UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

UN ALGORITMO PARA LA DESCODIFICACION DE CODIGOS CONVOLUCIONALES Y SU APLICACION EN UN SISTEMA DE TELEFONIA DIGITAL

T E S I S

Que para obtener el Título de

INGENIERO MECANICO ELECTRICISTA

P r e s e n t a n

SANTIAGO ISRAEL MONTERROSA REYES
FERNANDO FERNANDEZ VARELA LOPEZ
LUIS ANTONIO VALLEJO FLORES
ALAN ALBERTO RANGEL GONZALEZ

DIRECTOR: M. en I. FERNANDO LEPE CASILLAS

México, D. F. *1994*

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

A mis padres : Gaspar y Conchita por su apoyo incondiconal para llegar a realizar éste trabajo y como respuesta a su esperanza que fincaron en mi.

A mi esposa e hija : Maribel y Belinda por su amor, apoyo y comprensión siempre presentes en mi durante la realización de éste trabajo.

A mis hermanos : Gaspar, Raúl, Edith, y Joaquin porque siempre han sabido ser mis hermanos.

A mis estimados amigos : por su apoyo moral de siempre.

A la empresa Alcatel Indetel : por el apoyo logístico proporcionado.

Santiago Monterrosa Reyes.

Este trabajo se lo dedico a mi madre sra. Raquel López de Fernández Varela de quién he recibido todo el apoyo y sacrificio

Fernando Fernández Varela López

PARA 2 ILUSIONES
PARA 2 CORAZONES
PARA 2 ESPERANZAS
POR 2 SATISFACCIONES

TODO MI AMOR, RESPETO Y CARIÑO....

A MIS PADRES.

POR SER MI HAMBRE Y MI ALIMENTO
POR SER MI SED Y MI BEBIDA
POR SER MIS PIES Y MI APOYO
POR SER MIS OJOS Y MIS OIDOS
Y ALIMENTAR MI VIDA CON UNA
NUEVA ESPERANZA

TE OFRESCO MI VIDA....

ARELI

En agradecimiento a mis seres queridos que, con su apoyo y comprensión, hicieron posible que lograra alcanzar esta meta.
A cada uno de ellos, ausentes y presentes,
GRACIAS, DIOS LOS BENDIGA.

Alan Alberto Rangel González

INDICE GENERAL

1. INTRODUCCION A LA TELEFONIA DIGITAL 1

1.1 Introducción 2

Generalidades
Notas históricas

1.2 Objetivo de la comunicación digital 5

1.3 Descripción de una red de telefonía digital 6

Características generales
Características funcionales
Descripción del equipo físico

1.4 Sistemas PCM (Pulse Code Modulation) 17

Muestreo
Cuantización
Codificación
Decodificación e Integración
Multiplexaje
Estructura de una trama de canales
Sistemas PCM de Orden Alto

1.5 Sincronización 29

Sincronización de bit
Sincronización de trama
Sincronización de red

1.6 Conmutación digital 32

Evolución de la red telefónica
Red de telefonía analógica
Red de telefonía híbrida
Red digital integrada
Principios de la conmutación digital

1.7 Señalización 38

Señalización en un ambiente MFC analógico
Señalización de registro MFC
Señalización en un ambiente MFC digital
Señalización del canal asociado (CAS)
Señalización de canal común

2. CONCEPTOS FUNDAMENTALES DE LA COMUNICACION DIGITAL CODIFICADA 48

2.1 Introducción 49

2.2 Conceptos Básicos de Codificación 50

2.3 Descripción general de un proceso de comunicación digital 54

Fuente de datos
El codificador
El modulador
El canal de formas de onda
El desmodulador
El descodificador

2.4 El canal de datos discreto 58

2.5 Códigos de bloque y códigos convolucionales 59

Códigos de Bloque
Matriz Generadora y Matriz de Paridad
El Síndrome
Códigos Convolucionales
Representación Gráfica de Códigos Convolucionales
Representación Analítica de Códigos Convolucionales
Propiedades de Distancia para Códigos Convolucionales
Cotas de distancia para Códigos Convolucionales
Propagación de errores
Algoritmo de Descodificación de Viterbi
Operación del Descodificador de Viterbi
Parámetros del Descodificador de Viterbi

3. UN ALGORITMO DE DESCODIFICACION 90

3.1 Introducción 91
3.2 Notación y ecuaciones de codificación y descodificación 93
3.3 El descodificador secuencial 96
3.4 Desarrollo del Algoritmo Propuesto 100
3.5 Definición del Algoritmo Propuesto 101
3.6 Justificación del Algoritmo Propuesto 105
3.7 Diagrama de flujo para calcular las probabilidades de detectar dos diferencias en una etapa A 111
3.8 Listado del programa de evaluación de desempeño óptimo del descodificador 114
3.9 Gráficas de la probabilidad de desempeño del descodificador 117
3.10 Pruebas de simulación 118

4. DESCRIPCION DEL ALGORITMO DE DESCODIFICACION EMPLEADO EN EL SISTEMA DE TELEFONIA DIGITAL 168

4.1 Introducción 169
4.2 Arquitectura de la tarjeta del elemento de control 170
4.3 Descripción de la memoria asociada al microprocesador 173
4.4 Descripción funcional de la unidad de control de proceso integrado IPCU 180
4.5 Descripción del codificador aplicado en la central telefónica digital actualmente 184

APENDICE

CAPITULO 1

INTRODUCCION A LA TELEFONIA DIGITAL

1.1 Introducción

Generalidades [1][2][3]

Un sistema de comunicaciones completo incluye un transmisor, un medio de transmisión sobre el cual la información se transmite, y un receptor, el cual debe producir a su salida una réplica reconocible de la información de la entrada. En la mayor parte de los sistemas de comunicaciones, figura 1.1, la transmisión de información está estrechamente relacionada con la modulación o la variación que sufre en el tiempo una señal senoidal especial, llamada la portadora.

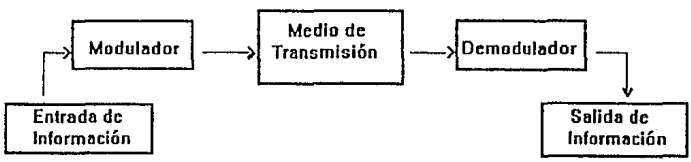


Figura 1.1 Un sistema de comunicaciones

El **transmisor** comprende una fuente de información que será transmitida, y que consiste en señales de audio, de TV, de datos de salida de una computadora, datos de telemetría, etc. La fuente, como cualquiera de las mencionadas arriba, origina un mensaje que para ser una señal eléctrica si es que originalmente no lo era, usa un transductor de entrada que lo convierte a tal señal eléctrica y comunmente se le conoce como señal en **banda base.**

Las señales de banda base producidas por diferentes fuentes de información no son siempre adecuadas para su transmisión directa a través de un canal dado. Al proceso de adecuación para facilitar su transmisión se conoce como **modulación.** En este proceso, que se ejecuta en el **modulador** que indica la figura 1.1 también, se utiliza la señal de banda base para modificar algún parámetro de la señal transmitida, a saber: su amplitud, o su frecuencia, o su fase.

