma_j \alpha^{-ij}, \qquad \sigma_0=1, \qquad i=0,\ldots,n-1
$$

la suma se implementa mediante (mt) compuertas XOR y (m-1) compuertas OR, aunadas a una compuerta NOT para la corrección de la palabra recibida. La cantidad de registros de corrimiento empleados en esta etapa es mt y se emplean, además de las anteriores, un máximo de mt compuertas XOR aproximadamente, el número exacto de compuertas depende del código particular que se desea.

La complejidad de esta etapa de la decodificación está dada por:

$$
\begin{aligned}
NR &= mt \\
NC &\leq 2mt \\
NL &= m
\end{aligned}
\qquad (5.28)
$$

iv) Corrección de la palabra recibida.

Para los códigos cíclicos binarios de BCH, la corrección de los errores contenidos en la palabra recibida se logra complementando los bits en las posiciones correspondientes. En términos de compuertas XOR, esta etapa exige una compuerta para lograr la corrección. Además, esto requiere de un almacenador de n registros para la palabra recibida de corrimiento, con un cierto retraso (debido a la implementación del algoritmo de Berlekamp-Massey). Como se señaló en el capítulo 3, la búsqueda de Chien y la corrección de la palabra recibida se realizan de manera simultánea, al encontrar que $\sigma(\alpha^{-i}) = 0$, se corrige el i-ésimo bit que sale del almacenador. En esta etapa final de la decodificación, el valor de los parámetros es:

$$\begin{aligned} NR &= n \\ NC &= 1 \end{aligned} \qquad (5.29)$$

Debe señalarse también que el almacenador de la palabra recibida debe ser inhibido, por la unidad de cálculo sobre CG(q), una vez que los n bits de la palabra recibida han sido almacenada. Cuando se hayan determinado los coeficientes del polinomio localizador de errores, y se pasen a la siguiente etapa, la unidad permitirá al almacenador comenzar a desahogar la palabra recibida para su posible corrección.

5.3.3 CODIGOS DE REED-SOLOMON

Recuérdese, del capítulo 4, que los códigos de Reed-Solomon trabajan con elementos de un campo CG(q). Entonces, tanto la codificación como la decodificación se efectúan con operaciones sobre CG(q), y éstas son la medida más adecuada de la complejidad de la implementación del esquema. Lo anterior implica la necesidad de una unidad de cálculo sobre CG(q), memoria y lógica secuencial y combinacional dedicadas (o de propósito especial).

5.3.3.1 CODIFICACION.

Método 1. División por G(x).

Sean los polinomios generador y de información respectivamente:

$$G(x) = G_0 + G_1 x + \ldots + G_{d-1} x^{d-1}$$

$$I(x) = I_0 + I_1 x + \ldots + I_{k-1} x^{k-1} \qquad , \; d=n-k+1 \qquad (5.30)$$

El polinomio, asociado a la palabra de código, está dado por:

$$C(x) = x^{n-k}\, I(x) + x^{n-k}\, I(x) \quad \text{módulo } G(x)$$

Para obtener el número de operaciones sobre $CG(q)$, es necesario analizar el algoritmo euclidiano de la división, usado en este método de codificación:

$$G_{d-1}x^{d-1} + G_{d-2}x^{d-2} + \ldots + G_0 \;\Big|\; \begin{array}{l} \dfrac{I_{k-1}}{G_{d-1}} x^{n-d} + \dfrac{R_{n-2}}{G_{d-1}} x^{n-2} + \ldots \dfrac{R_{n-k}}{G_{d-1}} x^{n-d-k+1} \\ \hline I_{k-1}x^{n-1} + I_{k-2}x^{n-2} + \ldots + I_0 x^{n-k} \\ \\ I_{k-1}x^{n-1} + \dfrac{I_{k-i}}{G_{d-1}} G_{d-2} x^{n-2} + \ldots \\ \hline \\ 0 \qquad R_{n-2}x^{n-2} + \ldots \end{array}$$

$$\vdots$$

De donde:

$$NS \leq k\,(n-k)$$
$$NM \leq k\,(n-k) \qquad (5.31)$$

Método 2. Recurrencia con $H(x)$.

Los coeficientes del polinomio $C(x)$ son obtenidos mediante la ecuación de recurrencia lineal (1.22):

$$C_j = - \sum_{i=0}^{k-1} h_i\, C_{k-i+j} \qquad , \; j \in [0, n-k-1]$$

los símbolos de paridad se obtienen después de:

$$\begin{aligned} NS &\leq k\,(n-k) \\ NM &\leq k\,(n-k) \end{aligned} \qquad (5.32)$$

operaciones sobre CG(q).

Como se observa, ambos métodos de codificación requieren el mismo número de operaciones sobre CG(q). Sin embargo, el método de división por G(x) requiere de más memoria y de más tiempo en el procesamiento de la información. Por esta razón, el método más usado para codificar información con códigos de Reed-Solomon, es el de recurrencia con H(x).

5.3.3.2 DECODIFICACION

i) Cálculo del síndrome.

El síndrome de la palabra recibida se calcula mediante la ecuación (3.11):.

$$S_i = R(\alpha_i) = \sum_{j=0}^{n-1} R_j\, \alpha^{ij}, \qquad i=1,2,\ldots,2t$$

Si la distancia mínima del código es d=2t+1., entonces son necesarias:

$$\begin{aligned} NS &\leq n\,(d-1) \\ NM &\leq (n-1)(d-1) \\ NE &\leq (n-1)(d-1) \end{aligned} \qquad (5.33)$$

operaciones sobre CG(q) para calcular los componentes del síndrome de la palabra recibida:

ii) Polinomio localizador de borrados, $\tau(x)$.

Recordando que $\tau(x)$ se define (3.27) como:

$$\tau(x) = \prod_{j=1}^{b} (x=X_j) = \tau_0 + \tau_1 x + \ldots + \tau_b x^b$$

su cálculo involucra:

$$NS = b-1 + \sum_{i=2}^{b} i = \frac{b^2+3b}{2} - 2$$
$$NM = 2 \sum_{i=2}^{b} i = b(b+1) - 2, \quad b > 1 \tag{5.34}$$

operaciones sobre CG(q).

iii) Síndromes de Forney, T(x).

Los síndromes de Forney se calculan mediante la expresión 3.28 :

$$T(x) = (1+S(x))\, \tau'(x) + 1 \text{ módulo } x^d$$

Ya que $1 + S(x)$ es de grado $(d-1)$ y $\tau'(x)$ es de grado b, el número de operaciones sobre CG(q) es:

$$NS = b(d-1)$$
$$NM = b(d-1) \tag{5.35}$$

iv) Algoritmo de Berlekamp-Massey:

En esta etapa, se obtienen los coeficientes del polinomio localizador de errores, al resolver el sistema de ecuaciones:

$$\sum_{j=0}^{e} \sigma_j T_{i+e+b-j} = 0, \quad i \in 1, 2t-e, \quad \sigma_0 = 1 \tag{5.36}$$

el cuál, expresado en forma explícita, es:

$$\begin{aligned} &T_{e+b+1} + T_{e+b}\sigma_1 + \cdots + T_{b+1}\sigma_e = 0 \\ &T_{e+b+2} + T_{2+b+1}\sigma_1 + \cdots + T_{b+2}\sigma_e = 0 \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ &T_{2t+b} + T_{2t+b-1}\sigma_1 + \cdots + T_{2t-e+b}\sigma_e = 0 \end{aligned} \tag{5.37}$$

Su solución, al igual que en el caso de los códigos cíclicos binarios de BCH, se encuentra después de:

$$\begin{aligned} NS &= 2e^2+e \\ NM &= 3e^2+e \end{aligned} \qquad (5.38)$$

operaciones sobre CG(q).

v) Búsqueda de Chien:

El cálculo de las posiciones de los errores, se realiza mediante la evaluación de $\sigma(x)$ para $x=\alpha^{-i}$, $i=0,1,\ldots,n-1$, puede expresarse como:

$$\sigma(\alpha^{-i}) = \sum_{j=0}^{e} \sigma_j \alpha^{-ij}, \ \sigma_0=1, \quad i=0,1,\ldots,n-1 \qquad (5.39)$$

el número de operaciones, cuando se emplea el método de la búsqueda de Chien, es:

$$\begin{aligned} NS &\leq (e+1)\, n \\ NM &\leq en \\ NE &\leq en \end{aligned} \qquad (5.40)$$

vi) Cálculo del polinomio localizador de erratas, $\Omega(x)$.

El polinomio $\Omega(x)$ se calcula por medio de la expresión 3.33 :

$$\Omega(x) = [1 + T(x)]\, \sigma(x) \quad \text{módulo } x^d$$

observando que el grado de $[1 + T(x)]$ es $(d-1)$ y el de $\sigma(x)$ es e, el número de operaciones sobre CG(q) es:

$$\begin{aligned} NS &= (d-e)\, e + \sum_{i=1}^{e-1} (e-i) \\ NM &= (d-e)\, e + \sum_{i=1}^{e-1} (e+i) \end{aligned} \qquad (5.41)$$

vii) Cálculo de los valores de las erratas:

Los valores de las erratas se obtienen a partir de los coeficientes de $\Omega(x)$ y de las posiciones de las erratas X_j, $j \in [1, e+b]$, por medio de la ecuación (3.34):

$$Y_j = \frac{X_j^{(e+b-1)}\ \Omega\ (X_j^{-1})}{\prod_{i \neq j} (X_j - X_i)}\ , \quad j \in [1, e+b]$$

$$= \frac{X_j^{(e+b-1)} \sum_{k=0}^{e+b-1} \Omega_k\ X_j^{-k}}{\prod_{i \neq j} (X_j - X_i)}\ , \quad j \in [1, e+b] \qquad (5.42)$$

estos cálculos requieren de:

$$\begin{aligned} NS &\leq d + \left\lceil \frac{e+b}{2} \right\rceil (e+b) \\ NM &\leq d + 1 + \left\lceil \frac{e+b}{2} \right\rceil (e+b) \\ NE &\leq (e+b)\ d \end{aligned} \qquad (5.43)$$

operaciones sobre CG(q).

viii) Corrección de la palabra recibida.

En esta etapa, a partir de las posiciones y los valores de las erratas (X_j, Y_j), $j \in [1, e+b]$, se realiza la corrección de la palabra recibida. Claramente, esta etapa se consuma realizando $(e+b)$ sumas, por lo tanto:

$$NS = e+b \qquad (5.44)$$

CAPITULO 6

IMPLEMENTACION DE UN CODIGO DE REED-SOLOMON PARAMETRABLE MEDIANTE UN SISTEMA MICROPROGRAMADO

CAPITULO 6

IMPLEMENTACION DE UN CODIGO DE REED SOLOMON PARAMETRABLE MEDIANTE UN SISTEMA MICROPROGRAMADO.

6.1 INTRODUCCION

El empleo de los códigos de Reed-Solomon ha estado limitado en gran parte por la complejidad de sus algoritmos de decodificación, a pesar de su alta capacidad para corregir paquetes múltiples de errores. Esta complejidad ha hecho que, para algunas aplicaciones, el retraso que introducen éstos códigos no sea aceptable y además que su costo sea elevado.

Sin embargo ahora con el rápido desarrollo de la tecnología y con la integración a muy grande escala (VLSI), se dispone de sistemas mucho más rápidos y relativamente más económicos, que permiten ampliar el campo de aplicación de éstos poderosos códigos.

En éste capítulo se describe el modo en que se implementaron los algoritmos de codificación y decodificación para los códigos de Reed Solomon en un sistema que tiene como soporte el microprocesador MC68000. El objetivo de ésta implementación es poder evaluar el comportamiento de ésta familia de códigos, midiendo los tiempos de codificación y de decodificación para diferentes combinaciones de errores y borrados.

6.2 DESCRIPCION DEL SISTEMA. [45,46].

El codificador y decodificador para códigos de Reed Solomon, se implementó en un sistema MEX68KECB/D2, el cual en una sola tarjeta de circuito impreso contiene un sistema completo de microprocesador, que incluye un microprocesador MC68000, 32 K bytes de memoria RAM dinámica arreglada como 16Kx16, 16 K bytes de ROM/EPROM arreglada como 8Kx16, dos puertos serie MC6850 compatibles con la norma RS-232C, un puerto paralelo y un temporizador programable PI/T MC68230. Figura 6.1.

El microprocesador tiene un conducto (bus) de datos de 16 bits; un conducto de direcciones de 23 bits que puede direccionar un rango de memoria de 8 megawords, mediante 14 modos de direccionamiento. Tiene también, un registro de

estado de 16 bits, siete registros de direcciones de 32 bits cada uno, ocho registros de datos de 32 bits cada uno, que pueden ser utilizados para efectuar operaciones de 8, 16 ó 32 bits, además cuenta con dos apuntadores de pila (stack) también de 32 bits. La frecuencia de trabajo del microprocesador es de 4 MHz.

Como la muestra la Figura 6.1, un puerto serie permite la interfaz con la terminal y el otro puerto serie con un sistema huésped (modem).

El PI/T MC68320 [47] es la interfaz entrada/salida paralelo. Puede tener un amplio rango de modos de operación mediante la programación de sus 23 registros internos, sin embargo, está configurado en la tarjeta para suministrar la interfaz con una impresora tipo Centronics y también la interfaz para grabadora de audio, la cual nos permite salvar programas o datos en cassette. El PI/T contiene además un temporizador programable de 24 bits, que fué empleado en éste trabajo como medidor de tiempos de ejecución de las diferentes subrutinas del programa implementado.

El sistema MEX68KECB/D2 tiene además un paquete residente "firmware" que suministra las funciones de monitoreo/rastreo de errores, ensamblador/desensamblador, entrada del programa y controles de entrada/salida. Cabe mencionar que esta tarjeta está diseñada para uso educacional y de entrenamiento, como una introducción a los sistemas de 16 bits basados en la familia MC68000.

En resumen, basados en este sistema se ha simulado el codificador-decodificador de códigos de Reed-Somomon, utilizando el lenguaje ensamblador. El puerto conectado a la terminal se utiliza como entrada/salida del sistema tanto para la información a codificar como para el ruido. En los párrafos siguientes se describe en detalle la simulación realizada.

6.3 DISEÑO DEL ESQUEMA.

Como parte inicial en la descripción del esquema, es conveniente mostrar las principales características del programa implementado en lenguaje ensamblador, pues aunque se siguen los mismos algoritmos que en la simulación en FORTRAN, la diferencia principal radica en la forma de implementar las operaciones básicas para éstos algoritmos y en la manera de manejar los polinomios

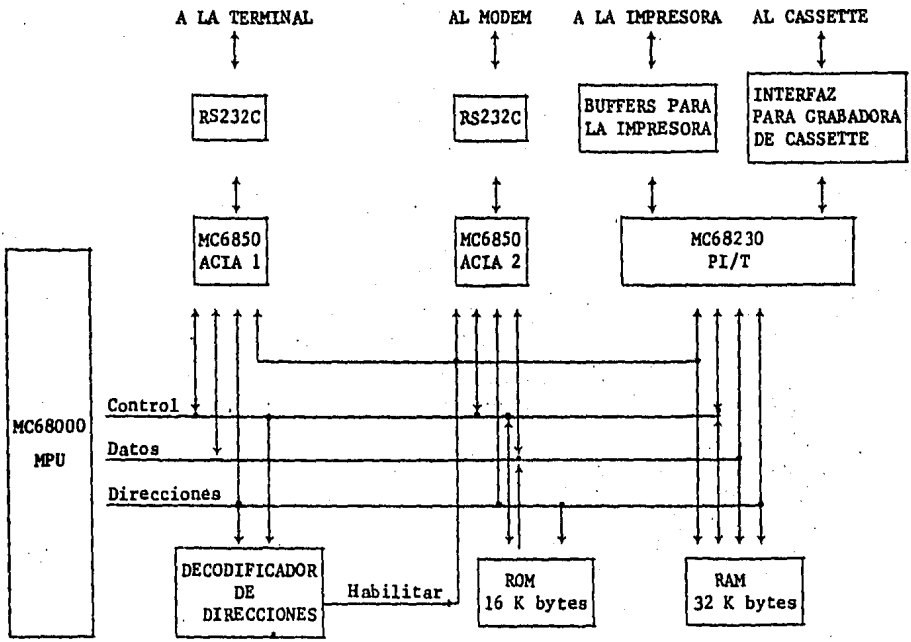


Figura 6.1 Diagrama funcional de bloques del Sistema MEX68KECB/D2.

dentro del programa.

6.3.1 IMPLEMENTACION DE OPERACIONES EN C.G.

A partir de un polinomio generador $g(x)$ de grado m, primitivo e irreductible, se obtiene la tabla de los elementos del campo finito, $CG(2^m)$, cuyos valores se guardan en memoria con su representación binaria o vectorial de m bits.

Dentro del programa estos 2^m elementos no se manejan con su representación binaria, sino por medio de su logaritmo (ver apéndice A); es decir se realiza un mapeo de los elementos del $CG(2^m)$ a logaritmos base α que corresponden a su posición dentro de la tabla. Asi por ejemplo para hacer referencia al elemento α^3, se considera el número hexadecimal $03 como apuntador. El elemento nulo ($\alpha^{-\infty}$) se representa por conveniencia mediante $FF; de manera que al sumar el logaritmo de un elemento a la dirección inicial de la tabla más uno, se obtiene la dirección de la localidad de memoria en donde se encuentra dicho elemento en su representación binaria.

A continuación se describe la forma en que se realizan las operaciones básicas en lenguaje ensamblador, las cuales fueron implementadas como subrutinas.

La suma de dos elementos se realiza mediante la subrutina de suma que se muestra en la figura 6.2. Los sumandos se encuentran originalmente en los registros de datos D0 y D1, los cuales contienen los logaritmos de los elementos a sumar representados por 8 bits. El resultado de la suma aparecerá en D0. Como se muestra en la figura, los pasos de ésta subrutina son los siguientes:

Salvar en la pila (stack) el contenido de los registros utilizados. Obtener la representación binaria de cada sumando a partir de su logaritmo como se mencionó anteriormente. Realizar la suma de los dos vectores módulo 2. Encontrar el logaritmo correspondiente al vector resultante mediante la búsqueda de su posición en la tabla de elementos. Transferir el logaritmo a D0 y recuperar los registros guardados temporalmente en la pila.

En la subrutina de multiplicación de elementos, figura 6.3, se emplean también los registros D0 y D1 para los operandos y el resultado de la opera

ción se guarda en D0. Esta subrutina se realiza de la siguiente forma:

Salvar en la pila el registro D1. Comparar los operandos con el elemento nulo ($FF), si alguno es igual, el producto es cero y sale de la subrutina, D0 con $FF; en caso contrario se realiza la suma algebraica, módulo 2^m-1, de los logarítmos (es decir el producto de los elementos), obteniéndose finalmente el resultado en D0. Por último recuperar D1 de la pila.

Para describir como se realiza la multiplicación de polinomios, veamos como se manejan éstos dentro del programa.

Los coeficientes de los polinomios siempre se guardan en memoria ordenados de mayor a menor potencia, esto es, si se tiene por ejemplo el polinomio $P(x) = \alpha^3 + x + \alpha^5 x^2 + \alpha^2 x^4$ que se guarda en memoria a partir de la localidad $1000, el contenido de la porción de memoria empleada será entonces:

Localidad	Contenido
$ 1000	$ 03
$ 1001	$ 00
$ 1002	$ 05
$ 1003	$ FF
$ 1004	$ 02

de manera que para hacer referencia dentro del programa, a un polinomio almacenado en memoria necesitamos conocer su grado y la dirección en memoria a partir de la cual se guarda.

La subrutina de multiplicación de polinomios, figura 6.4, emplea como datos de entrada las direcciones de los polinomios a multiplicar en los registros A0 y A1, cuyos grados correspondientes deberán estar en D5 y D6 respectivamente. A6 deberá apuntar a la localidad a partir de la cual se guardará el polinomio resultante, cuyo grado será calculado en la subrutina y aparecerá en D4. Las etapas de esta subrutina son las siguientes:

Guardar en la pila los registros empleados. Calcular el grado del polinomio resultante y limpiar las localidades de memoria que ocupará el nuevo polinomio. Mediante dos lazos anidados, hacer la multiplicación de polinomios

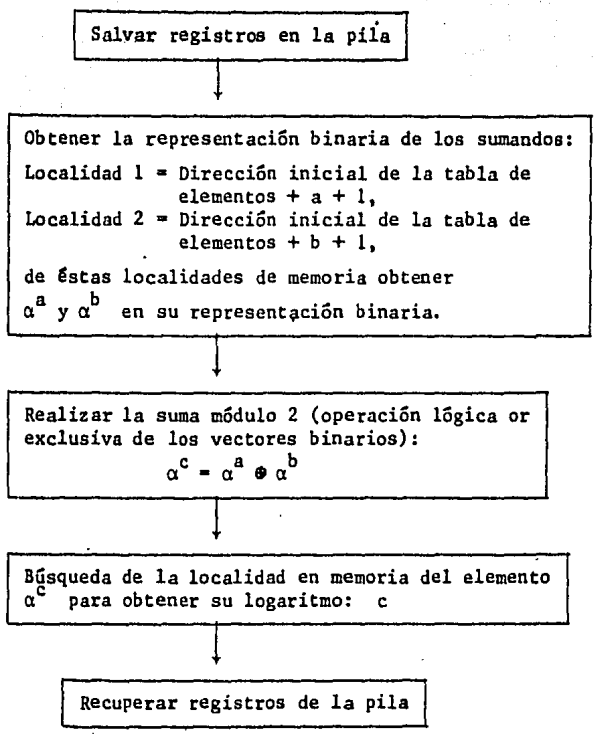


Figura 6.2 Subrutina para la suma de dos elementos en el $CG(2^m)$, α^a y α^b, cuyos logaritmos correspondientes se encuentran inicialmente en los registros D0 y D1. c es el logaritmo del elemento resultante.

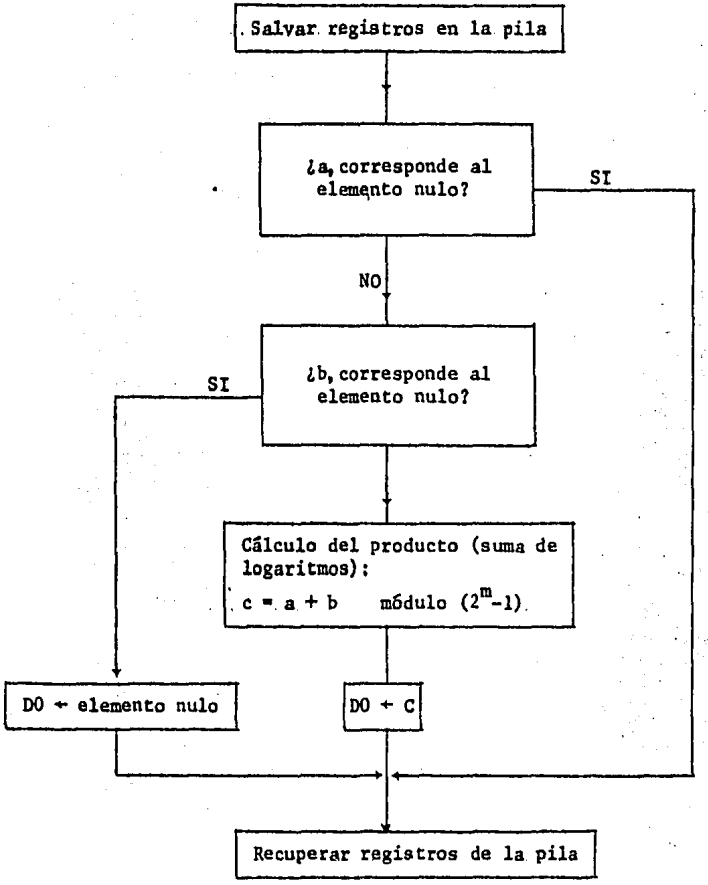


Fig: 6.3 Subrutina para la multiplicación de dos elementos en el $CG(2^m)$, α^a y α^b, cuyos logaritmos se encuentran originalmente en D0 y D1. Al terminar la subrutina D0 contiene el logaritmo del producto.