Cuando las señales atraviesan el medio de transmisión o **canal** se distorsionan; aparecen señales de interferencia y ruido por lo que la interpretación correcta de las señales que se reciben en el destino deseado se transforma en una tarea difícil. El canal puede ser un alambre, un cable coaxial, una guía de ondas, una fibra óptica o un enlace de radio.

En el **receptor** la señal modulada debe pasar a través de un proceso inverso que se llama demodulación con el fin de recuperar la señal de banda base. Así, la señal que se presenta a la salida del demodulador alimenta a un transductor de salida que convierte la señal eléctrica a su forma original.

El propósito de un sistema de comunicaciones es transportar información desde una fuente hasta un usuario destino a través del canal de comunicación. Básicamente un sistema de comunicaciones es de tipo analógico o de tipo digital. En un sistema de comunicación analógico las señales que representan la información varían en amplitud y tiempo de manera continua. Por otro lado, en un sistema de comunicación digital las señales que representan la información se procesan de tal forma que pueden representarse por una secuencia de mensajes discretos.

Las comunicaciones digitales han experimentado un desarrollo fenomenal tanto en su teoría como en su aplicación. El desarrollo de las comunicaciones digitales es extenso debido a los siguientes factores:

- El impacto de la computadora, no solamente como una fuente de datos sino también como una herramienta para comunicaciones, y la demanda de otros servicios digitales como el teletipo.

- El empleo de las comunicaciones digitales ofrece flexibilidad y compatibilidad al adoptar un formato digital común, que hace posible para un sistema de transmisión, sustentar muchas fuentes de información diferentes.

- La confiabilidad mejorada debido al uso de sistemas de comunicaciones digitales.

- La disponibilidad de canales de banda ancha que son proporcionados por satélites geoestacionarios, fibras ópticas, y cables coaxiales.

- La creciente disponibilidad de tecnología electrónica de estado sólido integrada, que ha hecho posible incrementar la complejidad de los sistemas digitales.

El desarrollo de las comunicaciones digitales continuará y se pronostican grandes logros en la segunda mitad del siglo, por lo pronto la rápida evolución que se ha experimentado en lo que va del siglo quedará escrito para la historia

Notas históricas[4]

A continuación se presentan algunas notas históricas sobre comunicaciones haciendo énfasis en telefonía digital y sus conceptos relacionados.

- Código binario

Los orígenes del código binario, básico en la operación de las comunicaciones digitales tiene su origen en los primeros trabajos de Francis Bacon, al comienzo del siglo diecisiete. Bacon utilizó combinaciones de dos letras distintas, agrupando símbolos de cinco en cinco para representar veinticuatro letras del alfabeto, como un medio para codificar mensajes secretos. El alfabeto de dos letras construido por Bacon fue publicado en 1605. Posteriormente, en 1641 John Wilkins publicó un libro en el que el alfabeto de Bacon no solamente se explicó sino también se expandió, empleando alfabetos de tres y cinco letras. Quedo demostrado por primera vez que se necesitan más elementos para reducir el promedio de la longitud de una palabra código.

En 1703, Gottfried W. Leibnitz publicó un artículo en la Real Academia de Ciencias de Paris, titulada "Explicación de la aritmética binaria". El texto de su lectura se publicó en las revistas de la Academia en 1705. Leibnitz utilizó los números cero y uno para su código binario. Parece ser que el código binario de Leibnitz se desarrolló independientemente de los de Bacon y Wilkins.

- Telefonía

En 1874 se concibió el teléfono en Brantford, Ontario, por Alejandro Graham Bell. El teléfono se hizo una realidad práctica, al permitir la transmisión de voz en tiempo real mediante su codificación eléctrica y su regeneración sonora. La primera versión del teléfono fue tosca y limitada, habilitando a la gente para hacer llamadas a corta distancia solamente. La calidad y el rango del teléfono fué enriquecido por la invención y desarro-

llo del micrófono de carbón y la bobina de inducción durante el periodo de 1877-1890. El uso de bobinas de carga en las líneas de transmisión hizo posible que la telefonía abarcara distancias de hasta tres mil kilómetros. Sin embargo fué hasta 1913 que la telefonía transcontinental se hiciera posible gracias al empleo de diez nuevos amplificadores electrónicos, a base de tubos al vacío.

Cuando el servicio tenía pocos años, se fomentó el interés por desarrollarlo. En 1897, Strowger desarrollo el conmutador automático paso a paso que lleva su nombre. De todos los conmutadores electromecánicos, este fué el más popular y el de mayor uso en esos años.

La invención del transistor en 1948 sugirió la aplicación de la electrónica a la conmutación. El motivo fué mejorar la confiabilidad, incrementar la capacidad, y reducir el costo. La primer llamada a través de un sistema computarizado fue hecha en marzo de 1958 en los laboratorios Bell. El primer teléfono comercial con conmutación digital comenzó a dar servicio en Morris, Illinois, en junio de 1960.

En 1937 Alec Reeves inventó la modulación por codificación de pulsos para codificar digitalmente señales de voz. La técnica se desarrolló durante la segunda guerra mundial para hacer posible la encriptación de señales de voz. En ese entonces se utilizó un sistema de veinticuatro canales. En 1945 De Loraine inventó el multiplexaje por división de tiempo, tanto aquí como en la modulación por codificación de pulsos, su primera aplicación fué integrada en redes de telefonía militares.

El concepto de transmisión y conmutación digital integrada fué propuesto después de 1950, casi al mismo tiempo que el primer sistema de conmutación digital. El avance hacia la red digital integrada (RDI) fue motivado por las expectativas de continuar reduciendo los costos en componentes e interfaces. El primer paso para extender la red digital a la telefonía fué dado en 1974.

El reto para el futuro es encontrar el objetivo de abarcar un rango de servicios (datos, voz y video), que se han venido proponiendo. Se presume que resulta económico convertir las señales generadas por esos servicios a un formato digital para su transmisión digital. El papel que las nuevas técnicas de conmutación jugarán en una red digital de servicios integrados (RDSI) estará determinado por el grado de integración deseado.

1.2 Objetivo de la transmisión digital

Debido al reciente desarrollo de la tecnología y técnicas digitales, actualmente se utiliza la transmisión digital con mayor frecuencia. En la figura 1.2 se aprecia claramente porque la transmisión digital se prefiere sobre la transmisión analógica[6].

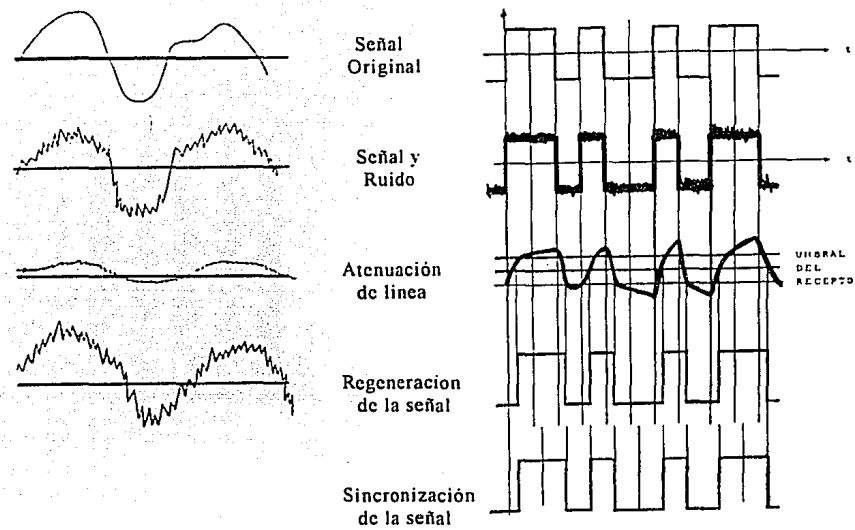


Figura 1.2 Transmisión analógica vs transmisión digital[6]

Cuando una señal analógica se ve modificada por el ruido, es difícil regenerar la señal original, pero éste problema es diferente en el caso de las señales digitales. Debido a que una señal digital tiene un número finito de niveles, es fácil regenerar la señal que se envió originalmente, sin pérdida de información, o cualquier otro tipo de inconveniente como sería el cruce de señales, distorsión por fase, etc.; los cuales son típicos en la transmisión analógica. Estos problemas se ven incrementados si tomamos en cuenta la longitud de las líneas de transmisión. El ruido se incrementa continuamente en proporción a la longitud de la línea de transmisión.

La calidad de la transmisión digital es casi independiente de la longitud de las líneas de transmisión, ya que existe la posibilidad de regenerar completamente la señal enviada sin ruido. Removiendo en toda regeneración de la señal original los efectos del entrecruzamiento de señales, retardos y distorsión, el resultado es que se puede establecer que la calidad de la señal transmitida es la misma al final de la trayectoria de transmisión que como había sido en la etapa inicial.

1.3 Descripción de una red telefónica digital

En este apartado se describe una central telefónica digital en forma general, comentando su propósito y características distintivas, posteriormente se describen brevemente las características funcionales y la arquitectura del tipo de central telefónica digital (sistema 1240 de ITT) en que se aplica el tema que nos ocupa en esta tesis.

Características Generales[5][6]

Una central telefónica digital utiliza un **sistema de conmutación** completamente digital, con el propósito de utilizarse casi siempre en redes de telefonía pública y redes de transmisión de datos.

El término completamente digital implica que la información transmitida tiene formato digital al pasar a través de toda la red de conmutación. Para tal efecto, las señales analógicas enviadas a la central por un abonado con línea analógica, son convertidas en señales digitales a la entrada de la central, y convertidas nuevamente de señales digitales a señales analógicas a la salida de la central.

Una central digital aumenta sus capacidades de servicio para justificarse por encima de centrales que utilizan tecnologías obsoletas como los conmutadores a base de relevadores y de alambres de cobre utilizados como medios de transmisión en enlaces entre central y central. Por ejemplo, la transmisión de varios canales analógicos de voz entre central y central se realizaba a través de varios hilos de transmisión, uno para cada canal, formándose así lo que se conoce como una *troncal*; ahora con la tecnología digital se puede tener hasta treinta troncales en un mismo hilo de transmisión físico.

Con estas mejoras en la tecnología para centrales telefónicas, éstas pueden atender un rango de 100 a 100000 líneas de abonado o equivalentemente hasta 60000 troncales. Por otro lado, una central digital tiene la flexibilidad de ser configurada al momento de instalarse como: una *Central local*, que realiza directamente la conexión entre abonados pertenecientes a la misma área urbana; *Central Tandem* que maneja tráfico de tránsito originado o terminado en centrales locales, subordinadas a ella. La *Central Toll o de cobro* es una central automática que cursa tráfico de tránsito interurbano originado o terminado en centrales subordinadas a ella las cuales pueden ser centrales locales u otros Toll. En la práctica existen centrales que ejecutan funciones de central local y central de tránsito ya sea Tandem y/o Toll. Cabe mencionar que también la capacidad de tráfico se ve mejorada ya que puede manejar hasta 750000 intentos de llamada en horas pico.

El caso particular de la central digital 1240 de ITT tiene las siguientes características funcionales:

- Transmisión Digital

Económica : en cuanto a equipo resulta más económica que la transmisión analógica.

Confiabilidad : con la utilización de equipo estático se eliminan los efectos indeseables del equipo mecánico y por tanto los sistemas digitales se vuelven más confiables.

Calidad mejorada : un equipo de transmisión digital tiene mayor inmunidad al ruido que un equipo analógico cuyo costo sea comparable.

Integración de la transmisión de voz y datos: la conmutación digital es el paso decisivo hacia la Red Digital de Servicios Integrados (RDSI). Esto es, la conmutación digital es una condición necesaria para tener una RDSI.

- Modularidad de la Construcción

Tanto el equipo físico *(Hardware)* como el soporte lógico *(Software)* están divididos en módulos funcionales y en diferentes niveles. Las interfaces entre estos están claramente definidos y estandarizados. La introducción de este concepto de construcción por bloques le da al sistema una estructura bastante flexible y hace posible lo siguiente:

Introducir nueva tecnología y nuevos servicios sin hacer cambios a la arquitectura del sistema

; extender una misma central sin reconfigurar el equipo existente, y con equipo basado en nuevas tecnologías

; dar seguridad al sistema en cuanto a fallas dado que los módulos están duplicados.

- Control Distribuido

Se emplea generalmente este término cuando se utilizan muchos procesadores, físicamente distintos, en un sistema grande. Existen tres formas de distribución de control:

Distribución funcional: en la que cada procesador realiza una función diferente. Aparte de que se pueda emplear la duplicación o alguna otra forma de redundancia, cada procesador es una entidad funcional única y tiene un paquete de programas también único. Si un procesador falla y no es reemplazado, la función que desempeña en el sistema se pierde, a menos que se realice una reconfiguración funcional total.

Distribución jerárquica: en la que cada procesador tiene su papel específico en una relación jerárquica con el resto de los procesadores. Un diseño cuidadoso de la central podría permitir reutilizar ciertos tipos de procesadores, pero la tendencia es que cada procesador y su paquete de programas sean únicos. En el caso de que falle un procesador, si no se le reemplaza, todas las porciones del sistema situadas por debajo en la jerarquía quedarán fuera de servicio.

Distribución espacial: en que a cada procesador se le asigna una parte de la operación total de la central (por ejemplo, todas las funciones que corresponden a un grupo concreto de abonados). Un gran número de estos procesadores y paquetes de programas serán idénticos, con la única

diferencia de los datos que reciben para el proceso. Si uno de estos procesadores falla y no se le reemplaza, solamente quedará fuera de servicio una parte de las funciones de la central (por ejemplo, servicio a un pequeño grupo de abonados), o sufrirá una reducción aparente en su dimensión.