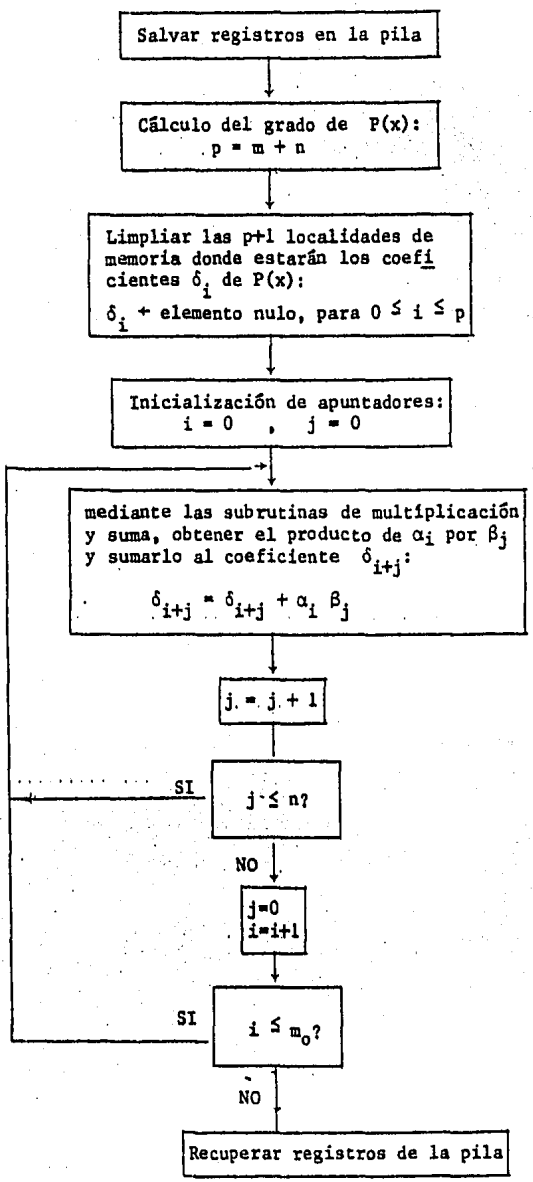


Fig. 6.4 Subrutina para multiplicar dos polinomios:
$M(x) = \alpha_0 + \alpha_1 x + \ldots + \alpha_{m_o} x^{m_o}$
$N(x) = \beta_0 + \beta_1 x + \ldots + \beta_n x^n$, donde
$P(x) = M(x)\, N(x) = \delta_0 + \delta_1 x + \ldots + \delta_{n+m_o} x^{n+m_o}$, con: α_k, β_k y $\delta_k \in CG(2^m)$, $\forall$ k

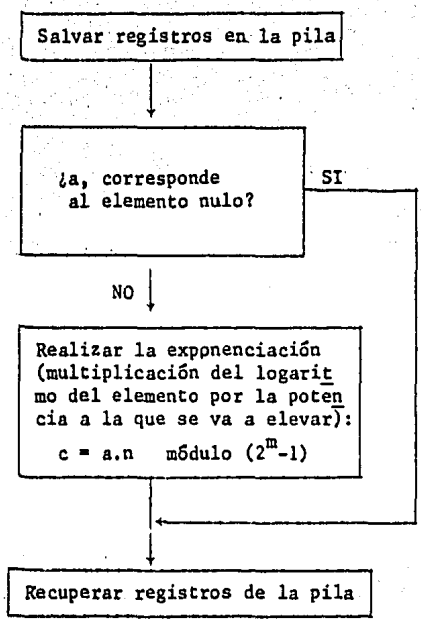


Fig. 6.5 Subrutina de exponenciación. D0 contiene el logaritmo del elemento α^a, el cual es elevado a la potencia n, indicada por D1. c es tal que $(\alpha^a)^n = \alpha^c$.

y finalmente recuperar los registros guardados en la pila. Esta subrutina emplea en su ejecución las subrutinas de suma y multiplicación antes mencionadas.

Por último en la figura 6.5 tenemos la subrutina de exponenciación. En esta subrutina el elemento cuyo logaritmo está en D0, es elevado a la potencia indicada por D1 y el resultado de la exponenciación aparece en D0. Esta subrutina se realiza de la siguiente manera:

Salvar en la pila los registros empleados. Comparar la base de la exponenciación con el elemento nulo, si son iguales termina la subrutina; en caso contrario se efectúa el producto del logaritmo del elemento por el exponente al que va a ser elevado. Este producto se realiza módulo 2^m-1. Por último se recuperan los registros guardados en la pila.

6.3.2 DESCRIPCION DEL PROGRAMA

El programa implementado en el sistema microprogramado MEX68KECB, para simular códigos de Reed-Solomon, tiene una extensión total de 2200 bytes de código, considerando también el espacio ocupado por las subrutinas. La tabla 6.1 ilustra la localización en memoria del programa principal y sus subrutinas.

Como datos de entrada al programa deben proporcionarse, en el rango de memoria entre las direcciones $900-$906, los parámetros del código. Dichos datos son leídos por el programa en base hexadecimal. La tabla 6.2 muestra las localidades y los contenidos de la tabla de datos manipulados por el programa principal. Los datos ocupan en total 2552 bytes de memoria.

Una vez introducidos en memoria los datos correspondientes a los parámetros del código, debe construirse el polinomio de información a codificar. Dicho polinomio estará dado por los logaritmos de los símbolos de información correspondientes, comenzando a partir de la localidad $A01 (ver tabla 6.2). Es importante aclarar que el programa fué implementado con propósitos de simulación de códigos de Reed-Solomon, y por lo tanto el usuario construye tanto la información a codificar como el patrón de errores y borrados. En el caso de una implementación a un canal de comunicación, al recibirse la información, el procesador deberá efectuar un mapeo entre los símbolos de m bits y su co

rrespondiente logaritmo dentro del campo de Galois en cuestión.

Antes de ejecutar la decodificación, deberá darse también el número de borrados y su respectiva posición, en las localidades indicadas en la tabla 6.2. Las posiciones de los borrados deben estar en el rango $0\text{-}(N\text{-}1)_{16}$, donde N es la longitud del código. Nuevamente cabe señalar que, en el caso de una implementación en un canal de comunicaciones, el demodulador indicará la presencia de un borrado dentro de la palabra recibida y procederá a almacenar la posición correspondiente en memoria. La palabra recibida deberá introducirse en las localidades correspondientes.

El funcionamiento del programa será entonces de la siguiente manera: Una vez almacenados en memoria los parámetros del código y el polinomio de información a codificar, el programa es ejecutado. El programa realiza la inicialización de registros y el cálculo del campo de Galois correspondiente, para posteriormente hacer la codificación del polinomio de información, por recurrencia con H(x). Tanto el campo de Galois como el polinomio de paridad son almacenados en memoria (ver tabla 6.2) al terminar esta etapa.

Antes de proceder a la decodificación, el programa debe detenerse para permitir -como se mencionó anteriormente- la introducción tanto del número de borrados y sus posiciones, como del polinomio recibido en las localidades de memoria correspondientes.

Una vez introducidos los datos mencionados, el programa proseguirá su ejecución indicando, al final de la misma, el resultado de la decodificación -exitosa, imposible, errónea o no hubo errores- usando las subrutinas de manipulación de carácteres ASCII.

Con respecto a las diferentes etapas del programa, éstas siguen los mismos procedimientos señalados en el capítulo 4, y el diagrama de flujo del programa es el mismo que el ilustrado en el párrafo D.7 del apéndice D, con las modificaciones pertinentes para su ejecución en el sistema microprogramado. El listado del programa y las subrutinas se presenta en el apéndice B.

Para una mejor comprensión de lo expuesto en este párrafo, se ejecutó el programa usando los mismos datos del ejemplo visto en el capítulo 4, a partir

del párrafo 4.2. El contenido de las localidades de memoria después de la ejecución se presenta en el listado 6.1.

Finalmente, debe observarse que la implementación en el sistema microprogramado, además de servir en la simulación de Códigos de Reed-Solomon, proporciona una herramienta importante para evaluar comparativamente el desempeño de los algoritmos empleados; y para servir de punto de partida en una implementación posterior del programa en memoria permanente (ROM), con miras a su aplicación en la simulación de sistemas digitales de comunicaciones.

6.4 MEDICION DE TIEMPOS DE DECODIFICACION

Con el objeto de disponer de una medida del desempeño del programa implementado en el sistema MEX68KECB, se incluyeron en el programa instrucciones que permitieran contabilizar el número de ciclos invertidos en cada etapa de la simulación. Para lograr lo anterior, se programó el circuito integrado MC68230 para funcionar como un contador de intervalos de tiempo controlado por programa (software).

El MC68230 consiste en una interfaz paralelo y un temporizador que incluye un contador de 24 bits, y es direccionado por el sistema mediante un grupo de registros de lectura y/o escritura de 8 bits cada uno. Para la aplicación en cuestión, los registros empleados y sus direcciones correspondientes son:

Dirección		Registro
$10021	TCR:	Registro de control del temporizador.
$10027	CPRH:	Registro de cuenta inicial, parte más significativa (MSB).
$10029	CPRM:	Registro de cuenta inicial, parte media (MB).
$1002B	CPRL:	Registro de cuenta inicial, parte menos significativa (LSB).
$1002F	CNTRH:	Registro de lectura de cuenta, MSB.
$10031	CNTRM:	Registro de lectura de cuenta, MB.
$10033	CNTRL:	Registro de lectura de cuenta, LSB.

El significado de los bits del registro de control del temporizador (TCR) se describe a continuación.

Dirección	Contenido
$1FF4-$205F	Inicialización de registros. Cálculo del campo de Galois. Cálculo del polinomio de paridad.
$2060-$20F3	Codificación por recurrencia con H(x).
$20F4-$2189	Cálculo de los componentes del síndrome.
$218A-$225D	Cálculo del polinomio localizador de borrados.
$225E-$22BF	Cálculo del síndrome de Forney.
$22C0-$2491	Cálculo del polinomio localizador de errores (Algoritmo de Berlekamp-Massey).
$2492-$2545	Búsqueda de Chien.
$2546-$2595	Cálculo del polinomio evaluador de erratas.
$2596-$2621	Evaluación de erratas.
$2622-$26FD	Corrección de la palabra recibida.
Subrutinas:	
$7000-$7037	Suma en CG(q).
$7038-$705B	Multiplicación en CG(q).
$705C-$709B	Multiplicación de polinomios sobre CG(q).
$709C-$70D1	Exponenciación en CG(q).
$2800-$2849	Envío de mensajes a terminal.
$4000-$4071	Tabla de caracteres ASCII.

Tabla 6.1 Localización en memoria del programa y subrutinas para la simulación de códigos de Reed-Solomon, mediante el sistema microprogramado MEX68KECB.

Dirección	Contenido
\$900	N-1: Longitud del código menos la unidad (Dato).
\$901	N_{nom}: Longitud nominal del código (Dato).
\$902	K_{na}-1: Dimensión nominal del código menos la unidad (Dato).
\$903	K: Dimensión del código (Dato).
\$904	d-1: Número de símbolos de redundancia (Dato).
\$905	t: Capacidad de corrección (Dato).
\$906	g(x): Polinomio generador de CG(q) (Dato).
\$907	Tabla de elementos de CG(q).
\$987	H(x): Polinomio de paridad del código.
\$A01	I(x): Polinomio de información.
\$A7A	C(x): Polinomio de código.
\$AF9	R(x): Polinomio recibido (Dato).
\$B79	S_i, $i \in [1, 2t]$: Componentes del síndrome.
\$BB8	b: Número de borrados (Dato).
\$BB9	X_{bi}, $i \in [1,b]$: Posiciones de los borrados (Dato).
\$BF9	$\tau(x)$: Polinomio localizador de borrados.
\$C79	T(x): Síndrome de Forney.
\$CBB \$CDC ⋮	$\sigma(x)^{(-1)}$, $\sigma(x)^{(0)}$, ...: Polinomios localizadores de errores, obtenidos en las etapas del algoritmo de Berlekamp-Massey, localizados cada #\$21 localidades.
\$113E ⋮	d_{-1}, d_0, ...: Discrepancias obtenidas en las etapas del algoritmo de Berlekamp-Massey.

Tabla 6.2 Localización en memoria de la tabla de datos empleados por el programa simulador de códigos de Reed-Solomon.

Dirección	Contenido
$1164	ℓ_{-1}, ℓ_0, ...: Grados de los polinomios obtenidos en las etapas del algoritmo de Berlekamp-Massey.
$1189	e: Número de errores.
$118A	X_i, $i \in 1,(e+b)$: Vector de erratas, arreglado de manera tal que se tienen las posiciones de los errores seguidas de las de los borrados.
$11EA	$\Omega(x)$: Polinomio evaluador de erratas.
$122B	Y_i, $i \in 1, (e+b)$: Valores de las erratas.
$126B	$\hat{C}(x)$: Polinomio decodificado.
$12DD	Localidad auxiliar empleada en la evaluación de las erratas.
$12E0	Area empleada para la obtención de polinomios.
$1400	Cuenta de la codificación.
$1403	Cuenta del cálculo de los componentes del síndrome.
$1406	Cuenta del cálculo del polinomio localizador de borrados y del síndrome de Forney.
$1409	Cuenta en el algoritmo de Berlekamp-Massey.
$140C	Cuenta de la búsqueda de Chien, cálculo del polinomio evaluador de erratas, evaluación de las erratas y corrección del polinomio recibido.

Tabla 6.2 Localización en memoria de la tabla de datos empleados por el programa simulador de Códigos de Reed-Solomon. (Continuación).

```
000900   0E 0F 08 09 06 03 19 00  01 02 04 08 09 0B 0F 07  ................
000910   0E 05 0A 0D 03 06 0C 01  FF FF FF FF FF FF FF FF  ................

000987   09 0E 0C 0B 01 05 05 02  0C 00 FF FF FF FF FF FF  ................

000A01   01 04 06 FF 03 05 06 0D  0A FF FF FF FF FF FF FF  ................

000A7A   05 03 0D 01 07 04 01 04  06 FF 03 05 06 0D 0A FF  ................

000AF9   07 03 0D 0E 07 01 01 04  06 FF 03 05 0B 0D 0A FF  ................

000B79   0B 05 02 02 00 0E FF FF  FF FF FF FF FF FF FF FF  ................

000BB8   02 00 05 FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................

000BF9   00 0A 05 FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................

000C79   00 07 06 07 0B 09 00 FF  04 FF FF FF FF FF FF FF  ................

000CBB   00 FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000CDC   00 FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000CFD   00 07 FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000D1E   00 04 FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000D3F   00 0B 09 FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000D60   00 05 00 FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................

00113E   00 07 00 02 00 FF FF FF  FF FF FF FF FF FF FF FF  ................

001164   00 00 01 01 02 02 FF FF  FF FF FF FF FF FF FF FF  ................

0011B9   02 0C 03 00 05 FF FF FF  FF FF FF FF FF FF FF FF  ................

0011EA   00 0E 0B 0B 00 FF FF 08  00 FF FF FF FF FF FF FF  ................

00122B   01 08 0E 05 FF FF FF FF  FF FF FF FF FF FF FF FF  ................

00126B   05 03 0D 01 07 04 01 04  06 FF 03 05 06 0D 0A FF  ................
```

Listado 6.1 Contenido de las localidades de memoria después de la simulación del código de Reed-Solomon (15,9,7) (Véase listado 4.1), mediante el programa implementado en el sistema microprogramado MEX68KECB.

Los bits 5-7 configuran las líneas $\overline{TOUT}$ y $\overline{TIACK}$ del puerto C del MC68230 y, en el caso de medición de intervalos de tiempo, estas líneas no son empleadas. El bit 4 especifica si el contador recibe datos del registro de cuenta inicial (CPRH-L) o si continúa contando al detectar un cruce con cero en la cuenta. El bit 3 no se usa. Los bits 1 y 2 sirven para configurar la trayectoria entre las líneas de reloj del sistema, o reloj externo, y el controlador del contador. El bit 0 por su parte sirve para habilitar o deshabilitar al temporizador. El TCR puede ser leído o escrito en cualquier momento.

Los registros de cuenta inicial (CPRH-L) son un grupo de tres registros de 8 bits, y son utilizados para almacenar los datos a transferir al contador. Los registros de lectura de cuenta (CNTRH-L) constituyen un grupo de tres direcciones de 8 bits en las que puede leerse el estado del contador.

Para efectuar la medición del intervalo de tiempo que toma determinado evento, el microprocesador escribe, antes de iniciado el evento, el valor inicial de la cuenta #$FFFFFF en los registros de cuenta inicial (CPRH-L). Al iniciarse el evento, el microprocesador escribe el valor #$11 en el registro de control del temporizador (TCR), habilitando a éste. Cuando el evento ha concluído, el microprocesador detiene al temporizador (Bit 0 del TCR=0) y procede entonces a leer los registros de lectura de cuenta (CNTRH-L). El contador del temporizador funciona decrementando la cuenta, por lo que para obtener el valor real de la cuenta es necesario efectuar una resta entre #$FFFFFF y el contenido de los registros de lectura de cuenta.

Empleando este procedimiento, se realizaron ejecuciones del programa simulador empleando patrones diversos de errores y borrados, con posiciones y valores aleatorios, dentro de la capacidad teórica del código. Se obtuvieron así las matrices de tiempo de decodificación, para corrección de errores y borrados, de los códigos RS(15,9,7) y RS(31,25,7), las cuales se presentan en las tablas 6.3 y 6.4. Los valores presentados en las mismas corresponden al valor esperado del tiempo de decodificación. Ya que dicho tiempo varía según sean las posiciones y los valores de las erratas, se efectuaron 10 ejecuciones para cada caso de e errores y b borrados, variando aleatoriamente la posición y el valor de los mismos. En las tablas mencionadas se presenta la variación máxima de tiempo registrada en las pruebas, así como el

tiempo de codificación.

Los resultados obtenidos permiten hacer un análisis comparativo de la complejidad de las etapas involucradas en la decodificación de diferentes códigos de Reed-Solomon. De esta manera, el programa permite observar la relación existente entre los parámetros del código y la complejidad de la decodificación.

En los casos presentados en las tablas 6.3 y 6.4 puede observarse que al variar la longitud del código, varía en proporción casi directa el tiempo invertido en la decodificación.

e ↓ \ b →	0	1	2	3	4	5	6
0	19.19	24.28	27.17	31.63	37.00	43.36	51.09
1	30.94	34.78	38.35	43.37	50.75		
2	37.99	42.18	46.61				
3	46.42						

Variación máxima = ± 4.5 Tiempo de codificación = 8.63

Tabla 6.3 Matriz de tiempos de decodificación para el código RS(15,9,7).
Tiempos en milisegundos.

e ↓ \ b →	0	1	2	3	4	5	6
0	52.03	58.22	62.72	68.38	75.70	84.15	94.56
1	75.73	80.47	85.79	92.45	100.32		
2	89.61	95.86	100.87				
3	104.33						

Variación máxima = ± 5.6 Tiempo de codificación = 32.55

Tabla 6.4 Matriz de tiempos de decodificación para el código RS(31,25,7).
Tiempos en milisegundos.

CAPITULO 7

CONCLUSIONES Y OBSERVACIONES

CAPITULO 7

CONCLUSIONES Y OBSERVACIONES

Lo conseguido con la realización de esta tésis puede ser analizado desde varios puntos de vista. Primero, se logró implementar un paquete de programas en minicomputadora que constituye un apoyo importante en el estudio de diversos esquemas de codificación de canal. Segundo, se analizó el comportamiento de los códigos de Reed-Solomon en lo que se refiere a la probabilidad de los diversos eventos a la salida del decodificador, análisis fundamental en lo que a sistemas de comunicación concierne. Por último, pero no por ello menos importante, se implementó el esquema de codificación de Reed-Solomon en un sistema microprogramado, basado en el microprocesador MC68000. Gracias a esto, se pudieron efectuar pruebas para determinar los tiempos de codificación y de decodificación para ciertos códigos. Mediante el esquema implementado se proporciona una herramienta fundamental para el estudio de los códigos de Reed-Solomon, ya que permite la simulación de una gran variedad de códigos por ser parametrable. Además, usando los resultados obtenidos en la implementación, se podrán tener los elementos necesarios para lograr la realización de un codificador de Reed-Solomon que trabaje a tasas de transmisión adecuadas para su aplicación en un canal real de comunicaciones.

Como observación final, debe destacarse el acelerado avance de la tecnología en los últimos años, lo que permitirá el uso de técnicas de codificación cada vez mas complejas. Por esta razón, esta tésis pretende servir de apoyo a las personas interesadas en adentrarse en el área de la codificación de canal la cuál, como se mencionó, tendrá una importancia fundamental en el diseño de los sistemas de comunicaciones del futuro.

APENDICE A

CAMPOS FINITOS Y ESPACIOS VECTORIALES TEOREMAS BASICOS

APENDICE A
CAMPOS FINITOS Y ESPACIOS VECTORIALES
TEOREMAS BASICOS

A.1 INTRODUCCION. Desde un punto de vista teórico, en la construcción y en la decodificación de la mayoría de los códigos cíclicos conocidos se emplean los campos finitos. También son importantes en muchas áreas de las matemáticas. En la primera parte de este apéndice se dá una breve descripción de ellos y en la segunda parte se repasan algunos de la teoría básica de espacios vectoriales.

A.2 CAMPOS FINITOS. [2,24]. Un campo es un conjunto de elementos, que incluye el 1 y el 0, en el cual cualquier par de elementos puede ser sumado o multiplicado para dar un resultado único en el campo. La suma y la multiplicación son asociativas y conmutativas, y la multiplicación es distributiva sobre la suma. Además cada elemento diferente de cero en el campo tiene su inverso aditivo y su inverso multiplicativo.

El orden de un campo se define como el número de elementos que contiene. Si el orden es finito lo llamamos campo finito o campo de Galois CG.

Por ejemplo, los campos CG(p), donde p es un número primo, están formados por los enteros módulo p. El orden del campo es p y las cuatro operaciones básicas se realizan módulo p.

Cuando un campo C_1 es un subconjunto de un campo C_2, décimos que C_2 es una extensión del campo C_1. E.g., el campo de los números reales es una extensión del campo de los números racionales.

Un polinomio $\Pi(x)$ es irreductible o primo sobre un CG(p), si no es el producto de dos polinomios de menor grado con coeficientes en el CG(p).

Si tenemos un polinomio $\Pi(x)$ irreductible sobre un CG(p) y su grado es m, entonces el conjunto de todos los polinomios en x con grado menor que m y coeficientes en el CG(p) forman un campo de orden p^m. Las operaciones en el $CG(p^m)$ se realizan módulo $\Pi(x)$.

Para ilustrar lo anterior, sea m=3 y p=2, es decir tenemos un CG(2) = {0,1}, y con un polinomio irreductible $\Pi(x) = x^3 + x + 1$ formamos un $CG(2^3)$, cuyos elementos pueden escribirse de varias formas como se muestra en la figura A.1.

Vector	Polinomio	Potencias de α	Logaritmo
0 0 0	0	0	$-\infty$
0 0 1	1	1	0
0 1 0	α	α	1
1 0 0	α^2	α^2	2
0 1 1	$\alpha+1$	α^3	3
1 1 0	$\alpha^2+\alpha$	α^4	4
1 1 1	$\alpha^2+\alpha+1$	α^5	5
1 0 1	α^2+1	α^6	6
0 0 1	1	$\alpha^7=1$	0

Fig. A.1 $CG(2^3)$ generado por $\alpha^3 + \alpha + 1 = 0$

Vemos que los elementos diferentes de cero del $CG(2^3)$ forman un grupo cíclico de orden 7. (Un grupo multiplicativo finito es cíclico si está formado por los elementos $1, \alpha, \alpha^2, \ldots, \alpha^{r-1}$, con $\alpha^r=1$, donde α es un generador del grupo).

A α (o a cualquier otro elemento generador de éste grupo cíclico) se le llama elemento primitivo del $CG(2^3)$ y al polinomio que tiene ese elemento primitivo como raíz se le llama polinomio primitivo. (No todos los polinomios irreductibles son primitivos).

Cualquier elemento γ diferente de cero del $CG(2^3)$ puede ser escrito como una potencia de α

$$\gamma = \alpha^i \qquad 0 \leq i \leq 6 \qquad \text{con } \alpha^0 = \alpha^7 = 1$$

donde a i se le conoce como el logaritmo de γ. Para el elemento cero, $0 = \alpha^{-\infty}$.

La representación de los elementos del $CG(2^3)$ en las primeras dos columnas de la figura A.1 es más conveniente para realizar adiciones o sustrac

ciones, mientras que para la multiplicación y división lo es la representación de los elementos como potencias de α o como logaritmos.