Se ha hecho esta distinción porque todos los sistemas modernos tienden a usar una o más de las formas de distribución explicadas, con objeto de aprovechar la tecnología de los microprocesadores. La forma (o la combinación de formas) de distribución utilizada determinará la eficiencia en el cumplimiento de los objetivos del sistema. **La estructura física del control de la central digital en cuestión es de distribución espacial, sin control central; sin embargo, la estructura de la programación utiliza los tres tipos de distribución ya expuestos[5].**

- Distribución del control en la central ITT 1240

La construcción de un sistema de distribución espacial presenta algunos problemas de diseño. Uno de estos problemas es el de intercomunicación de los procesadores. Las redes clásicas de procesadores, estructuras de *buses* y configuraciones de multiproceso difícilmente manejarían la cantidad de comunicaciones entre procesadores que necesita una central digital. Se precisa, por tanto, un nuevo planteamiento. La central telefónica digital resuelve el problema al utilizar la propia **red digital de conmutación** para tales comunicaciones. Puesto que la mayoría de los procesadores son ya responsables de establecer y mantener los caminos de conversación para sus grupos de abonados, pueden aplicar esa misma capacidad a establecer caminos para comunicación entre procesadores a través de la red, sobre canales de 64 *kbits*. Sin embargo, todo ello impone algunas exigencias a la red de conmutación :

- Control directo por el usuario final: la red debe ser controlada por el equipo conectado en sus terminaciones, con objeto de que la comunicación entre los procesadores no precise de otros elementos de control.

- Virtualmente sin bloqueo: cualquier bloqueo apreciable haría más lenta la comunicación entre procesadores y con ello aumentaría el tiempo de respuesta total del sistema. Este requisito debe cumplirse aunque no exista un mecanismo global de búsqueda de caminos, ya que la centralización de tal mecanismo impondría restricciones al sistema en cuanto a capacidad de proceso o lo haría sensible a las fallas.

- Establecimiento rápido de los caminos: en los mensajes entre procesadores, el camino debe establecerse en uno o dos milisegundos. Tiempos de establecimiento mayores afectarían desfavorablemente a los tiempos de respuesta de la central, ya que el retardo de los mensajes se haría apreciable dentro del proceso de una llamada.

- Crecimiento gradual: puesto que un sistema de control distribuido especialmente puede ofrecer un crecimiento gradual a lo largo de una extensa gama de tamaños, la red de conmutación debe apoyar esta

característica con la facilidad de crecer gradualmente en dicha gama (en cuanto a líneas y a tráfico).

La red digital de conmutación de la central telefónica digital 1240 de ITT satisface los cuatro criterios anteriores y por tanto es el núcleo de la arquitectura distribuida de la central. Básicamente, la central digital consiste en un conjunto de módulos o bloques funcionales, cada uno con su propio microprocesador, conectados a la red digital de conmutación, la cual sirve tanto para las comunicaciones de conversación como para las comunicaciones entre procesadores, según indica la figura 1.3.

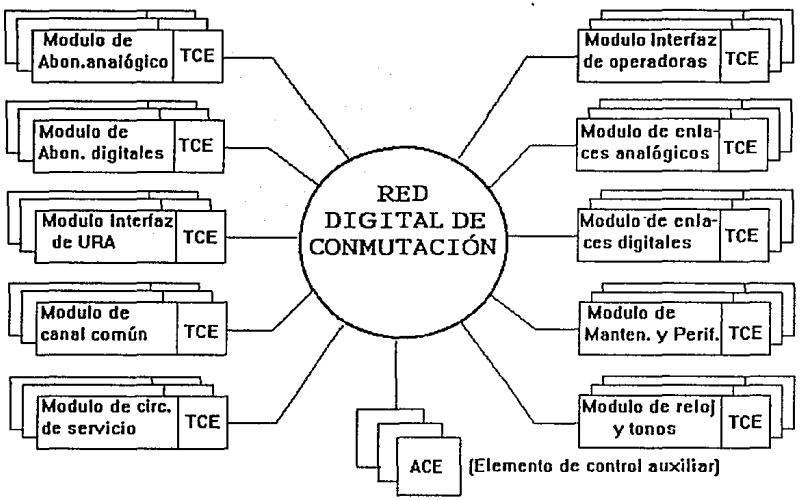


Figura 1.3 Arquitectura de la Central Digital 1240 ITT[6]

Cada módulo posee recursos de circuitos y de capacidad de proceso para un grupo de terminales (60 abonados o 30 troncales), o bien recursos en circuitos y en programas para atender una función de la central (circuitos de servicio, periféricos del procesador, generador de reloj y tonos). En el primer caso al módulo se le conoce como **Elemento de Control Terminal** o **TCE (Terminal Control Element)**; en el segundo caso se le conoce como **Elemento Auxiliar de Control** o **ACE (Auxiliary Control Element).**

En la figura 1.4 se muestra la arquitectura de control distribuido del *software* de la central digital. Como se observa, el sistema utiliza las tres formas de distribución en la estructura del soporte lógico o *software*; la división indicada enfatiza las funciones más importantes del sistema.

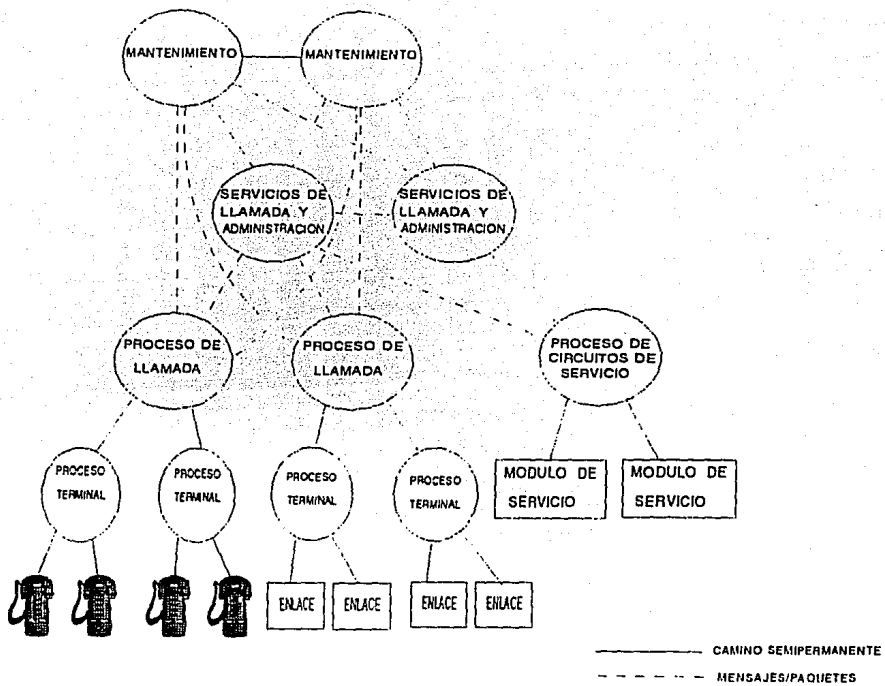


Figura 1.4. Disposición del *software* distribuido en la central digital 1240 de ITT[6].

Proceso de llamadas: esta función presenta una distribución espacial en grupos de 480 líneas o 240 troncales. Dentro de cada grupo se usa una forma de distribución jerárquica: los procesadores de los elementos de control terminal (**TCE**: *Terminal Control Element*) de cada módulo controlan los circuitos y las funciones de exploración y distribución, mientras que las funciones de manejo de llamadas y lógica de señalización se llevan a cabo por un procesador independiente, llamado elemento de control auxiliar (**ACE**: *Auxiliary Control Element*).

Servicios de soporte: están distribuidos funcionalmente. Los servicios de soporte incluyen los proporcionados por circuitos de servicio, periféricos del procesador, módulos de reloj y tonos, que también son **TCE's**, y los servicios de base de datos de la central realizados por parejas de **ACE**; cada pareja conserva los datos correspondientes a un servicio de central (manejo de grupos de troncales, traducciones de enrutamiento, etc.).

Administración: estas funciones (actualización de datos de la central, medidas, tarificación, etc.) están distribuidas por jerarquías, existiendo un par de **ACE** en el nivel superior. Los niveles más bajos están ocupados por los **ACE** de control de llamadas y por los **TCE**. Ambos tipos de elementos de control utilizan los módulos de periféricos de procesador para las operaciones de entrada/salida e interactuan con el resto de los procesadores para recopilar datos o modificar la base de datos de la central.

Mantenimiento: ésta función tiene distribución jerárquica. Los módulos de periféricos de procesador están en el extremo superior de la jerarquía y todos los demás procesadores en un mismo nivel más bajo.

Descripción del equipo físico[5]

La **Red Digital de Conmutación** se construye a partir de un puerto especial de conmutación bidireccional, realizado con tecnología LSI (LSI: *Large Scale Integration*) de diseño específico. El Elemento Digital de Conmutación o **DSE** (**DSE** : *Digital Switch Element*), unidad básica funcional de la red, está compuesto por dieciséis puertos de conmutación.

La interfaz entre un módulo y la red digital de conmutación es un par de enlaces PCM (PCM: *Pulse Code Modulation*) de 32 canales; mientras se mantenga esta interfaz, cualquier evolución en la tecnología o en la arquitectura que se introduzca en un módulo no afectará al resto del sistema. Esta característica es la que permite a la central digital aprovechar los avances tecnológicos.

La capacidad que tienen los elementos de control de comunicarse a través de la red digital de conmutación facilita un crecimiento gradual en un amplio margen de tamaños.

La figura 1.3 muestra cómo la red digital de conmutación constituye el núcleo de la central, ya que no sólo se usa para las conexiones de voz y datos entre terminales de la red, sino también para la comunicación de los elementos de control terminal entre sí y de estos con los ACE. La red se compone de un sólo tipo de elemento digital de conmutación, cada elemento de conmutación **SE** (SE: *Switch Element*) también llamado "multipuerto", realizado en una sola placa de circuito impreso ó **PBA** (**PBA**: *Printed Board Assembly*), conmuta espacialmente entre puertos y temporalmente entre canales PCM, permitiendo a cada uno de los 512 canales entrantes conectarse a cualquiera de los 512 canales salientes. En el apartado 1.6 se explica en detalle el **SE**.

Concepto de **módulo terminal (TCE)**: se vió en la fig. 1.3 que la central digital está constituida por diversos tipos de módulos terminales conectados a través de la red digital de conmutación. La figura 1.5 ilustra la estructura general de un módulo o elemento de control terminal **TCE**, consistente en dos partes: la terminal o *cluster* asociado y el elemento de control, **CE**.

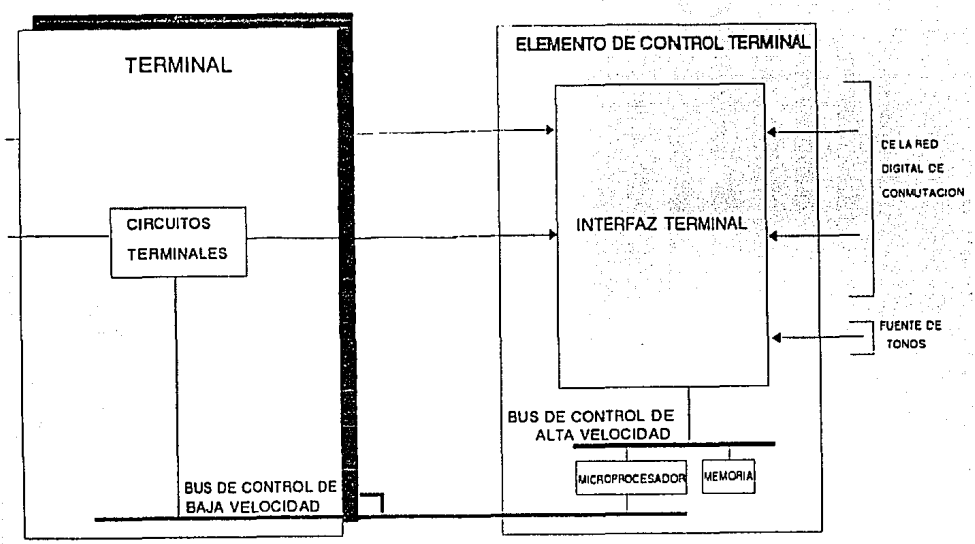


Figura 1.5 Esquema de un módulo terminal (TCE) subdividido entre sus circuitos terminales y el elemento de control(CE)[5]

Cada tipo de módulo realiza una tarea diferente: manejo de líneas analógicas, troncales analógicas, líneas digitales, troncales digitales, etc.

El corazón del **CE** es el microprocesador con su memoria asociada o **TCPB** (TCPB: *Terminal Control Processor Board*), que realiza las funciones repetitivas de entrada/salida. La otra parte del **CE** es la interfaz terminal o **TERI** (TERI: *Terminal Interface*) que constituye la interfaz de transmisión entre los circuitos terminales (*cluster*) y la red digital de conmutación y también entre el microprocesador y la red. Una o dos vías de transmisión PCM de 32 canales transportan información desde el cluster a la interfaz terminal, dando acceso a la misma, sin bloqueo, a un máximo de 60 líneas de abonados. Dos vías similares conectan la interfaz terminal con la red de conmutación. Además, la interfaz terminal lleva a cabo la función de intercambio de intervalos de tiempo.

Todas las operaciones del **TCE** están controladas por el microprocesador, el cual intercambia mensajes con los microprocesadores de otros elementos de control a través del propio **TERI** y la red digital de conmutación, para realizar las funciones de manejo de llamadas, mantenimiento y administración, eliminando con ello la necesidad de medios especializados de comunicación. Las funciones no repetitivas del proceso de llamadas y otras tareas diversas se realizan en un segundo tipo de elementos de control, los **ACE** (**ACE**: *Auxiliary Control Element*), que no controlan ninguna terminal o *cluster*. Así, un **ACE** se constituye solamente de un microprocesador con su memoria asociada y un **TERI**. Al igual que en los módulos terminales de la central, su interfaz con la red se realiza mediante canales PCM.

Tipos de Módulos: las funciones telefónicas más importantes se agrupan en siete tipos de módulos, a saber: módulos de troncales analógicas, módulos de abonados analógicos, módulos de circuitos de servicio, módulos de troncales digitales, módulos de reloj y tonos, módulos de canal común (para señalización de canal común), y módulos de interfaz de operadoras. Los periféricos de comunicación hombre máquina y de terminal computadora se controlan con un módulo de periféricos separado, que incorpora un disco Winchester, unidad de cinta magnética, impresora , unidades de pantalla y panel de alarmas.