Para el ejemplo en cuestión, por ser el $CG(2^3)$ una extensión del $CG(2)$, las operaciones de adición y sustracción son idénticas (se realizan módulo 2, i.e., $1 \pm 1 = 0$, $1 \pm 0 = 1$, $0 \pm 0 = 0$).

Ejemplos:

para sumar $(001) + (011)$

$$\begin{array}{l} 001 = \alpha^{\circ} \\ + \underline{011 = \alpha^{3}} \\ 010 = \alpha \end{array} \qquad \text{entonces} \quad \alpha^{\circ} + \alpha^{3} = \alpha = 010,$$

para multiplicar $(110) \cdot (111)$

$(110) = \alpha^{4}$ y $(111) = \alpha^{5}$

$\alpha^{4} \cdot \alpha^{5} = \alpha^{4+5} = \alpha^{9}$ dado que $\alpha^{7} = \alpha^{\circ} = 1$

entonces $\alpha^{9} = \alpha^{9-7} = \alpha^{2} = 100,$

para dividir $(011) / (101)$

$(011) = \alpha^{3}$

$(101) = \alpha^{6}$

$\alpha^{3}/\alpha^{6} = \alpha^{3-6} = \alpha^{-3}$ pero $\alpha^{7} = \alpha^{\circ} = 1$

entonces $\alpha^{-3} = \alpha^{7-3} = \alpha^{4} = 110,$

para hallar el recíproco de $(111) = \alpha^{5}$

$$(111)^{-1} = (\alpha^{5})^{-1} = \alpha^{-5} = \alpha^{7-5} = \alpha^{2} = 100$$

Al menor entero positivo p para el cual $\sum_{i=1}^{p} 1 = 0$ en el campo se le llama característica del campo (p es siempre un número primo). Por ejemplo el $CG(2^3)$ tiene característica 2, i.e., $1+1=0$.

Cada elemento β de un campo C de orden p^m satisface la igualdad:

$$\beta^{p^m} = \beta$$

o equivalentemente es una raíz de la ecuación:

$$x^{p^m} = x$$

de manera que:

$$x^{p^m} - x = \prod_{\beta \in C} (x-\beta) \qquad (A.1)$$

Sea $q = p^m$ la ecuación A.1 implica que todo elemento del CG(q) satisface la ecuación:

$$x^q - x = 0 \qquad (A.2)$$

Este polinomio tiene sus coeficientes en el CG(p) y es mónico, i.e., su primer coeficiente es 1. β sin embargo puede satisfacer una ecuación de menor grado que A.2.

El polinomio mínimo sobre el CG(p) de β es el polinomio mónico de menor grado M(x) con coeficientes en el CG(p) tal que:

$$M(\beta) = 0 \qquad (A.3)$$

Algunas propiedades de los polinomios mínimos son las siguientes:

a) M(x) es irreductible
b) $x^{p^m} - x$ es divisible por M(x)
c) El grado de M(x) $\leq m$
d) El polinomio mínimo de un elemento primitivo del $CG(p^m)$ tiene grado m. Tal polinomio es llamado polinomio primitivo.
e) β y β^p tienen el mismo polinomio mínimo.

A los elementos del campo que tienen un mismo polinomio mínimo se les llama elementos conjugados. Aplicando la propiedad e) para el $CG(2^3)$ tenemos que:

α, α^2, $(\alpha^2)^2 = \alpha^4$ ($(\alpha^4)^2 = \alpha^8 = \alpha$ nuevamente) tienen el mismo polinomio mínimo, de la misma manera α^3, α^6, $\alpha^{12} = \alpha^5$ ($(\alpha^5)^2 = \alpha^{10} = \alpha^3$ nuevamente) tienen el mismo polinomio mínimo.

Se observa que las potencias de α caen en conjuntos separados a los que se les llama arreglos cíclicos o translados ciclotômicos. La operación de multiplicar por p divide a los enteros módulo $p^m - 1$ en translados ciclotómicos por ejemplo:

los translados ciclotômicos módulo $2^3 - 1 = 7$ son:

$C_0 = \{0\}$
$C_1 = \{1,2,4\}$
$C_3 = \{3,6,5\}$

y los translados módulo $2^4 - 1 = 15$ son:

$C_0 = \{0\}$
$C_1 = \{1,2,4,8\}$
$C_3 = \{3,4,12,9\}$
$C_5 = \{5,10\}$
$C_7 = \{7,14,13,11\}$

Utilizando la notación C_s, el subíndice s es el número menor contenido en el translado C_s.

Sea ahora $M^{(i)}(x)$ el polinomio mínimo del elemento α^i perteneciente al $CG(p^m)$. Tenemos entonces una última propiedad de los polinomios mínimos:

f) Si i está en C_s entonces:

$$M^{(i)}(x) = \prod_{j \in C_s} (x - \alpha^j) \qquad \text{(A.4)}$$

y de la ecuación A.1.

$$x^{p^m - 1} - 1 = \prod_{s} M^{(s)}(x). \qquad \text{(A.5)}$$

$x^{p^m} - x$ es el producto de todos los polinomios mónicos irreductibles cuyos grados dividen a m.

Por ejemplo a partir de A.5 con p=2 y m=1, se tiene:

$$x^2 + x = x(x+1)$$

por lo tanto hay dos polinomios irreductibles de grado 1: x y x+1, que son los polinomios mínimos de 0 y 1 en el CG(2) respectivamente.

Para m=2 tenemos:

$$x^{2^2} + x = x^4 + x = x(x+1)\ (x^2 + x + 1)$$

por lo tanto hay un polinomio irreductible de grado 2, $x^2 + x + 1$. Los polinomios mínimos en el $CG(2^2)$ son:

Elemento	Pol. mínimo
0	x
1	$M^{(0)}(x) = x + 1$
α, α^2	$M^{(1)}(x) = x^2 + x + 1$

Para m=3 tenemos:

$$x^{2^3} + x = x^8 + x = x(x+1)\ (x^3 + x + 1)\ (x^3 + x^2 + 1)$$

entonces hay dos polinomios mínimos de grado 3, $x^3 + x + 1$ y $x^3 + x^2 + 1$. En el $CG(2^3)$ generado por $\alpha^3 + \alpha + 1 = 0$ se tiene:

Elemento	Pol. primitivo
0	x
1	$M^{(0)}(x) = x + 1$
α, α^2, α^4	$M^{(1)}(x) = x^3 + x + 1$
α^3, α^6, α^5	$M^{(3)}(x) = x^3 + x^2 + 1$

Como último ejemplo para m=4 tenemos:

$x^{2^4} + x = x(x+1)\,(x^2+x+1)\,(x^4+x+1)(x^4+x^3+1)\,(x^4+x^3+x^2+x+1)$ y para el $CG(2^4)$ generado por $\alpha^4 + \alpha + 1 = 0$ se tiene:

Elemento	Pol. primitivo
0	x
1	$M^{(0)}(x) = x + 1$
$\alpha, \alpha^2, \alpha^4, \alpha^8$	$M^{(1)}(x) = x^4 + x + 1$
$\alpha^3, \alpha^6, \alpha^{12}, \alpha^9$	$M^{(3)}(x) = x^4 + x^3 + x^2 + x + 1$
α^5, α^{10}	$M^{(5)}(x) = x^2 + x + 1$
$\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}$	$M^{(7)}(x) = x^4 + x^3 + 1$

A.3 NOTAS SOBRE ESPACIOS VECTORIALES. [31]. Para definir lo que es un espacio vectorial definamos primeramente un grupo.

Un grupo es un conjunto de elementos para las cuales se define una operación. Este conjunto tiene las siguientes propiedades: es cerrado, es asociativo, existe un elemento identidad único y cada elemento en el grupo tiene un elemento inverso también único, se llama grupo Abeliano si además satisface la ley conmutativa.

Un conjunto V de elementos es llamado espacio vectorial sobre un campo C si satisface los siguientes axiomas para cualesquiera $c,d \in C$ y $\bar{u}, \bar{v} \in V$.

a) El conjunto V es un grupo Abeliano bajo la suma.
b) Para cualquier $\bar{v}$ esta definido $c\bar{v} \in V$
c) Ley distributiva: $c(\bar{u} + \bar{v}) = c\bar{u} + c\bar{v}$
y tambien $(c+d)\,\bar{v} = c\bar{v} + d\bar{v}$
d) Ley asociativa $(cd)\,\bar{v} = c(d\bar{v})$ y $1\bar{v} = \bar{v}$

El conjunto de todas las combinaciones lineales de un conjunto de vectores $\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_k$ de un espacio vectorial V, es un subespacio vectorial de V.

Al número de vectores linealmente independientes que generan un espacio

se le llama dimensión del espacio.

Al conjunto de k vectores linealmente independientes (L.I.) que generan un espacio de dimensión k se le llama base del espacio.

Si V es un espacio de dimensión k cualquier conjunto de k vectores L.I. es una base de V.

Dos vectores son ortogonales si su producto punto es cero, i.e.

$$(a_1, a_2, \ldots, a_n)\cdot(b_1, b_2, \ldots, b_n) = a_1b_1 + a_2b_2 + \ldots + a_nb_n = 0$$

El conjunto de todos los vectores ortogonales a un subespacio V_1 forma un subespacio V_2 que se conoce como espacio nulo de V_1. (Al conjunto de vectores de largo n se le llama código lineal de bloque sí y solo sí es un subespacio del espacio vectorial de vectores de largo n.).