Por confiabilidad, el módulo de periféricos de procesador y el módulo de reloj y tonos están duplicados. La configuración se pone de modo que, inclusive en caso de falla, se mantenga un grado de servicio aceptable.

- Red Digital de conmutación

El problema del diseño de una red de conmutación capaz de extenderse en forma continua, para cubrir una gama de tamaños amplia, se ha solucionado con el desarrollo del multi-puerto, elemento digital que realiza conmutación temporal y espacial, incorporando funciones de búsqueda de caminos y de control. Las características más importantes de la red digital de conmutación son su capacidad de aceptar órdenes que lleguen por una entrada cualquiera y de establecer un camino en *símplex* a través de la red, eligiendo una trayectoria con retardo mínimo. La estructura numérica de la red determina el camino de interconexión exclusivamente a partir de las direcciones de origen y destino de la red. La segunda mitad de una conexión *dúplex* se establecerá por el elemento de control destino, hacia atrás y hasta el elemento de control origen.

- Elementos de control

Los elementos de control **TCE** (controlan terminales) y **ACE** (no controlan terminales) tienen, en parte, una composición idéntica que incluye un microprocesador de 16 bits, la memoria asociada a éste, y una interfaz terminal o **TERI** (*Terminal Interface*).

- Microprocesador y memoria asociada

El microprocesador también llamado **TCPB** está basado en un circuito *Intel* 8086, con capacidad de direccionamiento de 1 *Megabyte* de memoria de estado sólido, organizada en páginas de 256 *kilobytes*. Montado en una **PBA** que incluye un *bus* de control de baja velocidad con dirección y datos multiplexados, y un *bus* de alta velocidad con dirección y datos en paralelo.La memoria va montada en una placa separada, se trata de una memoria comercial **RAM** (*Random Access Memory*) dinámica de 64 *kbit* y tiene una capacidad de corrección de errores individuales y detección de errores dobles. La lógica de corrección y detección de errores está

localizada en la tarjeta del microprocesador. La tarjeta o **PBA** del microprocesador posee diversas posibilidades adicionales de detección de fallas, incluyendo un temporizador de vigilancia (*watchdog*) y lógica de protección de escritura en la memoria. Las partes críticas del temporizador de vigilancia (*watchdog*) del elemento de control y del programa cargador de la memoria están en memoria **ROM** (ROM: *Read Only Memory*) en la tarjeta del procesador.

- **Interfaces de Control**

Existen dos tipos de interfaces de control entre los microprocesadores de los **TCE**, de los **ACE**, y su entorno: un *bus* de control de baja velocidad y un *bus* de control de alta velocidad para comunicarse con la interfaz terminal.

Con el *bus* de baja velocidad el microprocesador controla los circuitos terminales de varios módulos, incluyendo los de abonados analógicos, troncales analógicas, troncales digitales, y periféricos del procesador. Este *bus* se constituye de 13 líneas, en las que se multiplexan tanto las direcciones como los datos.

Por otro lado, el *bus* de alta velocidad del microprocesador controla al **TERI** y a los periféricos de alta velocidad, como unidades de control de discos, también cuenta con acceso directo a su memoria principal.

- **Interfaz Terminal (TERI)**

Como lo muestra la figura 1.6, ésta es la interfaz entre las líneas de abonados, troncales, etc.(terminales) y la red digital de conmutación. Cada TERI tiene dos puertos para conectar el conjunto de terminales asociados, dos puertos para la conexión de la red, un puerto para la comunicación con el microprocesador y su memoria, y un puerto para la introducción de tonos. Los puertos de las terminales, los de conexión con la red, y el de conexión a la fuente de tonos tienen muchos de los atributos de los puertos de la red de conmutación, con las siguientes diferencias: la orden para establecer una trayectoria a través del **TERI** viene del puerto del procesador y no de la red de conmutación; en el **TERI** un canal entrante se puede conectar a todos los canales salientes que sea necesario, lo cual no ocurre en el puerto de la **DSN** (*Digital Switching Network* o Red Digital de Conmutación), y que permite que cualquier fuente de tonos o de voz que llega por un canal entrante se conecte a cualquier terminal de conversación.

El puerto de conexión del procesador al **TERI** está asociado con unas zonas de almacenamiento de mensajes entrantes y salientes, que pueden ser cargadas o descargadas por el microprocesador para aceptar mensajes de llegada, enviar mensajes a otros procesadores, u órdenes a los puertos de la interfaz terminal.

Los segmentos de memoria RAM del interfaz terminal almacenan los mensajes tanto entrantes como salientes. El microprocesador también pue-

de controlar los puertos de conexión de los terminales, los de conexión a la red de conmutación, y el puerto de tonos mediante el acceso de escritura/lectura a los registros de control del puerto, a través del *bus* de alta velocidad.

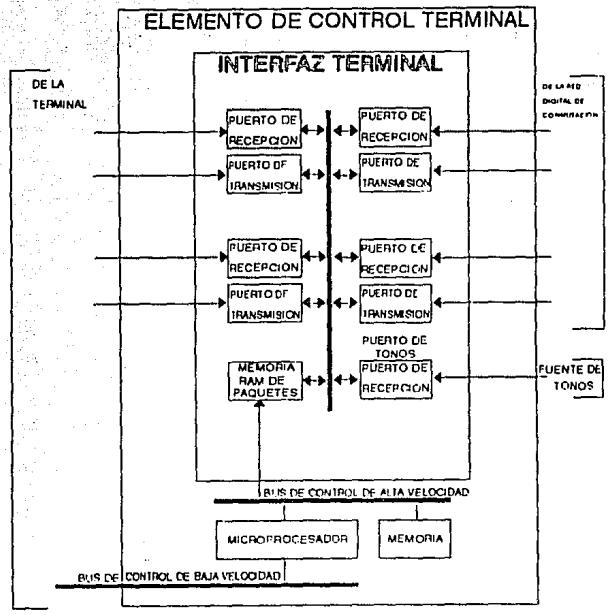


Figura 1.6 Interfaz Terminal (TERI), mostrando sus conexiones hacia el cluster, la red de conmutación y el procesador a cuyo módulo pertenece[5].

Cada puerto del interfaz terminal contiene dos circuitos LSI de diseño específico: el puerto de recepción y el puerto de transmisión. Todos los puertos están interconectados por un *bus* **TDM** (*Time Division Multiplexing*) indicado con la linea vertical gruesa de la figura 1.6, similar al que une los puertos de conmutación en el multipuerto. El **TERI** contiene cuatro puertos de transmisión y cinco de recepción en tecnología LSI, además de algunos componentes lógicos comerciales de integración a media escala; todo ello montado en un solo PBA.

1.4 Sistemas PCM (*Pulse Code Modulation*)

Muestreo[5][6]

En los sistemas de transmisión de audio, una frecuencia audible se transporta sobre un medio continuo denominado portadora.