APENDICE B

LISTADO DEL PROGRAMA IMPLEMENTADO EN EL SISTEMA MICROPROGRAMADO

```
001FF4    4282              CLR.L   D2
001FF6    4283              CLR.L   D3
001FF8    42B0              CLR.L   D0
001FFA    41F80907          LEA.L   $00000907,A0
001FFE    10380906          MOVE.B  $00000906,D0
002002    223C0000001F      MOVE.L  #31,D1
002008    0300              BTST    D1,D0
00200A    56C9FFFC          DBNE.L  D1,$002008
00200E    4218              CLR.B   (A0)+
002010    10BC0001          MOVE.B  #1,(A0)
002014    1418              MOVE.B  (A0)+,D2
002016    E30A              LSL.B   #1,D2
002018    0302              BTST    D1,D2
00201A    6702              BEQ.S   $00201E
00201C    B102              EOR.B   D0,D2
00201E    1082              MOVE.B  D2,(A0)
002020    0C100001          CMP.B   #1,(A0)
002024    66EE              BNE.S   $002014
002026    4DF812E2          LEA.L   $000012E2,A6
00202A    41F812E0          LEA.L   $000012E0,A0
00202E    43F80987          LEA.L   $00000987,A1
002032    14380904          MOVE.B  $00000904,D2
002036    16380901          MOVE.B  $00000901,D3
00203A    3A3C0001          MOVE.W  #1,D5
00203E    3C05              MOVE.W  D5,D6
002040    42280001          CLR.B   1(A0)
002044    42290001          CLR.B   1(A1)
002048    1083              MOVE.B  D3,(A0)
00204A    5303              SUBQ.B  #1,D3
00204C    1283              MOVE.B  D3,(A1)
00204E    5303              SUBQ.B  #1,D3
002050    6100500A          BSR.L   $00705C
002054    1083              MOVE.B  D3,(A0)
002056    CD49              EXG     A6,A1
002058    C946              EXG     D4,D6
00205A    B602              CMP.B   D2,D3
00205C    57C8FFF2          DBEQ.L  D3,$002050
002060    13FC00FF00010027  MOVE.B  #255,$00010027
002068    13FC00FF00010029  MOVE.B  #255,$00010029
002070    13FC00FF0001002B  MOVE.B  #255,$0001002B
002078    13FC00110001002(  MOVE.B  #17,$00010021
002080    41F80987          LEA.L   $00000987,A0
002084    4243              CLR.W   D3
002086    16380904          MOVE.B  $00000904,D3
00208A    43F80A7A          LEA.L   $000C0A7A,A1
00208E    45F80A01          LEA.L   $00000A01,A2
002092    14380902          MOVE.B  $00000502,D2
002096    47F13000          LEA.L   0(A1,D3.W),A3
00209A    17B220002000      MOVE.B  0(A2,D2.W),0(A3,D2.W)
0020A0    51CAFFF8          DBF.L   D2,$00209A
0020A4    5343              SUBQ.W  #1,D3
0020A6    4242              CLR.W   D2
0020A8    3A03              MOVE.W  D3,D5
0020AA    383C00FF          MOVE.W  #255,D4
0020AE    14380902          MOVE.B  $00000902,D2
```

```
0020B2    12302000          MOVE.B  0(A0,D2.W),D1
0020B6    10315001          MOVE.B  1(A1,D5.W),D0
0020BA    61004F7C          BSR.L   $007038
0020BE    3204              MOVE.W  D4,D1
0020C0    61004F3E          BSR.L   $007000
0020C4    3800              MOVE.W  D0,D4
0020C6    5245              ADDQ.W  #1,D5
0020C8    51CAFFE8          DBF.L   D2,$002092
0020CC    13843000          MOVE.B  D4,0(A1,D3.W)
0020D0    51CBFFD4          DBF.L   D3,$0020A6
0020D4    13FC001000010021  MOVE.B  #16,$00010021
0020DC    11F90001002F1400  MOVE.B  $0001002F,$00001400
0020E4    11F9000100311401  MOVE.B  $00010031,$00001401
0020EC    11F9000100331402  MOVE.B  $00010033,$00001402
0020F4    13FC00FF00010027  MOVE.B  #255,$00010027
0020FC    13FC00FF00010029  MOVE.B  #255,$00010029
002104    13FC00FF0001002B  MOVE.B  #255,$0001002B
00210C    13FC001100010021  MOVE.B  #17,$00010021
002114    4243              CLR.W   D3
002116    16380904          MOVE.B  $00000904,D3
00211A    5343              SUBQ.W  #1,D3
00211C    4245              CLR.W   D5
00211E    41F80AF9          LEA.L   $00000AF9,A0
002122    45F80B79          LEA.L   $00000B79,A2
002126    383C00FF          MOVE.W  #255,D4
00212A    5243              ADDQ.W  #1,D3
00212C    4242              CLR.W   D2
00212E    14380900          MOVE.B  $00000900,D2
002132    0C3000FF2000      CMP.B   #255,0(A0,D2.W)
002138    6718              BEQ.S   $002152
00213A    3203              MOVE.W  D3,D1
00213C    1002              MOVE.B  D2,D0
00213E    61004F5C          BSR.L   $00709C
002142    12302000          MOVE.B  0(A0,D2.W),D1
002146    61004EF0          BSR.L   $007038
00214A    1204              MOVE.B  D4,D1
00214C    61004EB2          BSR.L   $007000
002150    1800              MOVE.B  D0,D4
002152    51CAFFDE          DBF.L   D2,$002132
002156    0C00400FF         CMP.B   #255,D4
00215A    6704              BEQ.S   $002160
00215C    1A3C0001          MOVE.B  #1,D5
002160    158430FF          MOVE.B  D4,-1(A2,D3.W)
002164    5343              SUBQ.W  #1,D3
002166    57CBFFBE          DBEQ.L  D3,$002126
00216A    13FC001000010021  MOVE.B  #16,$00010021
002172    11F90001002F1403  MOVE.B  $0001002F,$00001403
00217A    11F9000100311404  MOVE.B  $00010031,$00001404
002182    11F9000100331405  MOVE.B  $00010033,$00001405
00218A    13FC00FF00010027  MOVE.B  #255,$00010027
002192    13FC00FF00010029  MOVE.B  #255,$00010029
00219A    13FC00FF0001002B  MOVE.B  #255,$0001002B
0021A2    13FC001100010021  MOVE.B  #17,$00010021
```

```
0021AA  4A05            TST.B   D5
0021AC  67000520        BEQ.L   $0026DA
0021B0  4222            CLR.B   -(A2)
0021B2  43FB0BF9        LEA.L   $00000BF9,A1
0021B6  4243            CLR.W   D3
0021B8  47FB0BB8        LEA.L   $00000BB8,A3
0021BC  1613            MOVE.B  (A3),D3
0021BE  4A03            TST.B   D3
0021C0  670000C6        BEQ.L   $002288
0021C4  3A3C0001        MOVE.W  #1,D5
0021C8  3C05            MOVE.W  D5,D6
0021CA  41FB12E0        LEA.L   $000012E0,A0
0021CE  4DFB0C39        LEA.L   $00000C39,A6
0021D2  0C030002        CMP.B   #2,D3
0021D6  6E36            BGT.S   $00220E
0021D8  670E            BEQ.S   $0021E8
0021DA  3806            MOVE.W  D6,D4
0021DC  136B00010001    MOVE.B  1(A3),1(A1)
0021E2  4211            CLR.B   (A1)
0021E4  60000078        BRA.L   $00225E
0021E8  CD49            EXG     A6,A1
0021EA  42280001        CLR.B   1(A0)
0021EE  42290001        CLR.B   1(A1)
0021F2  12AB0001        MOVE.B  1(A3),(A1)
0021F6  10AB0002        MOVE.B  2(A3),(A0)
0021FA  61004E60        BSR.L   $00705C
0021FE  CD49            EXG     A6,A1
002200  1011            MOVE.B  (A1),D0
002202  12A90002        MOVE.B  2(A1),(A1)
002206  13400002        MOVE.B  D0,2(A1)
00220A  60000052        BRA.L   $00225E
00220E  42280001        CLR.B   1(A0)
002212  42290001        CLR.B   1(A1)
002216  12AB0001        MOVE.B  1(A3),(A1)
00221A  10AB0002        MOVE.B  2(A3),(A0)
00221E  5543            SUBQ.W  #2,D3
002220  61004E3A        BSR.L   $00705C
002224  10B33002        MOVE.B  2(A3,D3.W),(A0)
002228  CD49            EXG     A6,A1
00222A  C946            EXG     D4,D6
00222C  4A03            TST.B   D3
00222E  57CBFFF0        DBEQ.L  D3,$002220
002232  4243            CLR.W   D3
002234  1613            MOVE.B  (A3),D3
002236  3203            MOVE.W  D3,D1
002238  08030000        BTST    #0,D3
00223C  670E            BEQ.S   $00224C
00223E  1DB110001000    MOVE.B  0(A1,D1.W),0(A6,D1.W)
002244  51C9FFF8        DBF.L   D1,$00223E
002248  1806            MOVE.B  D6,D4
00224A  CD49            EXG     A6,A1
00224C  1A13            MOVE.B  (A3),D5
00224E  4246            CLR.W   D6
002250  1DB150006000    MOVE.B  0(A1,D5.W),0(A6,D6.W)
002256  5246            ADDQ.W  #1,D6
```

```
002258  51CDFFF6        DBF.L   D5,$002250
00225C  CD49            EXG     A6,A1
00225E  4240            CLR.W   D0
002260  4241            CLR.W   D1
002262  4DF80C79        LEA.L   $00000C79,A6
002266  1C13            MOVE.B  (A3),D6
002268  4245            CLR.W   D5
00226A  1A380904        MOVE.B  $00000904,D5
00226E  41F80B78        LEA.L   $00000B78,A0
002272  61004DE8        BSR.L   $00705C
002276  1016            MOVE.B  (A6),D0
002278  4201            CLR.B   D1
00227A  61004D84        BSR.L   $007000
00227E  1C80            MOVE.B  D0,(A6)
002280  1DBC00FF5001    MOVE.B  #255,1(A6,D5.W)
002286  6018            BRA.S   $0022A0
002288  4DF80C79        LEA.L   $00000C79,A6
00228C  16380904        MOVE.B  $00000904,D3
002290  1DB230003000    MOVE.B  0(A2,D3.W),0(A6,D3.W)
002296  4A03            TST.B   D3
002298  57CBFFF6        DBEQ.L  D3,$002290
00229C  1CBC00FF        MOVE.B  #255,(A6)
0022A0  13FC001000010021 MOVE.B #16,$00010021
0022A8  11F90001002F1406 MOVE.B  $0001002F,$00001406
0022B0  11F9000100311407 MOVE.B  $00010031,$00001407
0022B8  11F9000100331408 MOVE.B  $00010033,$00001408
0022C0  13FC00FF00010027 MOVE.B  #255,$00010027
0022C8  13FC00FF00010029 MOVE.B  #255,$00010029
0022D0  13FC00FF0001002B MOVE.B  #255,$0001002B
0022D8  13FC001100010021 MOVE.B  #17,$00010021
0022E0  49F80CBB        LEA.L   $00000CBB,A4
0022E4  4BF81164        LEA.L   $00001164,A5
0022E8  4240            CLR.W   D0
0022EA  10380905        MOVE.B  $00000905,D0
0022EE  19BC00FF0000    MOVE.B  #255,0(A4,D0.W)
0022F4  19BC00FF0021    MOVE.B  #255,33(A4,D0.W)
0022FA  51C8FFF2        DBF.L   D0,$0022EE
0022FE  4240            CLR.W   D0
002300  4214            CLR.B   (A4)
002302  422C0021        CLR.B   33(A4)
002306  4255            CLR.W   (A5)
002308  10380B88        MOVE.B  $00000B88,D0
00230C  47F8113E        LEA.L   $0000113E,A3
002310  4213            CLR.B   (A3)
002312  177600010001    MOVE.B  1(A6,D0.W),1(A3)
002318  3C3C0001        MOVE.W  #1,D6
00231C  45F812E0        LEA.L   $000012E0,A2
002320  10380904        MOVE.B  $00000904,D0
002324  42320000        CLR.B   0(A2,D0.W)
002328  51C8FFFA        DBF.L   D0,$002324
00232C  3006            MOVE.W  D6,D0
00232E  04400001        SUB.W   #1,D0
002332  343C0081        MOVE.W  #129,D2
002336  4244            CLR.W   D4
```

```
002338    3600            MOVE.W  D0,D3
00233A    96350000        SUB.B   0(A5,D0.W),D3
00233E    15830000        MOVE.B  D3,0(A2,D0.W)
002342    B602            CMP.B   D2,D3
002344    6F0C            BLE.S   $002352
002346    0C3300FF0000    CMP.B   #255,0(A3,D0.W)
00234C    6704            BEQ.S   $002352
00234E    1403            MOVE.B  D3,D2
002350    3800            MOVE.W  D0,D4
002352    51C8FFE4        DBF.L   D0,$002338
002356    3604            MOVE.W  D4,D3
002358    3006            MOVE.W  D6,D0
00235A    C0FC0021        MULU.W  #33,D0
00235E    41F40000        LEA.L   0(A4,D0.W),A0
002362    4240            CLR.W   D0
002364    10380904        MOVE.B  $00000904,D0
002368    15BC00FF0025    MOVE.B  #255,37(A2,D0.W)
00236E    11BC00FF0021    MOVE.B  #255,33(A0,D0.W)
002374    51C8FFF2        DBF.L   D0,$002368
002378    4240            CLR.W   D0
00237A    4244            CLR.W   D4
00237C    0C3300FF6000    CMP.B   #255,0(A3,D6.W)
002382    661A            BNE.S   $00239E
002384    18380904        MOVE.B  $00000904,D4
002388    11B040004021    MOVE.B  0(A0,D4.W),33(A0,D4.W)
00238E    51CCFFF8        DBF.L   D4,$002388
002392    4244            CLR.W   D4
002394    1BB560006001    MOVE.B  0(A5,D6.W),1(A5,D6.W)
00239A    60000078        BRA.L   $002414
00239E    10333000        MOVE.B  0(A3,D3.W),D0
0023A2    4400            NEG.B   D0
0023A4    D0380901        ADD.B   $00000901,D0
0023A8    4241            CLR.W   D1
0023AA    12336000        MOVE.B  0(A3,D6.W),D1
0023AE    61004C88        BSR.L   $007038
0023B2    3E00            MOVE.W  D0,D7
0023B4    3203            MOVE.W  D3,D1
0023B6    C2FC0021        MULU.W  #33,D1
0023BA    43F41000        LEA.L   0(A4,D1.W),A1
0023BE    3406            MOVE.W  D6,D2
0023C0    9443            SUB.W   D3,D2
0023C2    45F22025        LEA.L   37(A2,D2.W),A2
0023C6    18353000        MOVE.B  0(A5,D3.W),D4
0023CA    4241            CLR.W   D1
0023CC    12314000        MOVE.B  0(A1,D4.W),D1
0023D0    3007            MOVE.W  D7,D0
0023D2    61004C64        BSR.L   $007038
0023D6    15804000        MOVE.B  D0,0(A2,D4.W)
0023DA    51CCFFF0        DBF.L   D4,$0023CC
0023DE    4244            CLR.W   D4
0023E0    D4353000        ADD.B   0(A5,D3.W),D2
0023E4    B4356000        CMP.B   0(A5,D6.W),D2
0023E8    6D06            BLT.S   $0023F0
0023EA    18826001        MOVE.B  D2,1(A5,D6.W)
0023EE    6006            BRA.S   $0023F6
```

```
0023F0    1BB560006001    MOVE.B  0(A5,D6.W),1(A5,D6.W)
0023F6    45F812E0        LEA.L   $000012E0,A2
0023FA    18356001        MOVE.B  1(A5,D6.W),D4
0023FE    10304000        MOVE.B  0(A0,D4.W),D0
002402    12324025        MOVE.B  37(A2,D4.W),D1
002406    61004BF8        BSR.L   $007000
00240A    11804021        MOVE.B  D0,33(A0,D4.W)
00240E    51CCFFEE        DBF.L   D4,$0023FE
002412    4244            CLR.W   D4
002414    18380BB8        MOVE.B  $00000BB8,D4
002418    E24C            LSR.W   #1,D4
00241A    5244            ADDQ.W  #1,D4
00241C    4242            CLR.W   D2
00241E    14356001        MOVE.B  1(A5,D6.W),D2
002422    D4380905        ADD.B   $00000905,D2
002426    9444            SUB.W   D4,D2
002428    BC02            CMP.B   D2,D6
00242A    6E46            BGT.S   $002472
00242C    06460001        ADD.W   #1,D6
002430    4245            CLR.W   D5
002432    4242            CLR.W   D2
002434    4243            CLR.W   D3
002436    1A380BB8        MOVE.B  $00000BB8,D5
00243A    DA46            ADD.W   D6,D5
00243C    14365000        MOVE.B  0(A6,D5.W),D2
002440    16356000        MOVE.B  0(A5,D6.W),D3
002444    4A03            TST.B   D3
002446    6720            BEQ.S   $002468
002448    04030001        SUB.B   #1,D3
00244C    10303022        MOVE.B  34(A0,D3.W),D0
002450    3E05            MOVE.W  D5,D7
002452    9E43            SUB.W   D3,D7
002454    123670FF        MOVE.B  -1(A6,D7.W),D1
002458    61004BDE        BSR.L   $007038
00245C    3202            MOVE.W  D2,D1
00245E    61004BA0        BSR.L   $007000
002462    3400            MOVE.W  D0,D2
002464    51CBFFE6        DBF.L   D3,$00244C
002468    4243            CLR.W   D3
00246A    17826000        MOVE.B  D2,0(A3,D6.W)
00246E    6000FEBC        BRA.L   $00232C
002472    13FC001000010021  MOVE.B  #16,$00010021
00247A    11F90001002F1409  MOVE.B  $0001002F,$00001409
002482    11F900010031140A  MOVE.B  $00010031,$0000140A
00248A    11F900010033140B  MOVE.B  $00010033,$0000140B
002492    13FC00FF00010027  MOVE.B  #255,$00010027
00249A    13FC00FF00010029  MOVE.B  #255,$00010029
0024A2    13FC00FF0001002B  MOVE.B  #255,$0001002B
0024AA    13FC001100010021  MOVE.B  #17,$00010021
0024B2    45F81189        LEA.L   $00001189,A2
0024B6    224D            MOVE.L  A5,A1
0024B8    4245            CLR.W   D5
0024BA    4A356001        TST.B   1(A5,D6.W)
0024BE    6748            BEQ.S   $002508
```

```
0024C0    41E80021    LEA.L   33(A0),A0
0024C4    4242        CLR.W   D2
0024C6    14380900    MOVE.B  $00000900,D2
0024CA    4243        CLR.W   D3
0024CC    3E02        MOVE.W  D2,D7
0024CE    4407        NEG.B   D7
0024D0    DE380901    ADD.B   $00000901,D7
0024D4    4244        CLR.W   D4
0024D6    18316001    MOVE.B  1(A1,D6.W),D4
0024DA    3007        MOVE.W  D7,D0
0024DC    1204        MOVE.B  D4,D1
0024DE    61004BBC    BSR.L   $007090
0024E2    12304000    MOVE.B  0(A0,D4.W),D1
0024E6    61004B50    BSR.L   $007038
0024EA    1203        MOVE.B  D3,D1
0024EC    61004B12    BSR.L   $007000
0024F0    1600        MOVE.B  D0,D3
0024F2    51CCFFE6    DBF.L   D4,$0024DA
0024F6    4A03        TST.B   D3
0024F8    6608        BNE.S   $002502
0024FA    06050001    ADD.B   #1,D5
0024FE    15825000    MOVE.B  D2,0(A2,D5.W)
002502    51CAFFC6    DBF.L   D2,$0024CA
002506    1485        MOVE.B  D5,(A2)
002508    3E05        MOVE.W  D5,D7
00250A    DE380BB8    ADD.B   $00000BB8,D7
00250E    BA316001    CMP.B   1(A1,D6.W),D5
002512    6600017E    BNE.L   $002692
002516    43F80BB8    LEA.L   $00000BB8,A1
00251A    4A11        TST.B   (A1)
00251C    6728        BEQ.S   $002546
00251E    4A05        TST.B   D5
002520    6724        BEQ.S   $002546
002522    323C0001    MOVE.W  #1,D1
002526    4242        CLR.W   D2
002528    1411        MOVE.B  (A1),D2
00252A    4243        CLR.W   D3
00252C    1612        MOVE.B  (A2),D3
00252E    1C323000    MOVE.B  0(A2,D3.W),D6
002532    BC312000    CMP.B   0(A1,D2.W),D6
002536    6700015A    BEQ.L   $002692
00253A    B601        CMP.B   D1,D3
00253C    57CBFFF0    DBEQ.L  D3,$00252E
002540    B401        CMP.B   D1,D2
002542    57CAFFE6    DBEQ.L  D2,$00252A
002546    06160001    ADD.B   #1,(A6)
00254A    224E        MOVE.L  A6,A1
00254C    1C380904    MOVE.B  $00000904,D6
002550    4DF811EA    LEA.L   $000011EA,A6
002554    61004B06    BSR.L   $00705C
00255B    4244        CLR.W   D4
00255A    18380904    MOVE.B  $00000904,D4
00255E    0C3600FF4000 CMP.B  #255,0(A6,D4.W)
002564    56CCFFF8    DBNE.L  D4,$00255E
002568    B807        CMP.B   D7,D4
```

```
00256A    6E000126          BGT.L   $002692
00256E    BE380904          CMP.B   $00000904.D7
002572    6E00011E          BGT.L   $002692
002576    43F80BB8          LEA.L   $00000BB8,A1
00257A    4242              CLR.W   D2
00257C    1411              MOVE.B  (A1).D2
00257E    4A02              TST.B   D2
002580    6714              BEQ.S   $002596
002582    04020001          SUB.B   #1.D2
002586    3007              MOVE.W  D7,D0
002588    15B120010000      MOVE.B  1(A1.D2.W),0(A2.D0.W)
00258E    04400001          SUB.W   #1,D0
002592    51CAFFF4          DBF.L   D2,$002588
002596    43F8122B          LEA.L   $0000122B.A1
00259A    3C07              MOVE.W  D7.D6
00259C    04460001          SUB.W   #1.D6
0025A0    3406              MOVE.W  D6.D2
0025A2    3E3C0001          MOVE.W  #1.D7
0025A6    1A16              MOVE.B  (A6).D5
0025A9    4243              CLR.W   D3
0025AA    1604              MOVE.B  D4.D3
0025AC    4A03              TST.B   D3
0025AE    6726              BEQ.S   $0025D6
0025B0    10322001          MOVE.B  1(A2.D2.W).D0
0025B4    4400              NEG.B   D0
0025B6    D0380901          ADD.B   $00000901.D0
0025BA    1203              MOVE.B  D3,D1
0025BC    61004ADE          BSR.L   $00709C
0025C0    12363000          MOVE.B  0(A6,D3.W),D1
0025C4    61004A72          BSR.L   $007038
0025C8    1205              MOVE.B  D5,D1
0025CA    61004A34          BSR.L   $007000
0025CE    1A00              MOVE.B  D0.D5
0025D0    B607              CMP.B   D7.D3
0025D2    57CBFFDC          DBEQ.L  D3.$0025B0
0025D6    10322001          MOVE.B  1(A2,D2.W).D0
0025DA    1206              MOVE.B  D6.D1
0025DC    61004ABE          BSR.L   $00709C
0025E0    1205              MOVE.B  D5.D1
0025E2    61004A54          BSR.L   $007038
0025E6    11C012DD          MOVE.B  D0.$000012DD
0025EA    4205              CLR.B   D5
0025EC    1005              MOVE.B  D5.D0
0025EE    3606              MOVE.W  D6,D3
0025F0    B403              CMP.B   D3.D2
0025F2    6714              BEQ.S   $002608
0025F4    10322001          MOVE.B  1(A2.D2.W).D0
0025F8    12323001          MOVE.B  1(A2.D3.W).D1
0025FC    61004A02          BSR.L   $007000
002600    1205              MOVE.B  D5.D1
002602    61004A34          BSR.L   $007038
002606    1A00              MOVE.B  D0.D5
002608    51CBFFE6          DBF.L   D3,$0025F0
00260C    4400              NEG.B   D0
00260E    D0380901          ADD.B   $00000901,D0
```

```
002612    123812DD          MOVE.B  $000012DD.D1
002616    61004A20          BSR.L   $007038
00261A    13802000          MOVE.B  D0.0(A1,D2.W)
00261E    51CAFF86          DBF.L   D2,$0025A6
002622    47F80AF9          LEA.L   $00000AF9.A3
002626    49F81268          LEA.L   $00001268.A4
00262A    4BF80A7A          LEA.L   $00000A7A,A5
00262E    4242              CLR.W   D2
002630    14380900          MOVE.B  $00000900,D2
002634    4245              CLR.W   D5
002636    19B320002000      MOVE.B  0(A3,D2.W),0(A4,D2.W)
00263C    4243              CLR.W   D3
00263E    1606              MOVE.B  D6.D3
002640    B4323001          CMP.B   1(A2,D3.W),D2
002644    6610              BNE.S   $002656
002646    10332000          MOVE.B  0(A3,D2.W),D0
00264A    12313000          MOVE.B  0(A1,D3.W),D1
00264E    610049B0          BSR.L   $007000
002652    19802000          MOVE.B  D0.0(A4,D2.W)
002656    51CBFFE8          DBF.L   D3,$002640
00265A    16352000          MOVE.B  0(A5,D2.W),D3
00265E    B6342000          CMP.B   0(A4,D2.W),D3
002662    6702              BEQ.S   $002666
002664    5245              ADDQ.W  #1,D5
002666    51CAFFCE          DBF.L   D2,$002636
00266A    4A05              TST.B   D5
00266C    6648              BNE.S   $0026B6
00266E    13FC001000010021  MOVE.B  #16.$00010021
002676    11F90001002F140C  MOVE.B  $0001002F,$0000140C
00267E    11F900010031140D  MOVE.B  $00010031.$0000140D
002686    11F900010033140E  MOVE.B  $00010033.$0000140E
00268E    60000170          BRA.L   $002800
002692    13FC001000010021  MOVE.B  #16.$00010021
00269A    11F90001002F140C  MOVE.B  $0001002F.$0000140C
0026A2    11F900010031140D  MOVE.B  $00010031.$0000140D
0026AA    11F900010033140E  MOVE.B  $00010033.$0000140E
0026B2    6000015E          BRA.L   $002812
0026B6    13FC001000010021  MOVE.B  #16.$00010021
0026BE    11F90001002F140C  MOVE.B  $0001002F,$0000140C
0026C6    11F900010031140D  MOVE.B  $00010031.$0000140D
0026CE    11F900010033140E  MOVE.B  $00010033.$0000140E
0026D6    6000014C          BRA.L   $002824
0026DA    13FC001000010021  MOVE.B  #16.$00010021
0026E2    11F90001002F140C  MOVE.B  $0001002F.$0000140C
0026EA    11F900010031140D  MOVE.B  $00010031.$0000140D
0026F2    11F900010033140E  MOVE.B  $00010033.$0000140E
0026FA    6000013A          BRA.L   $002836
```

```
007000  48E74002        MOVEM.L D1/A6,-(A7)
007004  5300            ADDQ.B  #1,D0
007006  5201            ADDQ.B  #1,D1
007008  06400907        ADD.W   #2311,D0
00700C  06410907        ADD.W   #2311,D1
007010  2C40            MOVE.L  D0,A6
007012  1016            MOVE.B  (A6),D0
007014  2C41            MOVE.L  D1,A6
007016  1216            MOVE.B  (A6),D1
007018  B300            EOR.B   D1,D0
00701A  4DF80907        LEA.L   $00000907,A6
00701E  123CFFFF        MOVE.B  #-1,D1
007022  B01E            CMP.B   (A6)+,D0
007024  6704            BEQ.S   $00702A
007026  5201            ADDQ.B  #1,D1
007028  60F8            BRA.S   $007022
00702A  1001            MOVE.B  D1,D0
00702C  0280000000FF    AND.L   #255,D0
007032  4CDF4002        MOVEM.L (A7)+,D1/A6
007036  4E75            RTS
007038  2F01            MOVE.L  D1,-(A7)
00703A  0C0000FF        CMP.B   #255,D0
00703E  6718            BEQ.S   $007058
007040  0C0100FF        CMP.B   #255,D1
007044  6604            BNE.S   $00704A
007046  1001            MOVE.B  D1,D0
007048  600E            BRA.S   $007058
00704A  D041            ADD.W   D1,D0
00704C  12380901        MOVE.B  $00000901,D1
007050  B041            CMP.W   D1,D0
007052  6D04            BLT.S   $007058
007054  9041            SUB.W   D1,D0
007056  60F8            BRA.S   $007050
007058  221F            MOVE.L  (A7)+,D1
00705A  4E75            RTS
00705C  48E7E700        MOVEM.L D0-D2/D5-D7,-(A7)
007060  3E05            MOVE.W  D5,D7
007062  3406            MOVE.W  D6,D2
007064  DE46            ADD.W   D6,D7
007066  3807            MOVE.W  D7,D4
007068  1DBC00FF7000    MOVE.B  #255,0(A6,D7.W)
00706E  51CFFFF8        DBF.L   D7,$007068
007072  10305000        MOVE.B  0(A0,D5.W),D0
007076  12316000        MOVE.B  0(A1,D6.W),D1
00707A  61BC            BSR.S   $007038
00707C  3E05            MOVE.W  D5,D7
00707E  DE46            ADD.W   D6,D7
007080  12367000        MOVE.B  0(A6,D7.W),D1
007084  6100FF7A        BSR.L   $007000
007088  1D807000        MOVE.B  D0,0(A6,D7.W)
00708C  51CEFFE4        DBF.L   D6,$007072
007090  3C02            MOVE.W  D2,D6
007092  51CDFFDE        DBF.L   D5,$007072
007096  4CDF00E7        MOVEM.L (A7)+,D0-D2/D5-D7
00709A  4E75            RTS
```

```
0070?C    48A73000        MOVEM.W D2-D3,-(A7)
0070A0    0C0000FF        CMP.B   #255,D0
0070A4    6716            BEQ.S   $0070BC
0070A6    C3C1            MULU.W  D1,D0
0070A8    4242            CLR.W   D2
0070AA    4243            CLR.W   D3
0070AC    16380901        MOVE.B  $00000901,D3
0070B0    3403            MOVE.W  D3,D2
0070B2    5342            SUBQ.W  #1,D2
0070B4    B042            CMP.W   D2,D0
0070B6    6F0E            BLE.S   $0070C6
0070B8    9043            SUB.W   D3,D0
0070BA    60F8            BRA.S   $0070B4
0070BC    0240007F        AND.W   #127,D0
0070C0    4C9F000C        MOVEM.W (A7)+,D2-D3
0070C4    4E75            RTS
0070C6    0C40FFFF        CMP.W   #-1,D0
0070CA    6CF0            BGE.S   $0070BC
0070CC    D043            ADD.W   D3,D0
0070CE    60F6            BRA.S   $0070C6
```

B.3 SUBRUTINAS PARA MENSAJES ASCII

```
002800    4EF94000          LEA.L   $00004000.A5
002804    4DF94018          LEA.L   $00004018.A6
002808    1E3C00F3          MOVE.B  #243,D7
00280C    4E4E              TRAP    #14
00280E    60000034          BRA.L   $002844
002812    4BF84020          LEA.L   $00004020.A5
002816    4DF8403A          LEA.L   $0000403A.A6
00281A    1E3C00F3          MOVE.B  #243,D7
00281E    4E4E              TRAP    #14
002820    60000022          BRA.L   $002844
002824    4EF84040          LEA.L   $00004040.A5
002828    4DF84058          LEA.L   $00004058.A6
00282C    1E3C00F3          MOVE.B  #243,D7
002830    4E4E              TRAP    #14
002832    60000010          BRA.L   $002844
002836    4BF84060          LEA.L   $00004060.A5
00283A    4DF84071          LEA.L   $00004071,A6
00283E    1E3C00F3          MOVE.B  #243,D7
002842    4E4E              TRAP    #14
002844    4242              CLR.W   D2
002846    1E3C00E4          MOVE.B  #228,D7
00284A    4E4E              TRAP    #14
```

TABLA DE CARACTERES ASCII

```
004000    0A 44 45 43 4F 44 49 46  49 43 41 43 49 4F 4E 20  .DECODIFICACION
004010    45 58 49 54 4F 53 41 0A  FF FF FF FF FF FF FF FF  EXITOSA........
004020    0A 44 45 43 4F 44 49 46  49 43 41 43 49 4F 4E 20  .DECODIFICACION
004030    49 4D 50 4F 53 49 42 4C  45 0A FF FF FF FF FF FF  IMPOSIBLE.......
004040    0A 44 45 43 4F 44 49 46  49 43 41 43 49 4F 4E 20  .DECODIFICACION
004050    45 52 52 4F 4E 45 41 0A  FF FF FF FF FF FF FF FF  ERRONEA........
004060    0A 4E 4F 20 48 55 42 4F  20 45 52 52 4F 52 45 53  .NO HUBO ERRORES
004070    0A FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
```

APENDICE C

CODIGOS DE FIRE

APENDICE C

CODIGOS DE FIRE

C.1 INTRODUCCION. [9,14,27,31,40]. Los códigos de Fire constituyen un ejemplo de códigos cíclicos óptimos para la corrección y/o detección de paquetes únicos de errores (i.e. un paquete por palabra). Su polinomio generador tiene la forma:

$$g(x) = (x^c - 1)\, p(x) \tag{C.1}$$

donde $p(x)$ es un polinomio primitivo e irreductible sobre $CG(q)$, que tiene orden e, los grados de $g(x)$ y $p(x)$ son m y l respectivamente. Los polinomios (x^c-1) y $p(x)$ son primos entre sí y además c no es divisible por e.

La longitud del código está dada por el mínimo común múltiplo de c y e:

$$n = MCM\{c,e\} \tag{C.2}$$

Este tipo de códigos pueden corregir todos los paquetes únicos de errores de longitud b y al mismo tiempo detectar todos los paquetes de longitud $d \geq b$, tales que:

$$b + d - 1 \leq c \tag{C.3}$$

si se toma $d=b$, entonces:

$$c \geq 2b - 1 \tag{C.4}$$

La dimensión (número de símbolos de información) del código es:

$$k = n - c - l \tag{C.5}$$

con una redundancia dada por:

$$m = c + l \tag{C.6}$$

Para los códigos de Fire, el número de símbolos de redundancia es mayor o igual a 3b-1, entonces:

$$c + l \geq 3b - 1 \qquad (C.7)$$

en el caso en que d=b se tiene, a partir de (C.4) y (C.7):

$$l \geq b \qquad (C.8)$$

Por lo tanto, los parámetros del código son:

$$\begin{aligned} n &= MCM\ (2b-1,\ e) \\ k &= n-2b+1-l \end{aligned} \qquad (C.9)$$

donde: $l \geq b$ y $d = b$.

Regresando al polinomio generador del código, puede decirse que (x^c-1) es suficiente para determinar completamente al patrón de error con un paquete de longitud no mayor a b (véase C.4). La información adicional, requerida para determinar el lugar donde ocurre el paquete de símbolos en error, está dada por p(x).

C.2 CODIFICACION Y DECODIFICACION. La codificación de códigos de Fire es la misma que la que se emplea en general para códigos cíclicos, y que fué descrita en el capítulo 1.

La operación de decodificar una palabra recibida comienza con el cálculo del síndrome S(x) de la misma:

$$S(x) = r(x) \text{ módulo } g(x) \qquad (C.10)$$

el cuál es un polinomio de grado inferior a m, que es nulo si e(x) es nulo, es decir si no hubo errores.

Si se supone un error e(x) decodificable, es decir un paquete de errores de longitud $b \leq (n-k)/2$, entonces:

$$e(x) = x^j p_o(x) \text{ módulo } x^{n-1}, \quad j \in [0,1,\ldots,n-1] \tag{C.11}$$

donde $p_o(x)$ es el polinomio de error:

$$p_o(x) = p_o + p_1 x + \ldots + p_{b-1} x^{b-1} \tag{C.12}$$

con $p_o = p_{b-1} = 1$, y j es la posición del paquete.

Ahora, si se tiene que: $j = k + (n-k)/2$, el polinomio $e(x)$ queda:

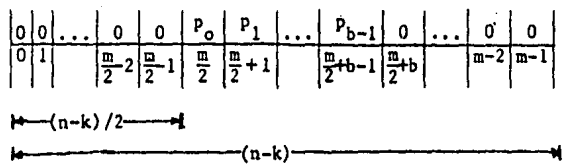
$$e(x) = x^{k+(n-k)/2} p_o(x) \text{ módulo } x^{n-1} \tag{C.13}$$

y representa un paquete de errores en la posición: $k + (n-k)/2$. El decodificador a continuación premultiplica al error por $x^{(n-k)}$ y obtiene el síndrome:

$$s(x) = x^{(n-k)/2} p_o(x) \text{ módulo } g(x) \tag{C.14}$$

de grado menor que $(n-k)$ y en el cuál $(n-k)/2$ términos son nulos.

La representación de este síndrome, en términos de un vector se esquematiza en la siguiente figura:



Debido a que, como se apuntó, los códigos de Fire son códigos cíclicos, el decodificador de Meggitt [31], estudiado en el capítulo 1, puede ser utilizado en la decodificación de paquetes de símbolos en error. Su circuito se muestra en la figura siguiente:

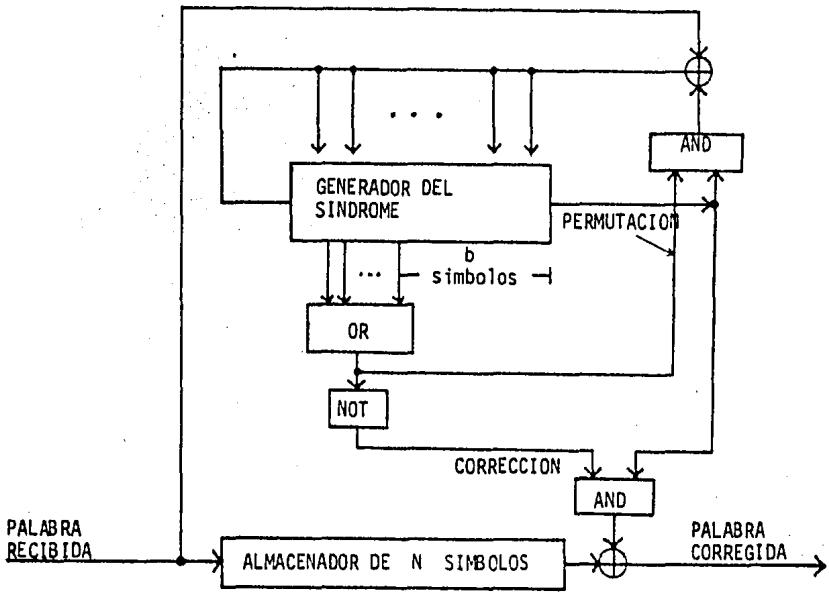


Figura C.1 Circuito decodificador Meggitt para códigos de Fire.

Suponiendo que ocurrió un paquete de errores de longitud menor o igual a b, el circuito calcula la versión recorrida cíclicamente del síndrome del patrón de error contenido en la palabra recibida (C.14). Si el paquete de errores se encuentra confinado en las b posiciones de mayor orden de la palabra recibida, entonces las b etapas del generador del síndrome a la derecha contienen al patrón de error y las n-k-b etapas de la izquierda contienen ceros.