El problema original radicaba en el hecho de que si en realidad era necesario transmitir la señal completa o si sólo era necesaria la transmisión del valor de la señal a intervalos regulares, es decir, transmitir muestras de la señal con una cierta frecuencia llamada de muestreo.

El teorema de Nyquist muestra que existe una relación entre la frecuencia de muestreo (f_s) y la frecuencia máxima (f_{max}) ocurrida en la banda de la señal de voz de la siguiente forma:

$$f_s \geq 2.f_{max.} \qquad (1.1)$$

Si $f_s < f_{max}$, entonces será imposible reconstruir la señal original adecuadamente.

Debido a que las señales de conversación tiene un ancho de banda de 300 - 3400 Hz , la señal analógica a puede escribirse como :

$$a = A.\cos(wt),\ (w=2\pi f) \qquad (1.2)$$

$$\text{con } 2\pi\ 300\ \text{Hz} < w < 2\pi\ 3400\ \text{Hz}$$

La señal muestreadora es una señal de pulsos, en este caso, con una frecuencia de 8 KHz y puede escribirse en forma de su serie de Fourier :

$$A_m = A_0 + A_1\cos(w_S.t) + A_2\cos(2w_St) + A_3\cos(3w_St) + \ldots \qquad (1.3)$$

Donde: A_0 es la componente de DC de la señal, y

$$w_S = 2pf_s = 2p8000,\ f_s = \text{frecuencia de muestreo}$$

Ya que la señal muestreada puede considerarse como el producto de la señal de muestreo A_m (con valor 0 ó 1) y la señal original, efectuado por un circuito muestreador, éste producto se puede escribir como :

$$a.A_m = A_0.A\cos(wt) + A_1A\cos(wt).\cos(w_St) + A_2A\cos(wt).\cos(2w_St) + A_3A\cos(wt).\cos(3w_St) + \ldots \qquad (1.4)$$

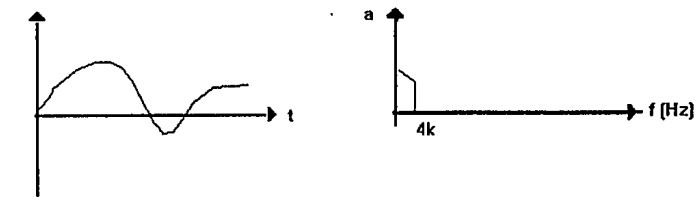
Como cosxcosy = 1/2[cos(x+y)+cos(x-y)], (1.4) equivale a:

$$
\begin{aligned}
a.A_m = {} & A_0 A\cos(wt) + \\
& + A.A1.\cos[(w_S+w)t]/2 + A.A1.\cos[(w_S-w)t]/2 \\
& + A.A2.\cos[(2w_S+w)t]/2 + A.A2.\cos[(2w_S-w)t]/2 \\
& + A.A3.\cos[(3w_S+w)t]/2 + A.A3.\cos[(3w_S-w)t]/2 \\
& + \ldots
\end{aligned}
\qquad (1.5)
$$

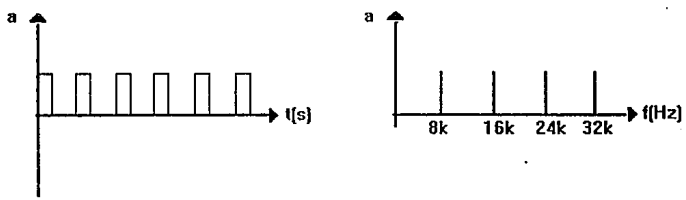
Banda superior para cada componente de a

Banda inferior para cada componente de a

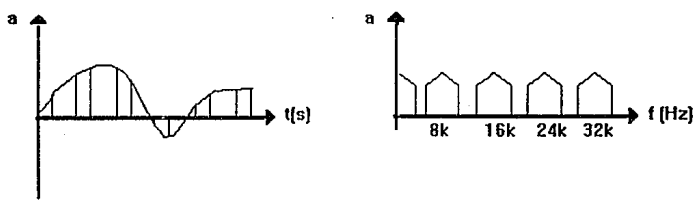
En la figura 1.7 se indica la señal a muestrear (a), la señal muestreadora (b), y la señal muestreada (c).



a. SEÑAL ANALOGICA



b. SEÑAL DE MUESTREO



c. SEÑAL MUESTREADA

Figura 1.7 Señales de entrada y salida del circuito de muestreo con sus espectros de frecuencia correspondientes[6]

Cuantización[6]

La cuantización está representando la amplitud de una muestra de la señal mediante la magnitud del nivel discreto más cercano al valor de la muestra.

Con el objeto de hacer factible una transmisión digital el valor de cada muestra tendría que ser representada mediante un código. Ya que los elementos del código son una cantidad finita, los valores de la amplitud de la muestra serán redondeados a los valores más cercanos posibles de representar mediante un elemento del código. Existen dos métodos principales de cuantización : lineal y no lineal. La figura 1.8 muestra el esquema de cuantización de una señal ya muestreada y sus niveles de cuantización.

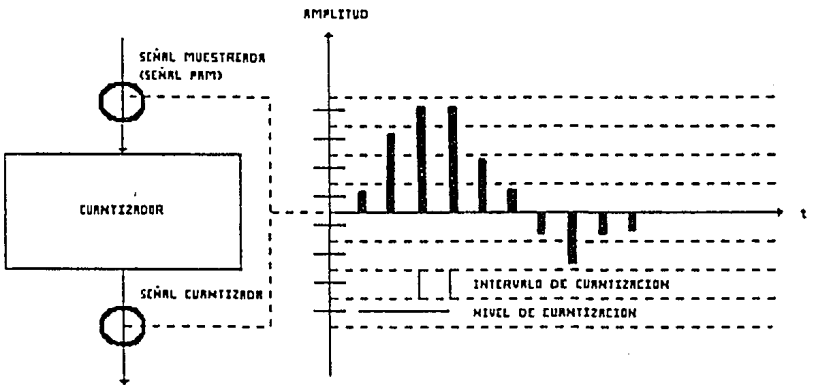


Figura 1.8 Cuantización[6]

- Cuantización lineal

El rango total de valores de voltaje que pueden manejarse se subdividen en un número de subrangos iguales de voltaje. Cada subrango corresponde a una combinación de *bits* dada (elemento del código). Al momento de la codificación, cualquier voltaje situado entre las cotas superior e inferior de un subrango se codifica con la combinación correspondiente.

Al momento de la decodificación, un código se representa mediante un voltaje correspondiente a la mitad del subrango. El resultado es que una

cierta cantidad de ruido se agrega a la señal original. Este es el llamado **ruido de cuantización.**

El ruido de cuantización es de hecho la diferencia entre la señal cuantizada decodificada y la señal original. El ruido en el caso de cuantización lineal, tiene un cierto nivel dependiendo de los subrangos, ver figura 1.9.