La lógica combinacional del circuito debe detectar esta situación, haciendo que su salida sea cero para abrir el circuito de realimentación del generador del síndrome y cerrar la trayectoria de corrección. Al recorrer el contenido del generador del síndrome y del almacenador simultáneamente, el síndrome -el paquete de errores- es sumado módulo 2 a la palabra recibida y la decodificación se completa.

Como ejemplo del funcionamiento de los códigos de Fire, se considera la

corrección de paquetes de 3 bits en error. Asi b=3, c=2b-1=5 y

$$p(x) = 1 + x + x^3$$

con $e = 2^3 - 1 = 7$ y $l = 3$. El polinomio generador del código es entonces:

$$g(x) = (x^5-1)\ (x^3 + x + 1) = x^8 + x^6 + x^5 + x^3 + x + 1$$

además

$n = MCM\ (5,7) = 35$

$k = n-c-l = 35 - 5 - 3 = 27$

por lo tanto, se tiene un Código de Fire (35,27) corrector de paquetes de hasta 3 bits en error.

Supóngase el mensaje 1001000100010010100110010001, de polinomio asociado:

$$i(x) = 1 + x^3 + x^6 + x^7 + x^{10} + x^{12} + x^{15} + x^{19} + x^{23} + x^{26}$$

y el polinomio asociado a la palabra de código es calculado según (1.23) y (1.24), obteniéndose:

$$C(x) = 1+x+x^3+x^5+x^6+x^7+x^8+x^{11}+x^{14}+x^{15}+x^{18}+x^{20}+x^{23}+x^{27}+x^{31}+x^{34}$$

El circuito que realiza la operación anterior es:

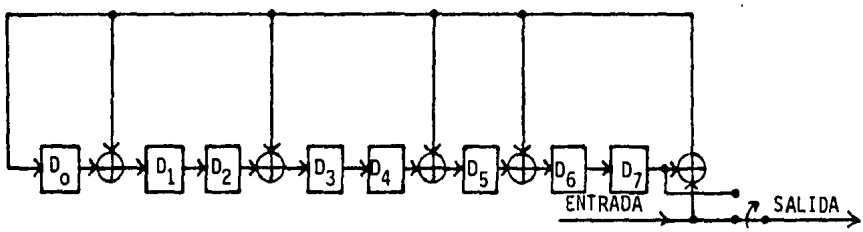


Figura C.2. Circuito codificador, por división por g(x), para el Código de Fire (35,27) corrector de paquetes de hasta 3 bits en error.

La decodificación se efectúa mediante el circuito de la figura C.3.

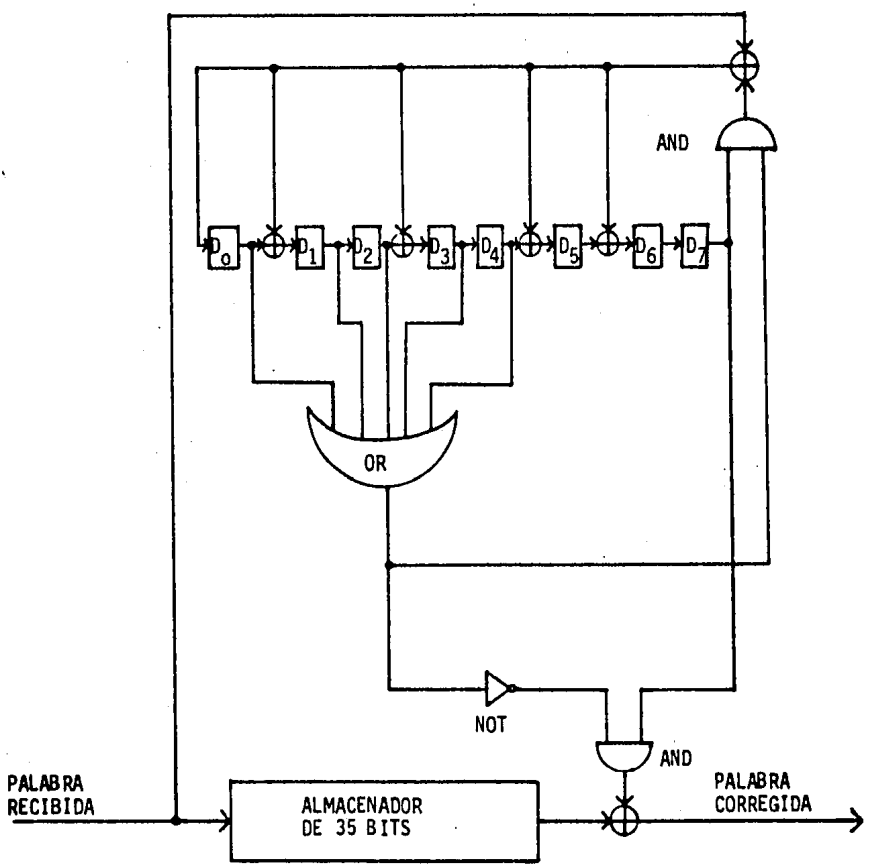


Figura C.3. Circuito decodificador Meggitt para el código de Fire (35,27) corrector de paquetes de hasta 3 bits.

Para simular el ejemplo se ejecutó el programa que aparece en el apéndice D.3 y cuyos resultados se presentan a continuación.

```
.RUN DK1:FIRE

  PROGRAMA SIMULADOR DE UN CODIGO DE FIRE

  GRADO DEL POLINOMIO GENERADOR, M=8

  POLINOMIO GENERADOR, G(X)?

110101101

   1+ 1*X** 1+ 0*X** 2+ 1*X** 3+ 0*X** 4+ 1*X** 5+ 1*X** 6+ 0*X** 7+ 1*X** 8+

  NO. DE BITS NO UTILIZADOS, IACOR= 0

-------------------------------------

  CODIGO DE FIRE ( 35, 27)
  CORRECTOR DE PAQUETES DE   3 BITS

-------------------------------------

  INFORMACION, I(X)?

10010011001010010001000 1001

   1+ 0*X** 1+ 0*X** 2+ 1*X** 3+ 0*X** 4+ 0*X** 5+ 1*X** 6+ 1*X** 7+ 0*X** 8+ 0*X** 9+ 1*X**10
 0*X**11+ 1*X**12+ 0*X**13+ 0*X**14+ 1*X**15+ 0*X**16+ 0*X**17+ 0*X**18+ 1*X**19+ 0*X**20
 0*X**21+ 0*X**22+ 1*X**23+ 0*X**24+ 0*X**25+ 1*X**26+

  CODIGO, C(X)?


11010111100100110010100100010001001

   1+ 1*X** 1+ 0*X** 2+ 1*X** 3+ 0*X** 4+ 1*X** 5+ 1*X** 6+ 1*X** 7+ 1*X** 8+ 0*X** 9+ 0*X**10
 1*X**11+ 0*X**12+ 0*X**13+ 1*X**14+ 1*X**15+ 0*X**16+ 0*X**17+ 1*X**18+ 0*X**19+ 1*X**20
 0*X**21+ 0*X**22+ 1*X**23+ 0*X**24+ 0*X**25+ 0*X**26+ 1*X**27+ 0*X**28+ 0*X**29+ 0*X**30
 1*X**31+ 0*X**32+ 0*X**33+ 1*X**34+

  # DE OPERACIONES EN LA CODIFICACION =      2.6300000000E+03
```

Listado C.1 Resultados de la simulación del código de Fire (35,27), corrector de paquetes de 3 bits en error.

```
  ERRORES, E(X)?

00000000000011100000000000000000000

  PALABRA RECIBIDA, RECU(X):


11010111100111010010100100010001001

   1+ 1*X** 1+ 0*X** 2+ 1*X** 3+ 0*X** 4+ 1*X** 5+ 1*X** 6+ 1*X** 7+ 1*X** 8+ 0*X** 9+ 0*X**10
 1*X**11+ 1*X**12+ 1*X**13+ 0*X**14+ 1*X**15+ 0*X**16+ 0*X**17+ 1*X**18+ 0*X**19+ 1*X**20
 0*X**21+ 0*X**22+ 1*X**23+ 0*X**24+ 0*X**25+ 0*X**26+ 1*X**27+ 0*X**28+ 0*X**29+ 0*X**30
 1*X**31+ 0*X**32+ 0*X**33+ 1*X**34+

  SINDROME, S(X):


0010011

   0+ 0*X** 1+ 1*X** 2+ 0*X** 3+ 0*X** 4+ 1*X** 5+ 1*X** 6+

  CONTENIDO DEL GENERADOR DEL SINDROME:

  00100110
  00010011
  11011111
  10111001
  10001010
  01000101
  11110100
  01111010
  00111101
  11001000
  01100100
  00110010
  00011001
  11011010
  01101101
  11100000
  01110000
```

Listado C.1 Resultados de la simulación del código de Fire (35,27), corrector de paquetes de 3 bits en error (continuación).

```
00111000
00011100
00001110
00000111

DECODIFICACION EXITOSA.

1101011111001001100101001000100010001

  1+ 1*X** 1+ 0*X** 2+ 1*X** 3+ 0*X** 4+ 1*X** 5+ 1*X** 6+ 1*X** 7+ 1*X** 8+ 0*X** 9+ 0*X**10
1*X**11+ 0*X**12+ 0*X**13+ 1*X**14+ 1*X**15+ 0*X**16+ 0*X**17+ 1*X**18+ 0*X**19+ 1*X**20
0*X**21+ 0*X**22+ 1*X**23+ 0*X**24+ 0*X**25+ 0*X**26+ 1*X**27+ 0*X**28+ 0*X**29+ 0*X**30
1*X**31+ 0*X**32+ 0*X**33+ 1*X**34+

 # DE OPERACIONES EN LA DECODIFICACION =     2.2508000000E+04
```

Listado C.1 Resultados de la simulación del código de Fire (35,27), corrector de paquetes de 3 bits en error (continuación).

APENDICE D

DESCRIPCION DE PROGRAMAS Y SUBRUTINAS PARA LA SIMULACION DE CODIGOS CICLICOS

APENDICE D

DESCRIPCION DE PROGRAMAS Y SUBRUTINAS PARA LA SIMULACION DE CODIGOS CICLICOS

D.1 INTRODUCCION.

Con el objeto de apoyar el trabajo teórico sobre códigos cíclicos, se elaboró un paquete de programas y subrutinas que se escribieron en lenguaje FORTRAN. Dicho paquete se implementó en la minicomputadora PDP 11/40 del laboratorio CAD de la División de Estudios de Posgrado de la Facultad de Ingeniería de la UNAM.

Mediante este paquete es posible simular la codificación y la decodificación de información haciendo uso de los algoritmos, o simulando los circuitos, correspondientes a las familias de códigos cíclicos que se analizan en este trabajo, es decir: Hamming, Fire, BCH y Reed-Solomon.

Los programas permiten analizar el comportamiento de una amplia variedad de códigos ante diferentes tipos de patrón de error determinados por el usuario, así como evaluar la complejidad de las operaciones realizadas por los algoritmos, o circuitos, haciendo una comparación del número de ciclos que tardaría un microprocesador de propósito general, MC68000 (ver capítulo 6), al efectuar dichas operaciones.

D.2 SIMULACION DE CODIGOS DE HAMMING.

Para simular el funcionamiento de los circuitos de codificación y decodificación empleando códigos de Hamming, se realizaron dos programas correspondientes a: (i) los códigos de Hamming (i.e., para los que $n=2^m-1-i_{acor}$) y a (ii) los códigos extendidos de Hamming ($n=2^m-i_{acor}$).

En ambos casos, el usuario proporciona los datos correspondientes al polinomio generador del código $g(x)$, y su grado m. Además, se especifica el orden de acortamiento del código i_{acor}, con lo cuál es posible simular una gran cantidad de códigos de Hamming, con el mismo polinomio (o matriz) generador.

Además de lo anterior, cuando se ejecuta el programa, el usuario proporciona los bits de información, que son leídos como un vector de números enteros. Una vez codificada la información, la palabra de código es desplegada en la terminal y el usuario introduce la palabra o patrón de error, simulando los errores introducidos por el canal. El resultado de la decodificación, que puede ser: exitosa, errónea o imposible, y la palabra decodificada, son desplegados en la terminal al finalizar la ejecución del programa. En las figuras D.1 y D.2 se muestran los diagramas de flujo correspondientes a los programas para la simulación de la familia de códigos de Hamming.

D.3 SIMULACION DE CODIGOS DE FIRE.

Para la familia de códigos de Fire, la simulación se logró mediante un programa cuyos datos iniciales son el polinomio generador del código $g(x)$, su grado m y el orden de acortamiento i_{acor}. La información es leída desde la terminal y codificada en una palabra de código que se despliega. Entonces, desde la terminal se lee el patrón de error y el programa procede a efectuar la decodificación, al cabo de la cual se indican el resultado y la palabra decodificada. La figura D.3, corresponde al diagrama de flujo del programa para simular códigos de Fire.

Durante la decodificación, al contenido del generador de síndrome se indica en cada paso del proceso, simulando el circuito decodificador Meggitt.

D.4 SUBRUTINAS SOBRE CG(2).

Para lograr la simulación de las familias de códigos cíclicos que trabajan con elementos binarios, fué necesario elaborar algunas subrutinas que permitieran efectuar operaciones entre dichos elementos. Las subrutinas pueden ser divididas en dos clases: (i) para operaciones (suma y multiplicación) entre elementos binarios, y (ii) para operaciones (multiplicación y división) entre polinomios cuyos coeficientes pertenecen a CG(2).

En el primer caso, se implementaron dos funciones enteras correspondientes a la multiplicación (operación lógica AND) y la suma (operación lógica XOR) entre elementos binarios. Estas funciones tienen la ventaja de facili

DIAGRAMA DE FLUJO.
SIMULACION DE CODIGOS DE HAMMING.

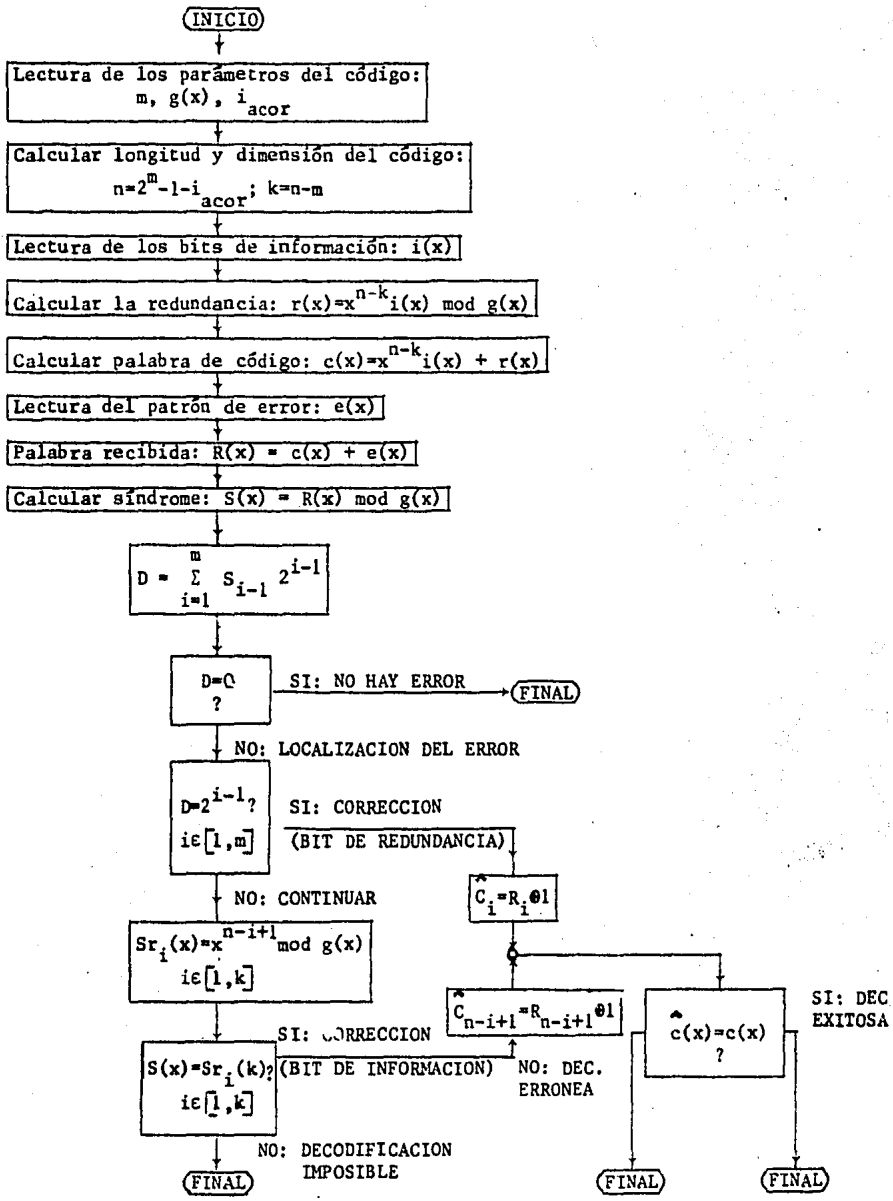


FIGURA D.1. DIAGRAMA DE FLUJO DEL PROGRAMA PARA SIMULAR CODIGOS DE HAMMING.

DIAGRAMA DE FLUJO
SIMULACION DE CODIGOS EXTENDIDOS DE HAMMING.

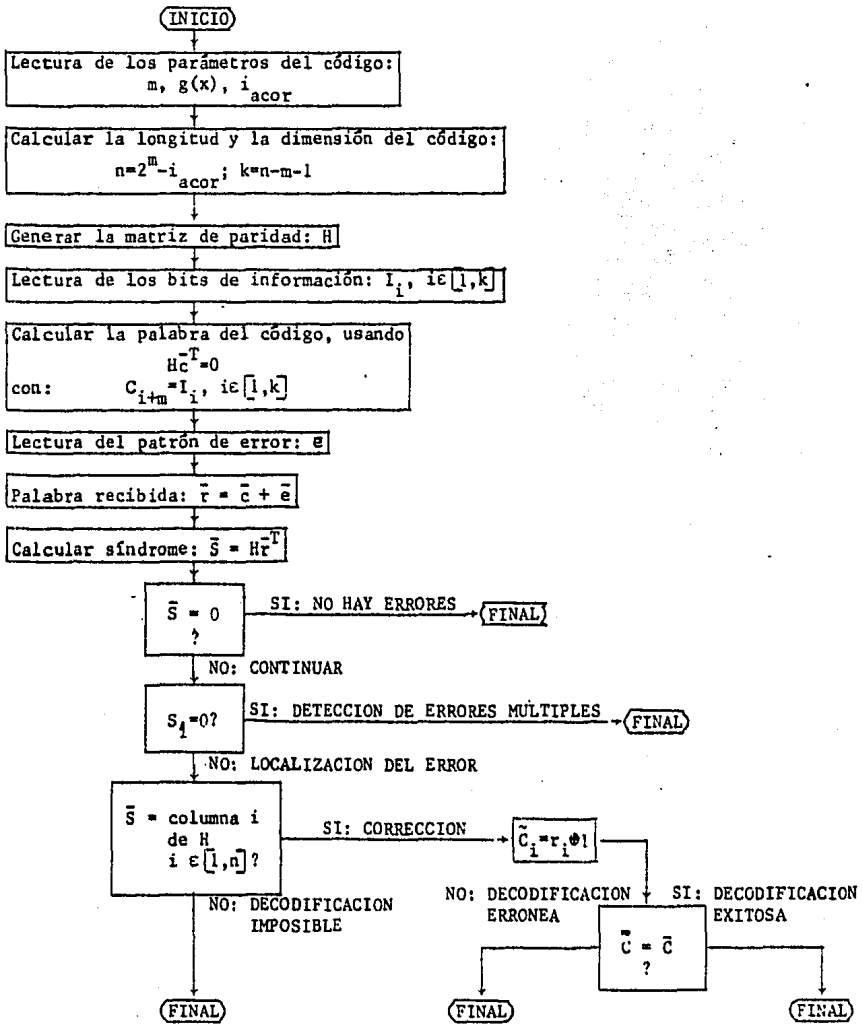


FIGURA D.2. DIAGRAMA DE FLUJO DEL PROGRAMA PARA SIMULAR CODIGOS EXTENDIDOS DE HAMMING

DIAGRAMA DE FLUJO
SIMULACION DE CODIGOS DE FIRE

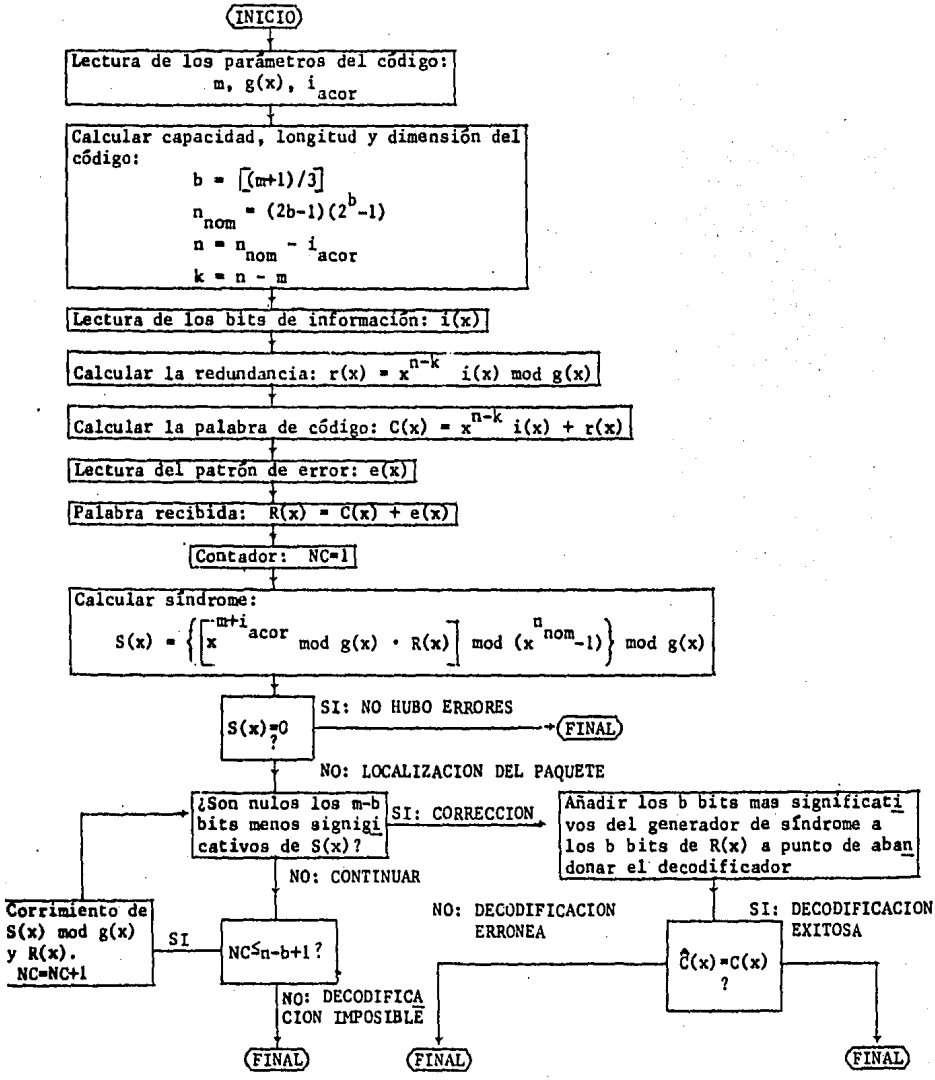


FIGURA D.3 DIAGRAMA DE FLUJO DEL PROGRAMA SIMULADOR DE CODIGOS DE FIRE.

tar la comprensión de los algoritmos que simulan la operación de los circuitos electrónicos empleados en la codificación y decodificación de códigos cíclicos binarios.

Con respecto a la segunda clase de subrutinas, se realizaron dos subrutinas. Una de ellas simula la operación de un circuito multiplicador de polinomios sobre CG(2) y la otra simula un circuito divisor de dos polinomios con elementos binarios como coeficientes (usando el algoritmo euclidiano de la división), dando como salida el residuo de la división. Esta última subrutina es la que tiene mayor importancia en la codificación y decodificación mediante códigos cíclicos binarios, su descripción mediante un diagrama de flujo se presenta en la figura D.4. Como entradas a la subrutina se tienen dos vectores A y B, con NA y NB coeficientes respectivamente. La salida es el vector R con NR elementos, que corresponde al residuo del algoritmo euclidiano de la división:

$$\frac{A(x)}{B(x)} = Q(x) + \frac{R(x)}{B(x)}$$

es decir: $R(x) = A(x) \bmod B(x)$.

D.5 SIMULACION DE CODIGOS DE BCH.

En el caso de los códigos cíclicos binarios de BCH, el programa que los simula tiene como datos iniciales los siguientes: polinomio generador $g(x)$ y orden multiplicativo m del campo de Galois $CG(2^m)$, capacidad de corrección del código t; longitud del código n, dimensión nominal del código k_{na} y el polinomio generador del código $G(x)$. Una vez que el usuario ha introducido estos datos, se calculan los parámetros restantes del código: longitud nominal n; orden de acortamiento i_{acor} y dimensión del código k. En seguida se procede a la construcción de tablas, de logaritmos y de suma, de elementos del $CG(2^m)$, esta etapa se logra mediante las subrutinas descritas en el párrafo D.6. El usuario introduce entonces los bits de información, el programa los codifica y despliega en la terminal la palabra de código correspondiente.

El ruido en el canal se simula mediante la lectura del patrón de error que corresponda y el programa inicia entonces la decodificación de la palabra recibida. Al terminar la decodificación, el programa indica el resulta

DIAGRAMA DE FLUJO
DIVISION DE POLINOMIOS SOBRE CG(2)

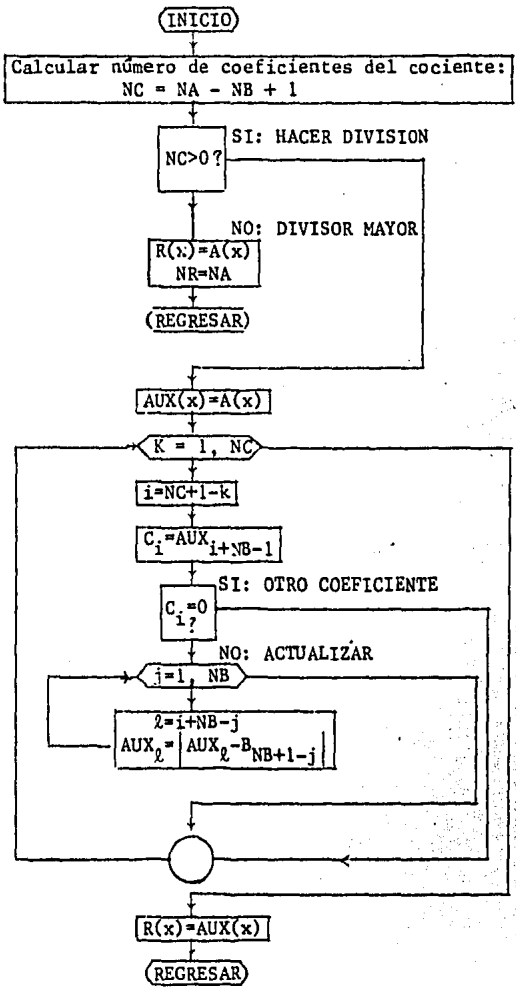


FIGURA D.4 DIAGRAMA DE FLUJO DE LA SUBRUTINA PARA REALIZAR EL ALGORITMO EUCLIDIANO DE LA DIVISION, SOBRE CG(2).

do de la misma (sin errores, imposible, errónea o exitosa) y la palabra decodificada. En la figura D.5 puede apreciarse el diagrama de flujo del programa descrito en este párrafo.

Como se señaló en el capítulo 3, en la decodificación se emplea el algoritmo de Berlekamp-Massey para determinar el polinomio localizador de errores $\sigma(x)$. La figura D.6 ilustra la implementación del algoritmo en la minicomputadora.

D.6 SUBRUTINAS SOBRE $CG(2^m)$.

Para los programas que efectúan operaciones con elementos de campos finitos $CG(2^m)$, se elaboraron las subrutinas que permitieran: (i) generar la tabla de elementos (logaritmos) del campo, (ii) generar la tabla de suma entre elementos del campo y (iii) multiplicar y elevar a una potencia entera los elementos pertenecientes a $CG(2^m)$.

La generación de la tabla de elementos se logra usando el polinomio generador $g(x)$ y el orden multiplicativo del campo, m. La tabla consiste en una matriz de 2^m renglones, en donde cada renglón corresponde a un elemento del campo, las m columnas están asociadas a los m bits de cada elemento. Por ejemplo, para el caso en el que $g(x) = 1 + x + x^3$ con $m=3$, la matriz estará dada por:

$$M = \begin{bmatrix} 0\ 0\ 0 \\ 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \\ 1\ 1\ 0 \\ 0\ 1\ 1 \\ 1\ 1\ 1 \\ 1\ 0\ 1 \end{bmatrix}$$

Renglón	Logaritmo
1	$-\infty$
2	0
3	1
4	2
5	3
6	4
7	5
8	6

Como se observa, los renglones de la matriz y los logaritmos de los elementos no nulos se relacionan mediante la expresión $r=\ell+2$, siendo r = número de renglón y ℓ = logaritmo del elemento correspondiente al renglón r. Esta característica se emplea al efectuar las operaciones entre los elemen

DIAGRAMA DE FLUJO
SIMULACION DE CODIGOS B.C.H.

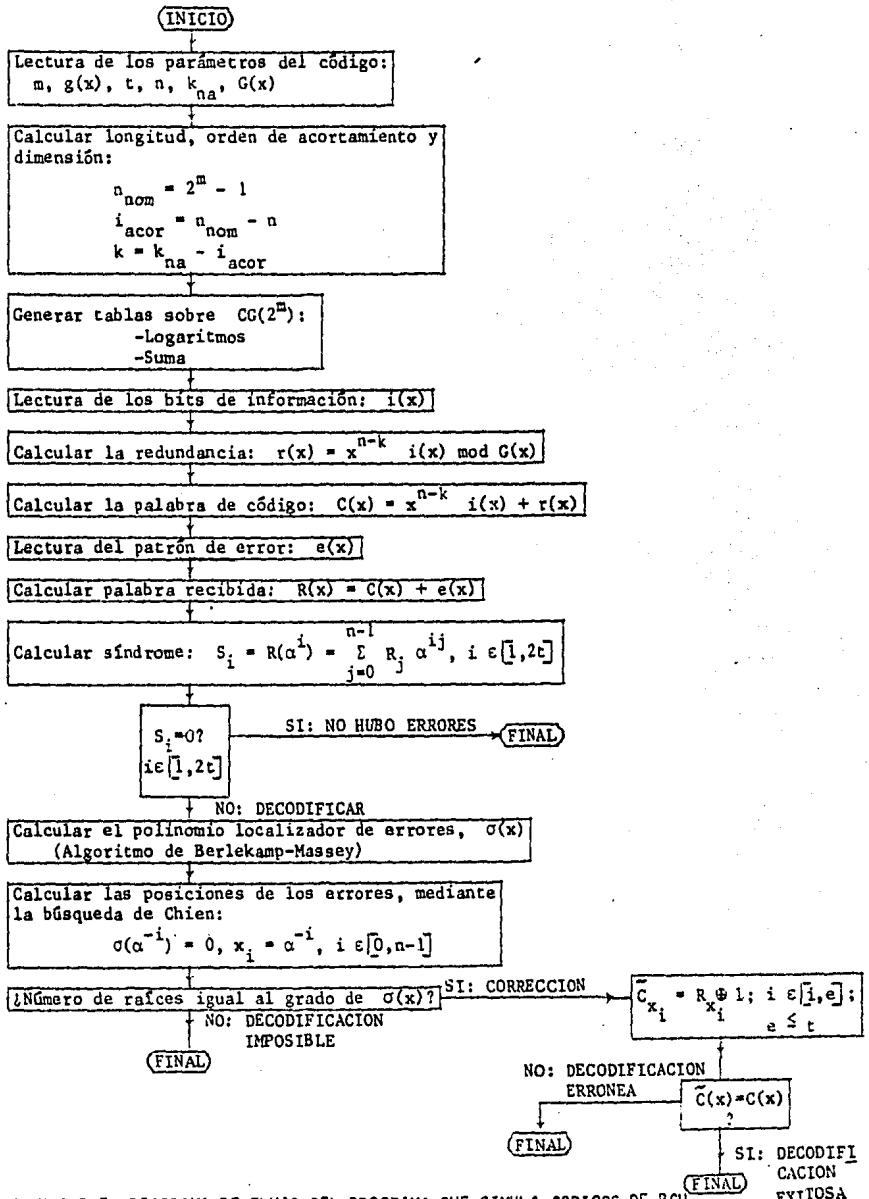


FIGURA D.5 DIAGRAMA DE FLUJO DEL PROGRAMA QUE SIMULA CODIGOS DE BCH.

DIAGRAMA DE FLUJO
ALGORITMO DE BERLEKAMP-MASSEY
CASO BINARIO

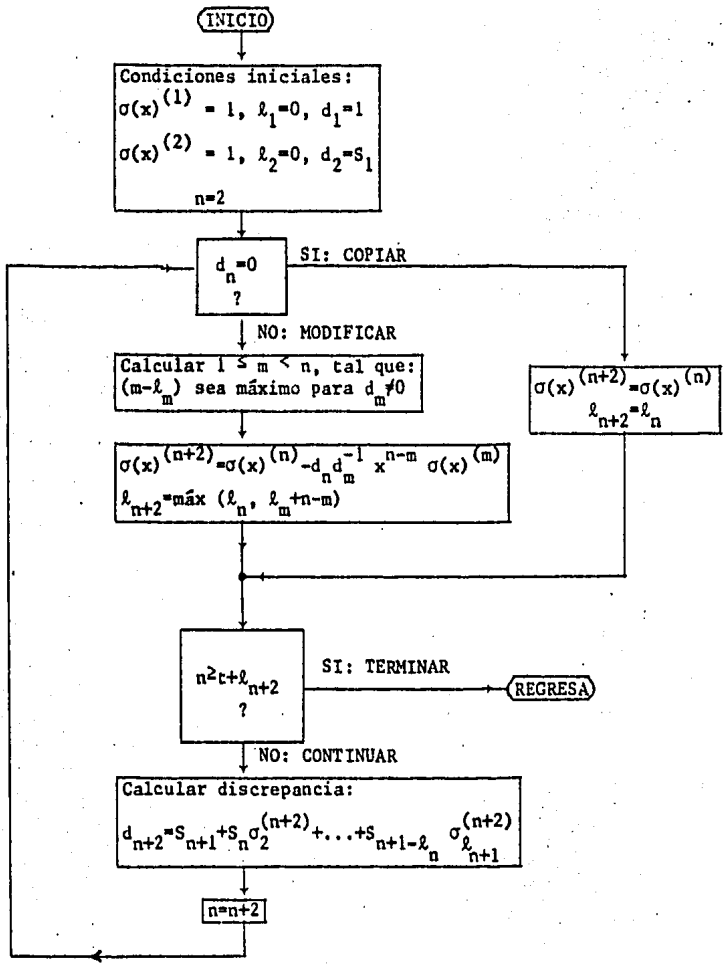


FIGURA D.6. DIAGRAMA DE FLUJO DEL ALGORITMO DE BERLEKAMP-MASSEY PARA EL CASO BINARIO.

tos.

Para la construcción de la tabla de suma, cada renglón de la matríz se suma con los demás (mediante la operación lógica XOR, descrita en el párrafo D.4) y el renglón (elemento) resultante se localiza en la tabla de logaritmos y se guarda en la posición i,j de la tabla de suma, siendo i y j los renglones asociados a los elementos que se sumaron. Como un ejemplo, si m=3 y $g(x) = 1 + x + x^3$, el elemento 2,4 (logaritmos 0,2): 100 ⊕ 001 = 101 y el elemento resultante se localiza en la tabla de logaritmos: $M_{8,j} = 101$, obteniéndose $TAB_{24} = 8 = TAB_{42}$. La tabla de suma consiste entonces en $2^m x 2^m$ enteros y permite que el tiempo de procesamiento se reduzca significativamente, una suma entre los elementos dados por los renglones i,j se reduce a una simple referencia al elemento TAB_{ij} de la tabla de suma. En las figura D.7 y D.8 se presentan los diagramas de flujo para la construcción de las tablas de logaritmos y de suma.

Mediante la utilización de una tabla de logaritmos, las operaciones entre los elementos de $CG(2^m)$ son realizadas haciendo referencia a su logaritmo (i.e. a su renglón en la matriz de elementos). De esta forma, la multiplicación de elementos consiste en la suma algebraica de los logaritmos y la exponenciación es una multiplicación entera.

Ambas operaciones se describen en las figuras D.9 y D.10.

D.7 SIMULACION DE CODIGOS DE REED-SOLOMON.

Para simular las diversas etapas involucradas en la codificación y decodificación de códigos de Reed-Solomon, se implementó el programa cuyo diagrama de flujo se muestra en la figura D.11.

Como datos iniciales, el usuario debe introducir: el polinomio generador g(x) y el orden multiplicativo m del campo $CG(2^m)$; la capacidad de corrección t y la longitud del código N. El programa calcula los parámetros restantes del código, es decir: la longitud nominal $N_{nom} = 2^m - 1$; el orden de acortamiento $i_{acor} = N_{nom} - N$, la distancia mínima $d = 2t+1$, la dimensión nominal $k_{na} = N_{nom} - 2t$ y la dimensión del código $k = k_{na} - i_{acor}$. El programa continua su ejecución construyendo las tablas, de logaritmos y de

DIAGRAMA DE FLUJO
CALCULO DEL CAMPO DE GALOIS, $CG(2^m)$
(TABLA DE LOGARITMOS)

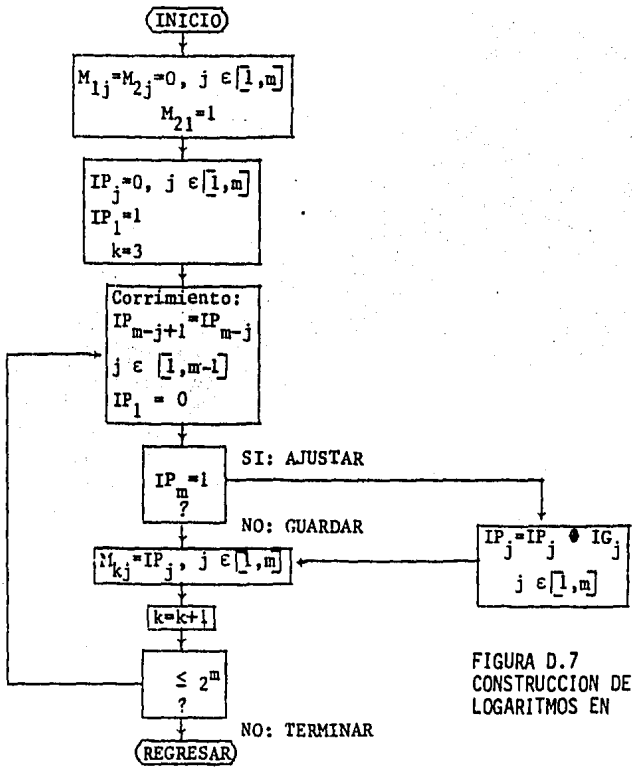


FIGURA D.7
CONSTRUCCION DE LA TABLA DE LOGARITMOS EN $CG(2^m)$

DIAGRAMA DE FLUJO
CONSTRUCCION TABLA DE SUMA EN $CG(2^m)$

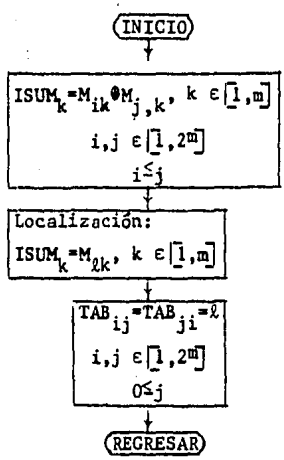


FIGURA D.8
CONSTRUCCION DE LA TABLA DE SUMA EN $CG(2^m)$

DIAGRAMA DE FLUJO
MULTIPLICACION DE ELEMENTOS EN $CG(2^m)$

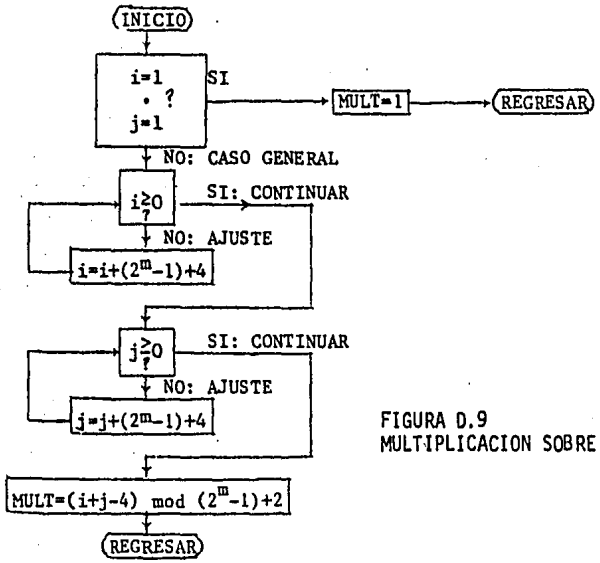


FIGURA D.9
MULTIPLICACION SOBRE $CG(2^m)$

DIAGRAMA DE FLUJO
EXPONENCIACION DE UN ELEMENTO EN $CG(2^m)$

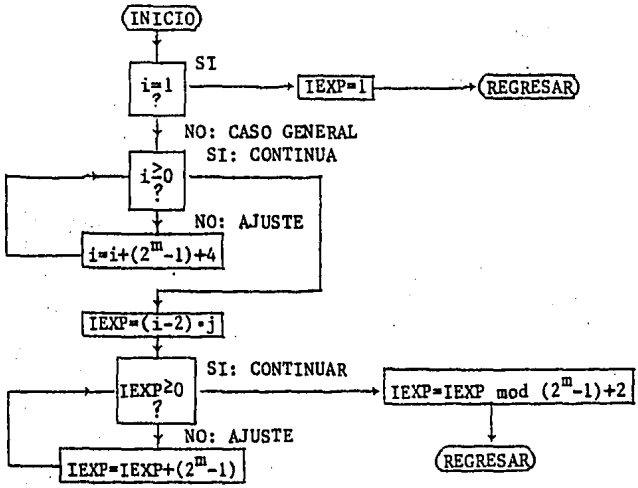


FIGURA D.10
EXPONENCIACION SOBRE $CG(2^m)$

suma, sobre $CG(2^m)$, mediante las subrutinas descritas en el párrafo anterior. Posteriormente, el programa calcula los polinomios generador $G(x)$ y de paridad $H(x)$ del código, usando la tabla de suma y la función de multiplicación. Después, el usuario introduce la información a codificar, especificando los renglones correspondientes a los símbolos de información, de m bits cada uno. Se procede a la codificación de los símbolos de información, al terminar esta, la palabra de código se despliega en la terminal.

Para simular el ruido del canal, y las decisiones de símbolos borrados hechas por el demodulador, el usuario proporciona desde la terminal tanto el número de errores y sus posiciones y valores como el número de borrados y sus respectivas posiciones y valores.

De esta manera, el programa prosigue su ejecución con la decodificación de la palabra recibida. Al terminar la decodificación, el programa despliega en la terminal el resultado (decodificación sin errores, imposible, errónea o exitosa) y la palabra decodificada.

Finalmente, en la figura D.12, se muestra el algoritmo de Berlekamp-Massey aplicado en el caso no binario y considerando que se tiene capacidad de corrección de símbolos borrados.

DIAGRAMA DE FLUJO
SIMULACION DE CODIGOS DE REED-SOLOMON

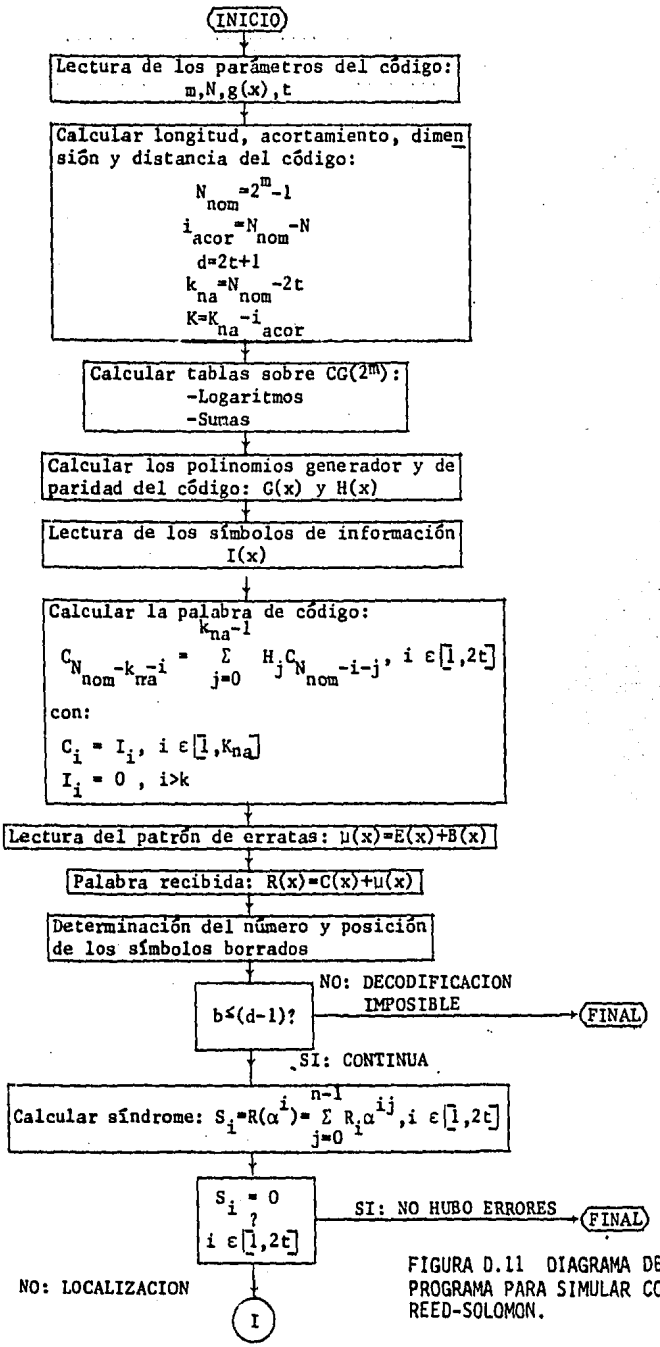


FIGURA D.11 DIAGRAMA DE FLUJO DEL PROGRAMA PARA SIMULAR CODIGOS DE REED-SOLOMON.

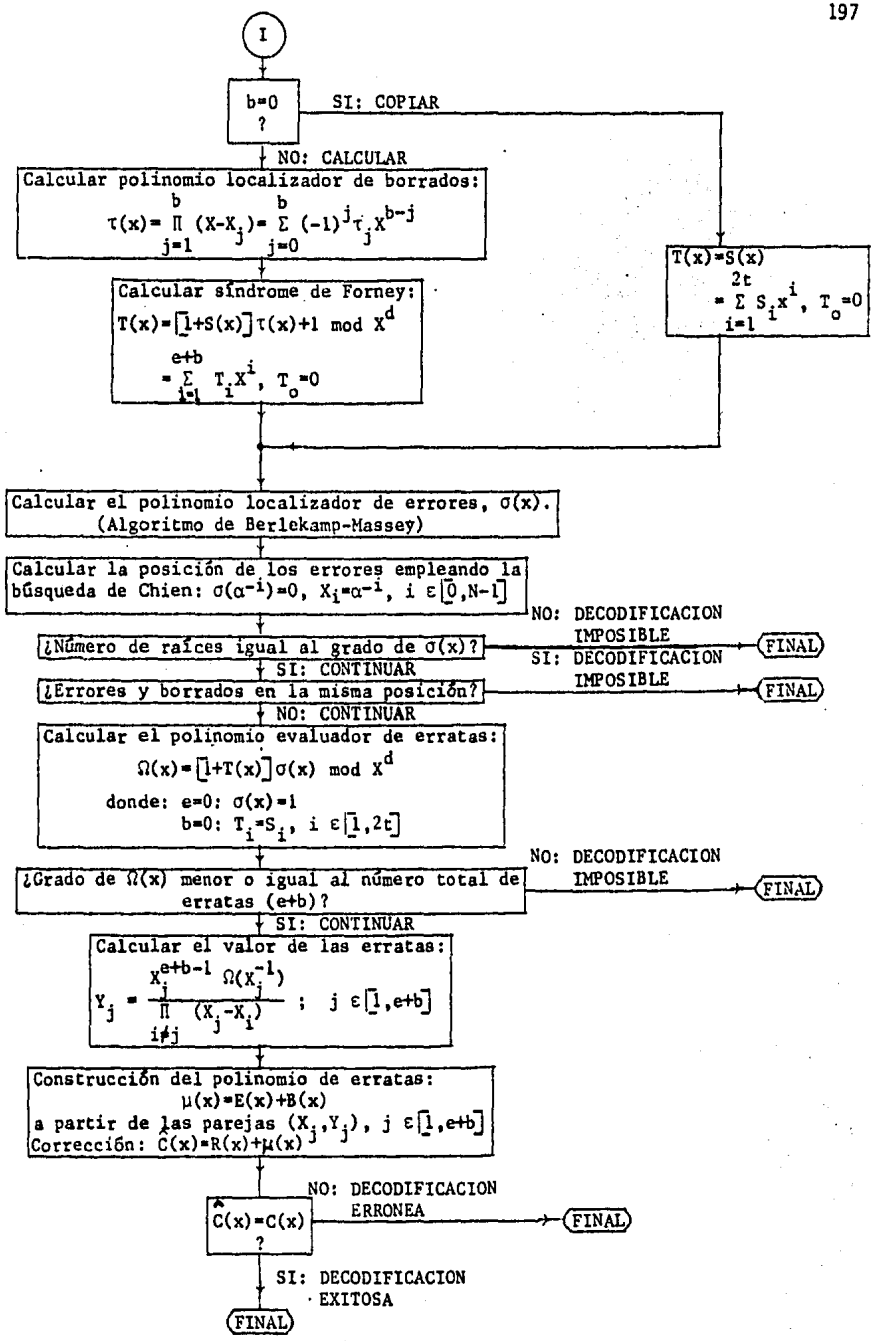


FIGURA D.11 (Cont.)

ALGORITMO DE BERLEKAMP
(CASO NO BINARIO
Y CON BORRADOS)

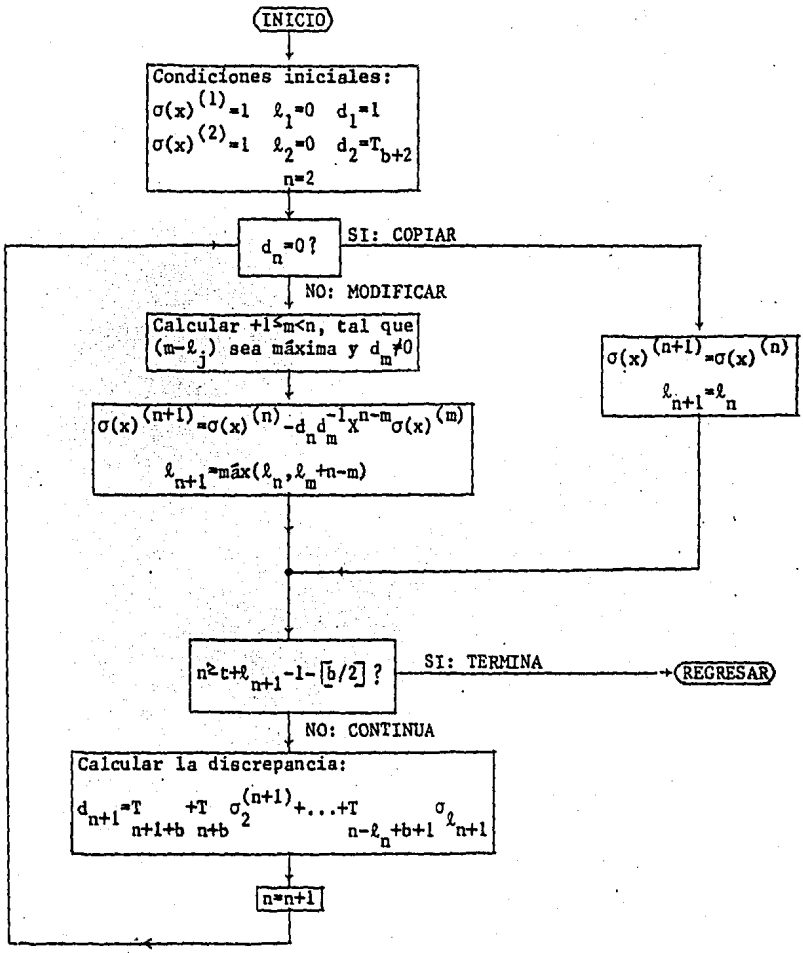


FIGURA D.12 DIAGRAMA DE FLUJO DEL ALGORITMO DE BERLEKAMP-MASSEY PARA EL CASO NO BINARIO Y CON BORRADOS.

APENDICE E

CODIFICACION DE MUESTRAS DE VOZ MEDIANTE EL CODIGO REED-SOLOMON (62,56,7)

APENDICE E
CODIFICACION DE MUESTRAS DE VOZ MEDIANTE EL CODIGO DE REED-SOLOMON (62,56,7).

E.1 INTRODUCCION. Los códigos de Reed-Solomon (R.S.) son empleados generalmente en aquellos sistemas digitales de comunicación en los que los errores se presentan como paquetes múltiples de bits en error [4], así como en los que se utilizan esquemas de codificación por concatenación [12, 18].

Recientemente [30], los códigos R.S. se han implementado exitosamente en los sistemas digitales de audio de disco compacto. En esos sistemas, el código R.S. corrige tanto los errores aleatorios causados por imperfecciones del disco (introducidos en el proceso de fabricación), como los paquetes de errores debidos a ralladuras y marcas en la superficie del disco. Además, detecta errores que aunque no puede corregir, su efecto es atenuado mediante un filtrado posterior.

El código, implementado electrónicamente en el reproductor digital de disco compacto mediante un circuito integrado y una memoria de acceso aleatorio (RAM), es tan poderoso que puede corregir completamente un paquete en error de hasta 8232 bits consecutivos, correspondiente a una banda de pistas de 2.3 mm. en el disco. Además, se pueden detectar y corregir parcialmente errores de hasta 28224 bits, es decir una banda de pistas de 7.8 mm. Lo anterior se logra mediante un proceso conocido como interpolación, en el que los valores incorrectos de los bloques de bits encontrados en error son estimados a partir de los valores de los bloques anteriores y subsecuentes.

Basándose en esta idea se implementó, en la computadora PDP 11/40 del laboratorio CAD de la División de Estudios de Posgrado de la Facultad de Ingeniería, la simulación de un código R.S. (62,56,7) aplicado en la protección de muestras de voz.

E.2 DESCRIPCION DEL PROGRAMA Y SUBRUTINAS. Se empleó un paquete de programas, ya existente en el laboratorio, para el muestreo y reproducción de la se

ñal de voz. Para limitar en banda la señal antes del muestreo, se utilizó un filtro externo paso-banda con frecuencias de corte de 100 Hz y 3200 Hz. La tasa de muestreo se eligió igual a la de Nyquist, es decir 6400 muestras por segundo.

Una vez muestreada la señal, las muestras se almacenaron en un archivo de datos grabado en disco. Debido a que cada muestra está representada en la computadora mediante un número entero de 12 bits, fué seleccionado un código R.S. que trabajará con símbolos de 6 bits, es decir con una longitud nominal $n = 2^6 - 1 = 63$. Para facilitar la sincronía de los bloques de datos del archivo con las muestras de la señal de voz, la dimensión del código se seleccionó como el número par más cercano a la dimensión nominal, considerando una distancia de 7, se obtuvo $k = 56$.

Una vez seleccionado el código, el siguiente paso fué adecuar uno de los programas desarrollados en esta tésis (Simulación de códigos de Reed-Solomon (Apéndice D)), a las diversas operaciones necesarias en el tratamiento de las muestras de voz.

Se elaboró una subrutina (escrita en lenguaje FORTRAN como las anteriores) que hiciera la partición de una muestra en dos números en base octal (de dos cifras cada uno), y otra que efectuara la operación inversa. Una subrutina mas se implementó para hacer el mapeo entre la representación en base octal y el campo de Galois CG (64), mediante la construcción de dos tablas de conversión en memoria (una de octal a logaritmo base α, $\alpha \varepsilon CG\ (64)$ y la otra de logaritmo base α a octal). Las operaciones anteriores están representadas en la figura E.1, mediante el bloque intitulado "Conversión de base".

Una vez efectuada la conversión de base de las muestras (en una implementación electrónica, las muestras serían alimentadas directamente al codificador), éstas son codificadas y se simula un canal de transmisión. Recuérdese que, si se tratara del sistema digital de audio de disco compacto, las muestras codificadas serían grabadas en disco.

El canal se simuló con una distribución estadística de errores uniforme, en lo referente tanto al valor como a la posición del símbolo en error. El número de errores por palabra de código, introducidos en el canal (o

en la lectura de las muestras codificadas en el disco compacto), se consi_ deró con una distribución gaussiana condicional $(f(x|x>0))$ y discreta.

Las muestras recibidas por el decodificador son almacenadas en un ar_ chivo de datos, con el objeto de poder conocer la señal contaminada por el ruido. En el decodificador, se consideran tres eventos, a saber: De_ codificación correcta (corrección), decodificación imposible (detección) y decodificación errónea (cuando la palabra decodificada es diferente de la palabra transmitida, este último evento se puede contabilizar solamen_ te en una simulación). Al terminar la decodificación se hace un conteo del evento ocurrido y las muestras decodificadas son almacenadas en un tercer archivo de datos.

El procedimiento anterior continua palabra por palabra (donde cada pa_ labra contiene 28 muestras de 12 bits cada una) hasta que la totalidad de las muestras de voz han sido codificadas (grabadas en disco), enviadas al canal (leídas de disco) y decodificadas (reproducidas). En la figura E.1, se muestra el diagrama de bloques donde se presentan las diversas etapas de la simulación.

E.3 RESULTADOS. Se realizaron varias ejecuciones del programa simulador, des_ crito en el párrafo anterior, para diferentes estadísticas del canal. Después de cada ejecución se obtuvo la relación señal a ruido (SNR), tan_ to de la señal contaminada como de la señal recuperada. Además, se extra_ jo la señal de ruido antes y después del decodificador. En las figuras E.2-E.6, se muestran las gráficas de las señales de ruido y voz, para va_ rios valores de la variancia (σ) de la distribución del número de errores y borrados (en todos los casos se tomó el mismo valor de variancia para los errores y los borrados).

En la figura E.7 se graficó la relación señal a ruido en decibeles a la entrada (SNR_i) y a la salida (SNR_o) del decodificador, contra la va_ riancia del número de errores y borrados, donde:

$$SNR_i = 10 \log_{10} \frac{\sum_{i=1}^{N} S_i^2}{\sum_{i=1}^{N} (S_i - r_i)^2}$$

$$SNR_o = 10 \log_{10} \frac{\sum_{i=1}^{N} S_i^2}{\sum_{i=1}^{N} (S_i - \hat{r}_i)^2}$$

siendo S_i: i-ésima muestra de la señal de voz transmitida (grabada)

r_i: i-ésima muestra de la señal de voz recibida (leída)

$\hat{r}_i$: i-ésima muestra de la señal de voz decodificada (reproducida)

N: número total de muestras de la señal de voz analizada.

En la Tabla E.1 se muestran los resultados numéricos de la simulación:

σ	SNR_i	SNR_o	NDC	NDI	NDE	C1%	C2%
0.5	21.50	356.60	146	0	0	100.0	100.0
1.0	12.55	65.90	145	1	0	99.3	100.0
1.5	8.76	15.60	139	7	0	95.2	100.0
2.0	7.55	13.11	133	10	3	91.1	97.9
2.5	5.75	9.74	120	19	7	82.2	95.2
3.0	3.89	6.68	105	27	14	71.9	90.4

Tabla E.1 Resultados numéricos obtenidos en la simulación del código R.S. (62,56,7) aplicado en la protección de muestras de voz.

El significado de cada variable en la tabla anterior es el siguiente:

σ: Variancia de la distribución del número de errores y borrados.

SNR_i, SNR_o: Relación señal a ruido a la entrada y a la salida del decodificador, respectivamente.

NDC: Número de muestras sin error o corregidas.

NDI: Idem detectadas en error.

NDE: Idem decodificadas erroneamente.

C1%: Porcentaje de palabras corregidas.

C2%: Idem de palabras decodificadas correctamente.

Finalmente, en la figura E.8 se grafican los porcentajes de palabras corregidas (C1%) y de palabras decodificadas correctamente (C2%), para los valores de σ que se emplearon en la simulación.

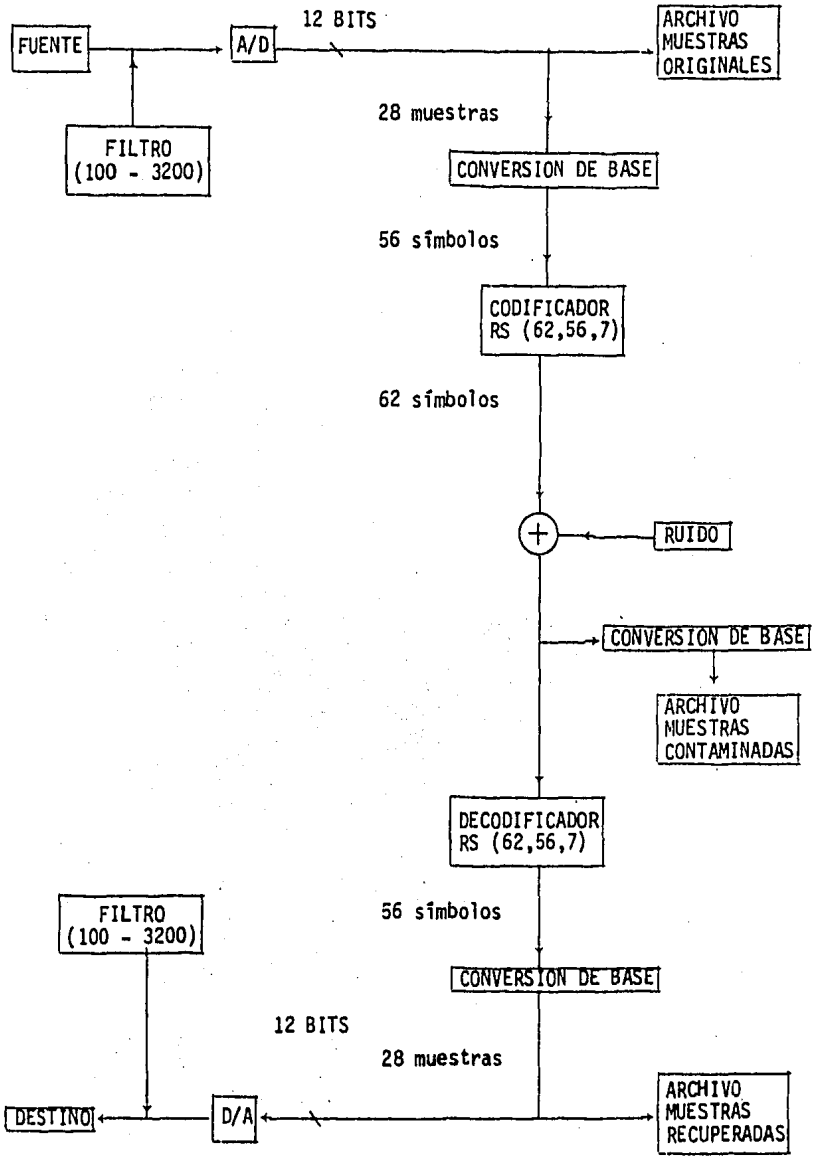
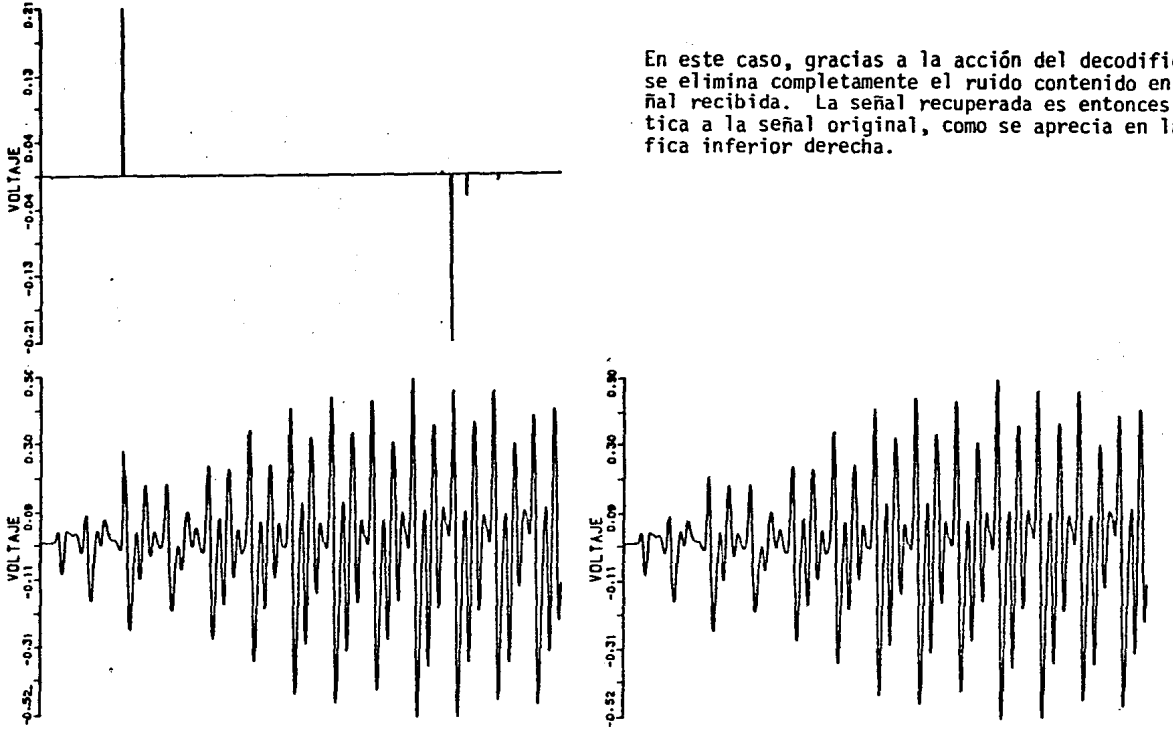


FIGURA E.1

ESQUEMA DE CODIFICACION DE MUESTRAS DE VOZ, MEDIANTE UN CODIGO DE REED-SOLOMON

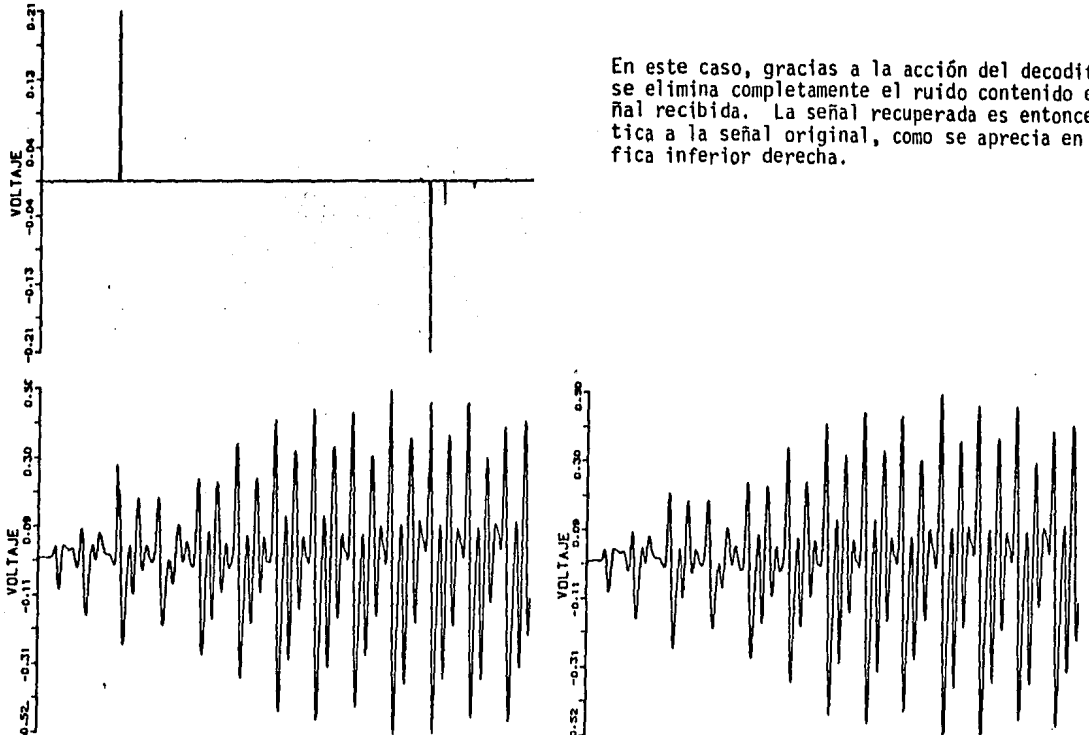
En este caso, gracias a la acción del decodificador, se elimina completamente el ruido contenido en la señal recibida. La señal recuperada es entonces idéntica a la señal original, como se aprecia en la gráfica inferior derecha.



(a) SNR = 21.50 dB

(b) SNR = ∞

Figura E.2. Muestras de la señal de voz (de 96 mseg de duración) y el ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=0.5.

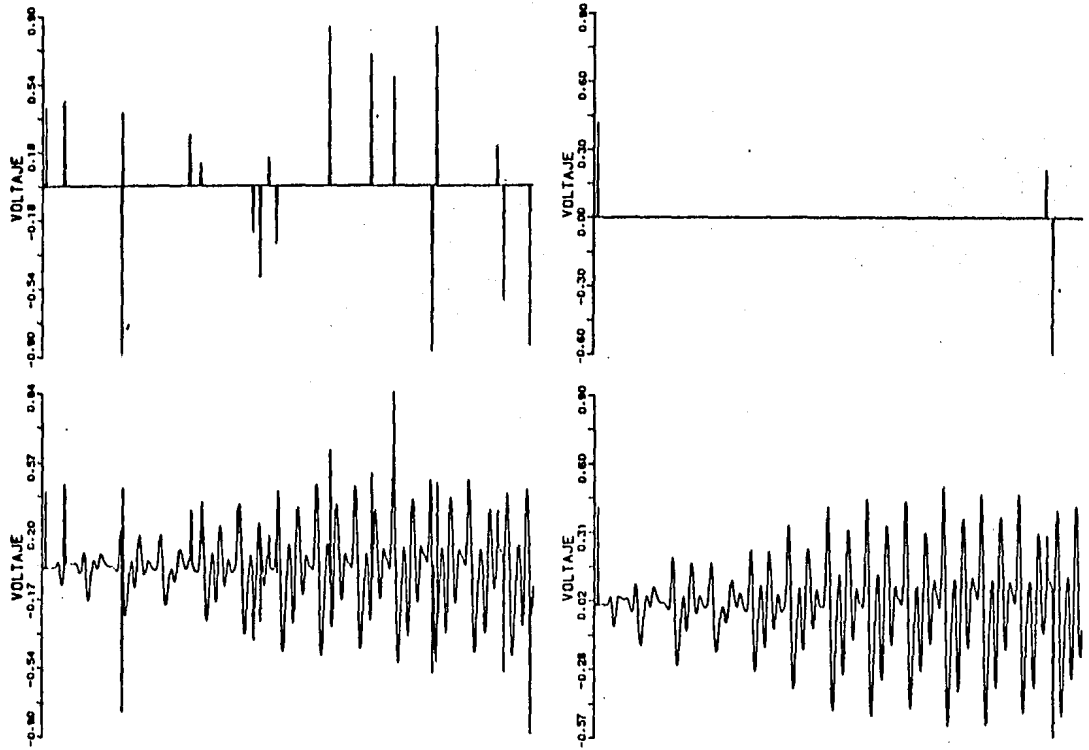


En este caso, gracias a la acción del decodificador, se elimina completamente el ruido contenido en la señal recibida. La señal recuperada es entonces idéntica a la señal original, como se aprecia en la gráfica inferior derecha.

(a) SNR = 21.50 dB

(b) SNR = ∞

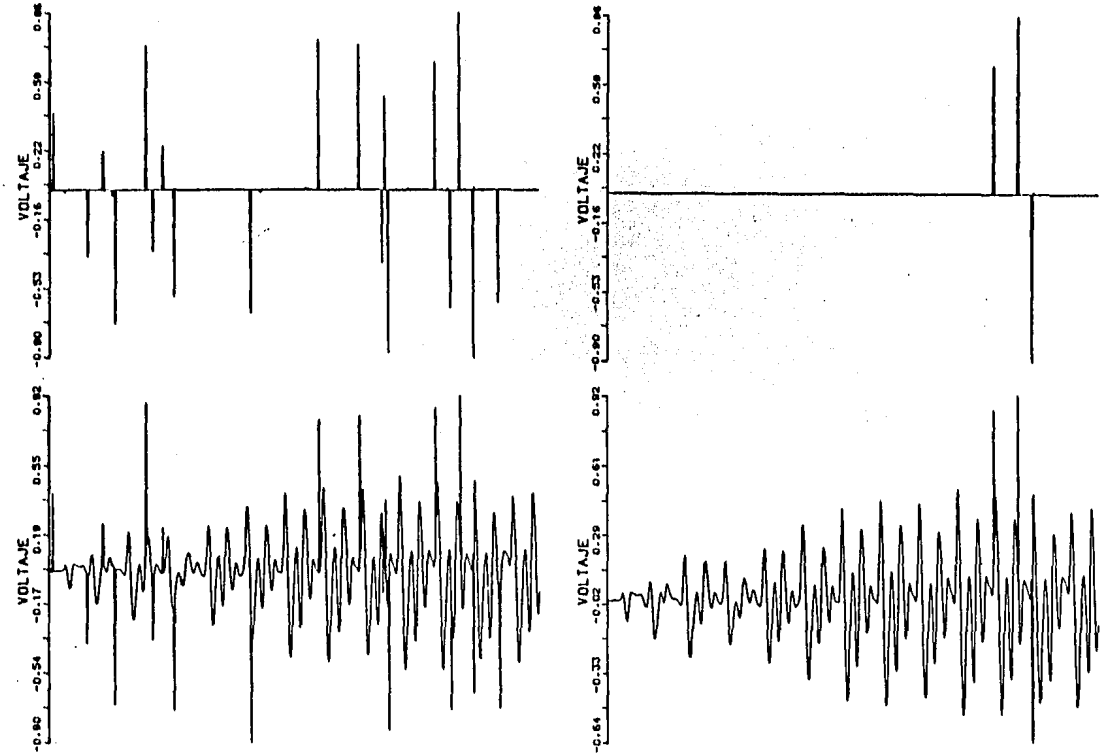
Figura E.2. Muestras de la señal de voz (de 96 mseg de duración) y el ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=0.5.



(a) SNR = 12.55 dB

(b) SNR = 65.90 dB

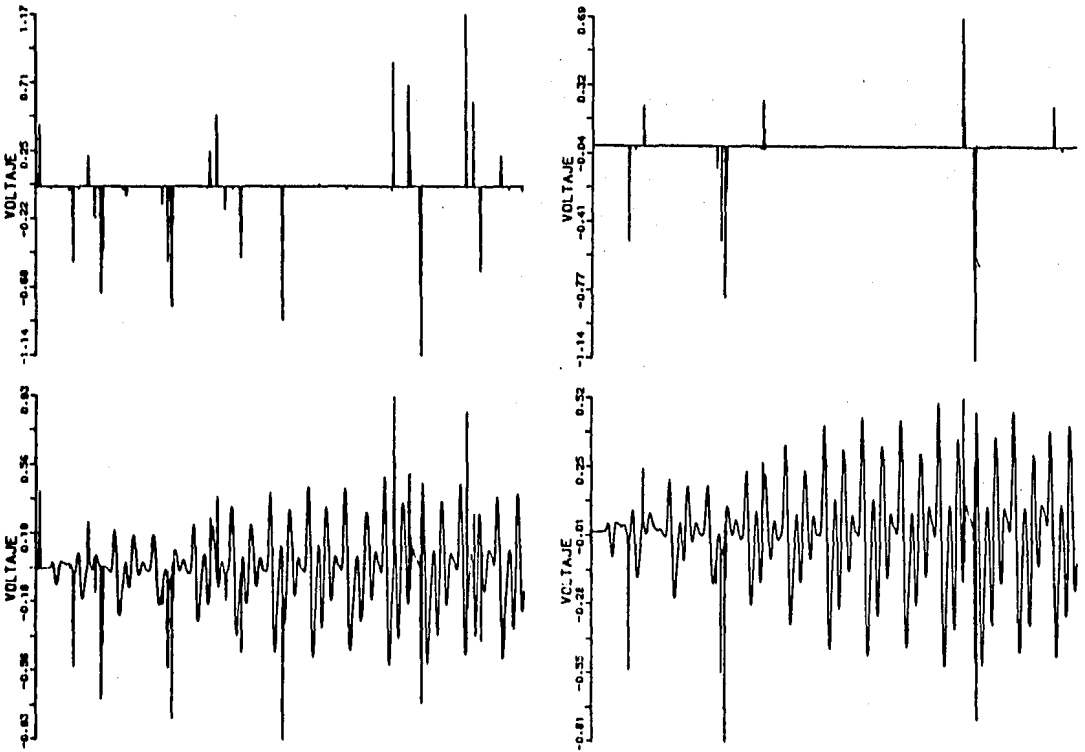
Figura E.3. Muestras de la señal de voz (de 96 mseg de duración) y del ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=1.0.



(a) SNR = 8.76 dB

(b) SNR = 15.60 dB

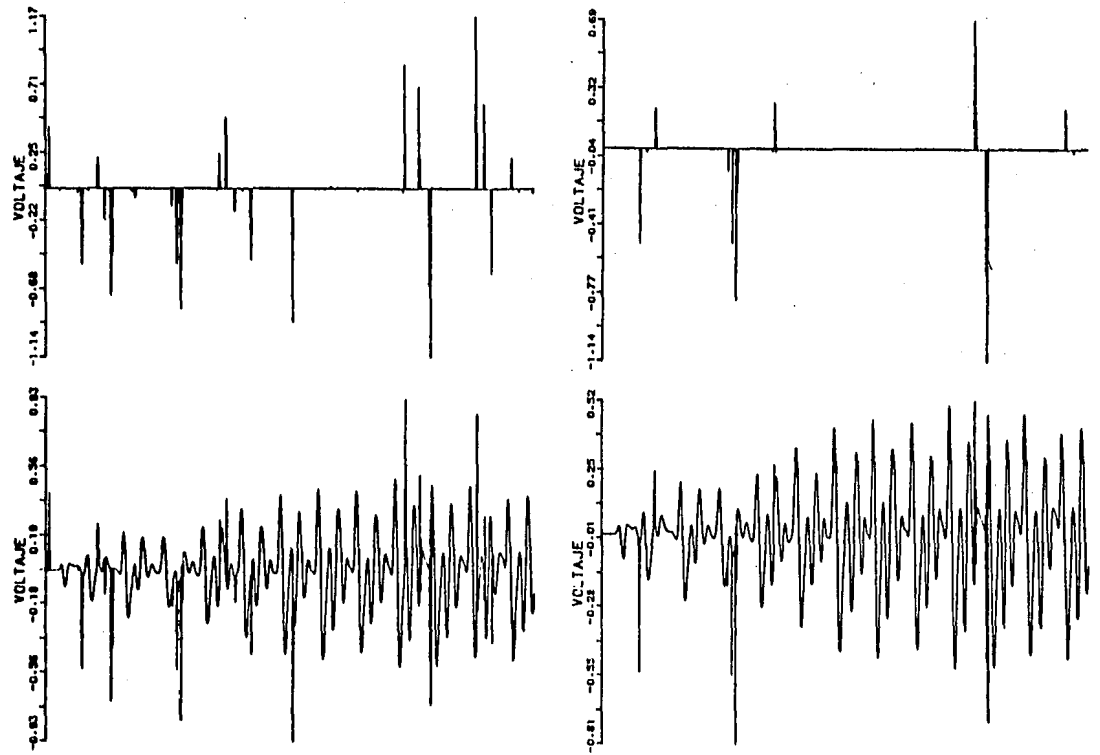
Figura E.4. Muestras de la señal de voz (de 96 mseg. de duración) y del ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=1.5.



(a) SNR =7.55 dB

(b) SNR = 13.11 dB

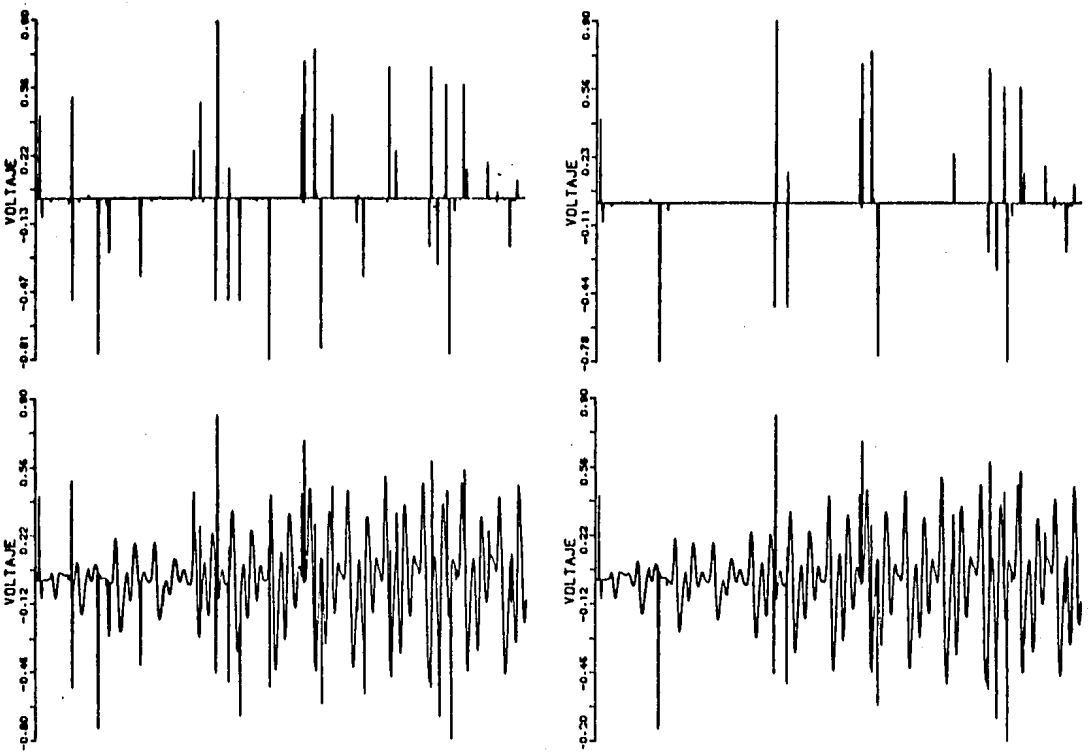
Figura E.5. Muestras de la señal de voz (de 96 mseg. de duración) y del ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=2.0.



(a) SNR =7.55 dB

(b) SNR = 13.11 dB

Figura E.5. Muestras de la señal de voz (de 96 mseg. de duración) y del ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=2.0.



(a) SNR = 5.75 dB

(b) SNR = 9.74

Figura E.6 Muestras de la señal de voz (de 96 mseg de duración) y del ruido contenido en ésta, antes (a) y después (b) del decodificador RS(62,56,7), para σ=2.5.

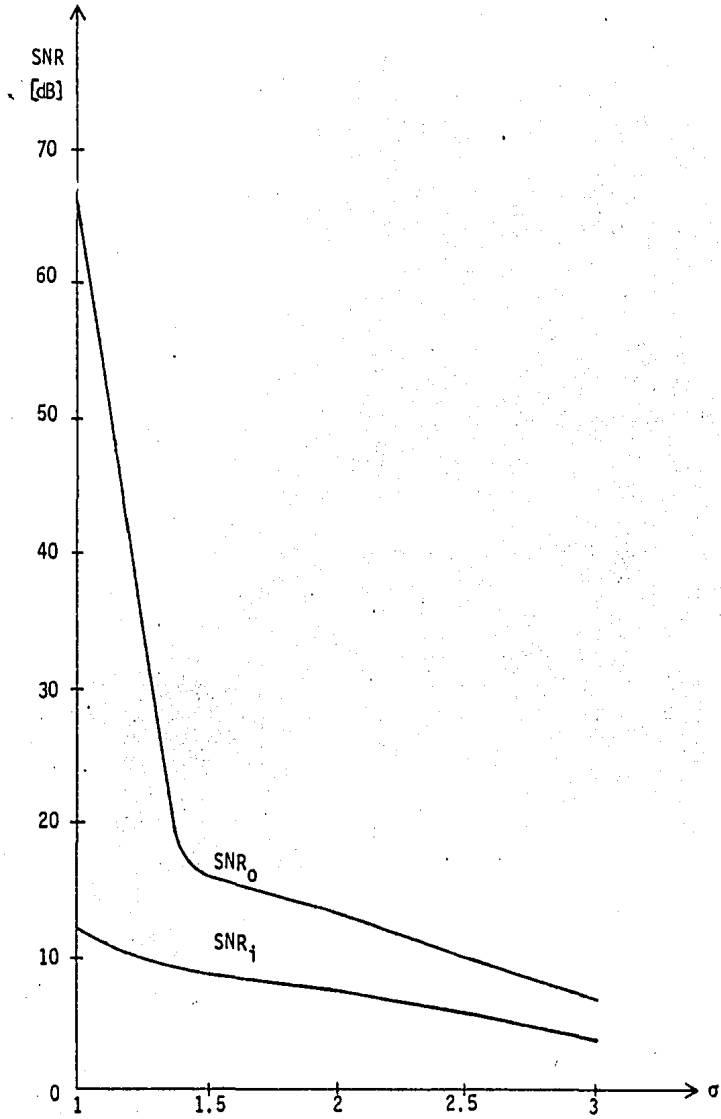


FIGURA E.7 Relación señal-a-ruido a la entrada (SNR_i) y a la salida (SNR_0) del decodificador.

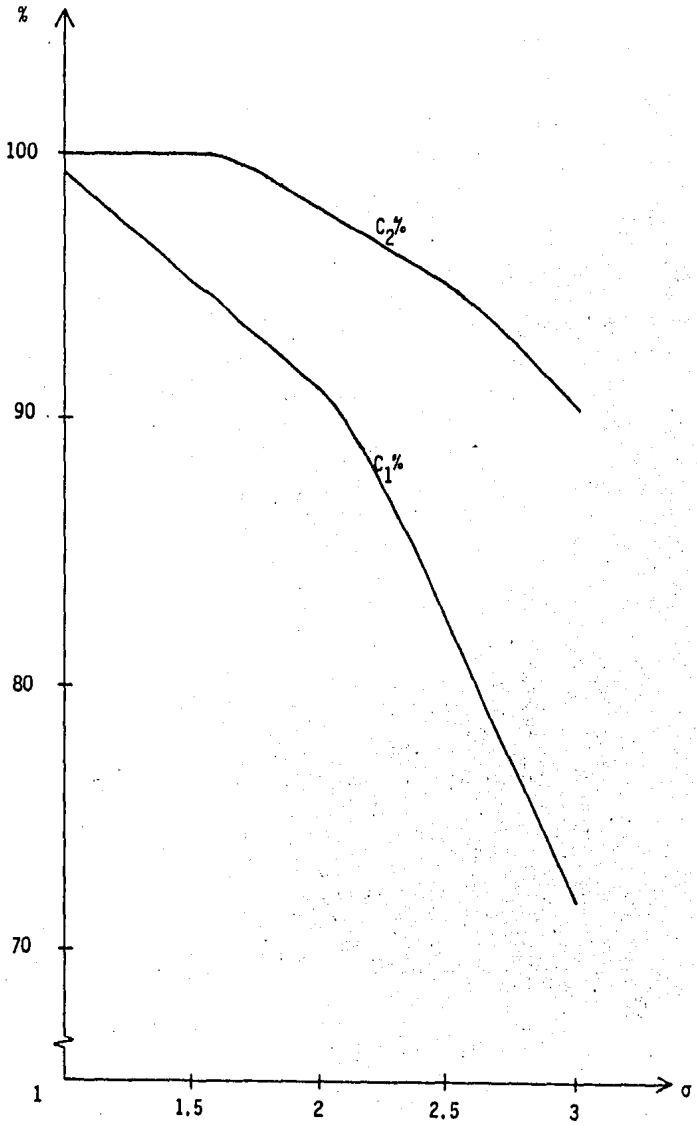


FIGURA E.8 $C_1\%$ es el porcentaje de palabras corregidas exitosamente. $C_2\%$ es el porcentaje de palabras corregidas y/o detectadas.

B I B L I O G R A F I A

B I B L I O G R A F I A

1. Balakhrisnan, A.V.
"Communication Theory"
McGraw-Hill, 1968.

2. Berlekamp, E.R.
"Algebraic Coding Theory"
McGraw-Hill, 1968

3. Berlekamp, E.R.
"Key Papers in The Development of Coding Theory"
IEEE Press, 1974.

4. Berlekamp, E.R.
"The Technology of Error-Correnting Codes"
Proc. IEEE - Vol. 68, No. 5, May, 1980.

5. Berlekamp, E.R. and Ramsey, J.L.
"Readable Erasures Improve the Performance of Reed-Solomon Codes"
IEEE Trans. Info. Theory - Vol. IT-24, No. 5, September, 1978.

6. Bhargava, V.K.
"Efficient Block Coding Techniques"
Seminario México-Canadá de Tecnología de Comunicaciones
México, D.F., Marzo, 1984.

7. Chen, C.L.
"High-Speed Decoding of BCH Codes"
IEEE Trans. Info. Theory - Vol. IT-27, No. 2, March 1981.

8. Chien, R.T.
"Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes"
IEEE Trans. Info. Theory - Vol. IT-10, No. 10, October, 1964.

9. Chien, R.T.
"Burst-Correcting Codes with High-Speed Decoding"
IEEE Trans. Info, Theory - Vol. IT-15, No. 1, January, 1960.

10. Chien, R.T., Cunningham, B.D. and Oldham, I.B.
"Hybrid Methods for Finding Roots of a Polynomial - with Application to BCH Decoding"
IEEE Trans. Info. Theory - Vol. IT-15 No. 2, March 1969.

11. Forney, G.D. Jr.
"On Decoding BCH Codes"
IEEE Trans. Info. Theory - Vol. IT-11 No. 10, October 1965.

12. Forney, G.D. Jr.
"Coding and its Applications in Space Communications"
IEEE Spectrum - Vol. 7 No. 6, June 1970.

13. Hamming, R.W.
"Coding and Information Theory"
Prentice Hall, 1980

14. Harari, S. and Nasse, D.
"Protection contre les Erreurs en Enregistrement Magnêtique Numêrique de Television"
Ann. Telecommunic. 34, No. 7-8, 1979.

15. Huntoon, Z.M. and Michelson, A.M.
"On the Computation of the Probability of Post-Decoding Error Events for Block Codes"
IEEE Trans. Info. Theory - Vol. IT-23, No. 5, May 1977.

16. Jacobs, I.M.
"Practical Applications of Coding"
IEEE Trans. Info. Theory - Vol. IT-20, No. 5, May 1977

17. Justesen, J.
"On the Complexity of Decoding Reed-Solomon Codes"
IEEE Trans. Info. Theory - Vol. IT-22, No. 5, May 1976

18. Kuang Y. Liu and Jun-Ji L.
"Recent Results on the Use of Concatenated Reed-Solomon/Viterbi Channel Coding and Data Compression for Space Communications"
IEEE Trans. Comm. - Vol. COM-32 No. 5, May 1984.

19. Kasami, R., Klove, T. and Lin, S.
"Linear Block Codes for Error Detection"
IEEE Trans. Info. Theory - Vol. IT-29, No. 1, January 1983.

20. Kodek, D.M.
"Conditions for the Existence of Fast Theoretic Transforms"
IEEE Trans. on Computers - Vol. C-30 No. 5, May 1981.

21. Leibowitz, L.M.
"A Simplified Binary Arithmetic for the Fermat Number Transform"
IEEE Trans. on ASSP - Vol. ASSP-24 No. 5, October, 1976.

22. Leung, K.S. and Welch, L.R.
"Erasure Decoding in Burst-Error Channels"
IEEE Trans. Info. Theory - Vol. IT-27, No. 2, March 1981.

23. Liu, K.Y., Reed, I.S. and Truong, T.K.
"High-Radix Transforms for Reed-Solomon Codes over Fermat Primes"
IEEE Trans. Info. Theory - Vol. IT-23, No. 6, Nov. 1977.

24. Mac Williams, F.J. and Sloane, N.J.A.
"The Theory of Error Correcting Codes"
North Holland Publishing Co, 1977

25. Mandelbaum, D.
"On Decoding of Reed-Solomon Codes"
IEEE Trans. Info. Theory - Vol. IT-17 No. 1, Feb. 1971.

26. Massey, J.L.
"Shift_Register Synthesis and BCH Decoding"
IEEE Trans. Info. Theory - Vol. IT-15 No. 1, Jan. 1969.

27. Matt, H.J. and Massey, J.L.
"Determining the Burst-Correcting Limit of Cyclic Codes"
IEEE Trans. Info. Theory - Vol. IT-26 No. 3, May, 1980.

28. McClellan, J.H.
"Hardware Realization of a Fermat Number Transform"
IEEE Trans. on ASSP - Vol. ASSP-24 No. 3, June 1976.

29. Miller, R.L., Truong, T.K. and Reed, I.S.
"Efficient Program for Decoding the (255,223) Reed-Solomon Code over GF(2^8) with both Errors and Erasures, Using Transform Decoding"
Proc. IEEE - Vol. 127 No. 4, July 1980.

30. Miyaoka, S. (Sony Corp).
"Digital Audio is Compact and Rugged"
IEEE Spectrum - Vol. 21 No. 3, March 1984.

31 Peterson, W.W. and Weldon, E.J.
"Error-Correcting Codes"
The MIT Press, 1972.

32. Peterson, W.W.
"Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes",
IRE Trans. Info. Theory, Vol. IT-6 No. 5, June 1960.

33. Reed, I.S., Scholtz, R.A., Truong, R.K. and Welch, L.R.
"The Fast Decoding of Reed-Solomon Codes Using Fermat Theoretic Transforms and Continued Fractions"
IEEE Trans. Info. Theory - Vol. IT-24 No. 1, January 1978.

34. Reed, I.S., Truong, T.K. and Miller, R.L.
"Decoding of BCH and RS Codes With Errors and Erasures using Continued Fractions"
Electronic Letters, Vol. 15, No. 17.

35. Reed, I.S., Truong, T.K. and Miller, R.L.
"Simplified Algorithm for Correcting both Errors and Erasures of Reed-Solomon Codes"
Proc. IEEE, Vol. 126, No. 10, October, 1979.

36. Reed, I.S., Truong, T.K. and Welch, L.R.
"The Fast Decoding of Reed-Solomon Codes Using Fermat Transforms"
IEEE Trans. Info. Theory, Vol. IT-24 No. 4, July 1978

37. Schwartz, M.
"Information Transmission, Modulation and Noise"
McGraw-Hill, 1980.

38. Shannon, C.E. and Waver, W.
"The mathematical Theory of Communication"
Univ. of Illinois, 1969.

39. Truong, T.K., Miller, R.L. and Reed, I.S.
"Fast Technique for Computing Syndromes of BCH and RS Codes"
Electronic Letters , Vol. 15, No. 22

40. Wagner, W.
"Best Fire Codes with Lenght up to 1200 bits"
IEEE Trans. Info. Theory - Vol. IT-16 No. 5, September 1970

41. Waii, P.K.S., and Siegenthaler, T.
"Practical Transform Techniques for Error and Erasure Correction"
Electronic Letters, May 1983.

42. Wolf, J.K., Michelson, A.M. and Levesque, A.H.
"On the Probability of Undetected Error for Linear Block Codes"
IEEE Trans. Comm. Vol. COM-32 No. 2, Febrary 1982.

43. Jovanovic V.M. and Budisin S.Z.
"On the Coding Gain of Linear Binary Block Codes"
IEEE Trans. Comm. Vol. COM-32 No. 5, May 1984.

44. Torrieri, Don J.
"The Information-Bit Error Rate for Block Codes"
IEEE Trans. Comm - Vol. COM-32 No. 4, April, 1984.

45. MEX68KECB Educational Computer Board User's Manual
Motorola Semiconductors, Inc. July, 1982.

46. MC68000 16-bit Microprocessor User's Manual Third Edition
Motorola Semiconductors, Inc., 1982

47. MC68230 Parallel Interface/Timer Advance Information
Motorola Semiconductors, Inc., 1984.