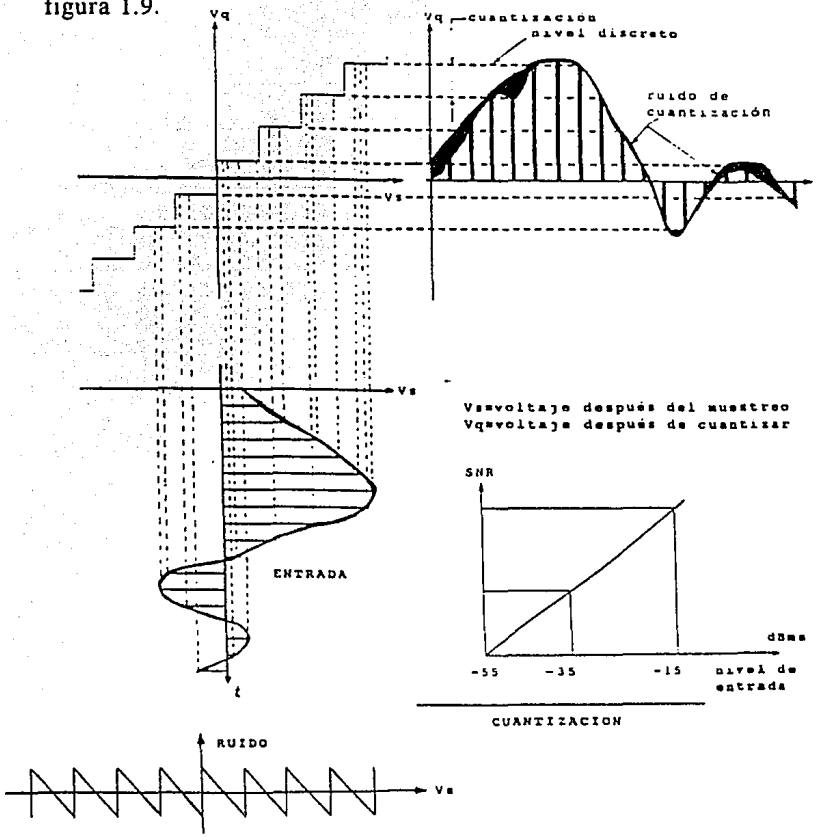


Figura 1.9 Cuantización lineal[6]

- Cuantización no lineal

Una cuantización lineal da como resultado una mala relación señal a ruido. Con el objeto de obtener una relación señal a ruido que presente un valor constante para cualquier nivel de señal se desarrolló otro método de cuantización. Para éste método los niveles de cuantización tienen que ser

seleccionados con una escala logarítmica, lo cual significa utilizar una cuantización no lineal, fig. 1.10. Es posible que los niveles de ruido más altos sean permitidos en una señal muestreada con más alto nivel que para señales con bajos niveles.

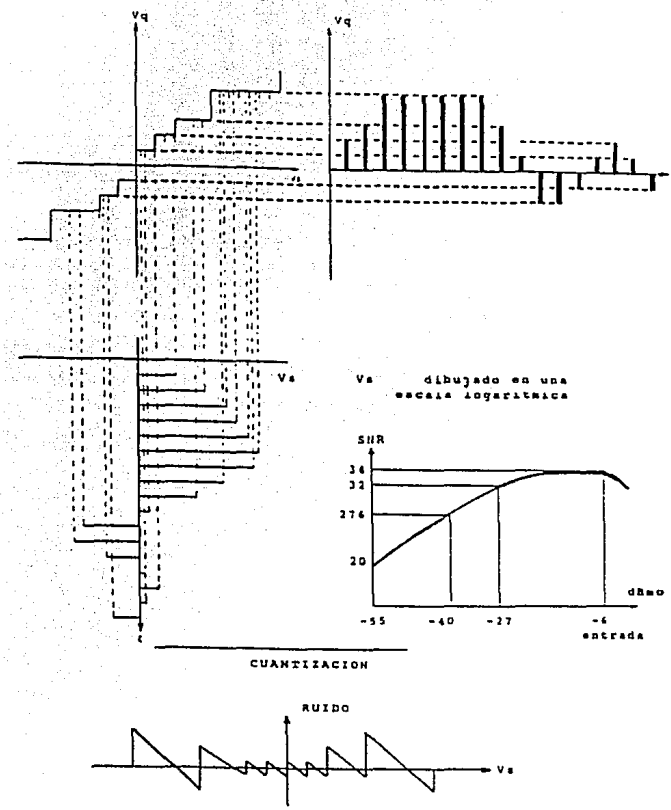


Figura 1.10 Cuantización no lineal[6]

Codificación[6]

Después de haber sido cuantizada una muestra, se le limita a tomar un valor de entre 256 niveles discretos. La mitad de estos valores son para codificar muestras positivas, mientras que las otras codificaran muestras con signo negativo. Existen 256 niveles, por lo tanto sólo se necesitan 8 *bits* para codificar todos los niveles. Cada combinación de ocho *bits* corresponde a un nivel. Para seleccionar que combinación debe corresponder con que nivel, se tienen diferentes posibilidades.

Existen muchos códigos diferentes, los más útiles son: el código natural y el código simétrico.

El código natural utiliza la combinación (00000000) del código para representar el valor más negativo de las muestras de la señal, y la combinación (11111111) del código lo utiliza para representar el mayor valor, o el más positivo, de las muestras.

En el caso del código simétrico siete de los ocho *bits* se utilizan para representar la magnitud de la muestra y el octavo *bit* representa su signo.

Decodificación e Integración[6]

El tren de pulsos que llegan al receptor debe transformarse en una serie de muestras, como las que tenemos después de la cuantización en el lado transmisor.

Si suponemos que se utiliza el código simétrico en el lado transmisor, como resultado, cada vez que se envíen 8 bits, estos deberán dividirse en dos partes, una para el signo y la otra para la magnitud, para poder representar la magnitud de la muestra.

En la fig. 1.11 se da un esquema general de la modulación y demodulación de pulsos codificados.

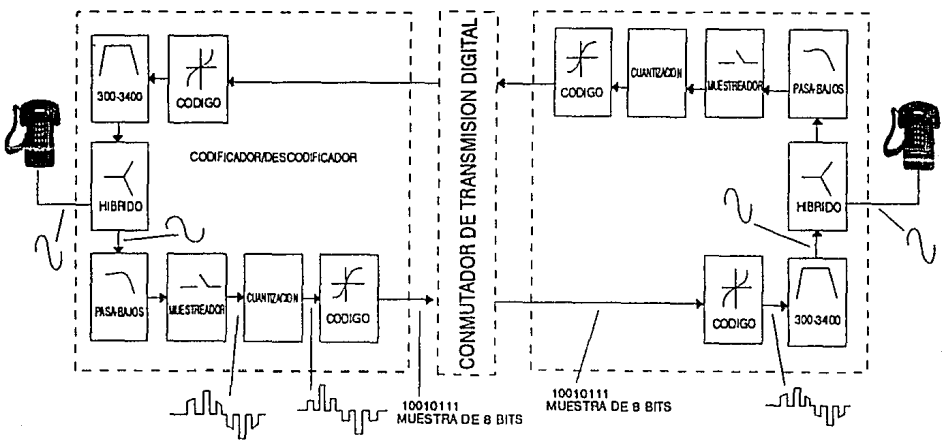


Figura 1.11 Modulación/demodulación de pulsos codificados

Después de la decodificación se recuperan las series originales de muestras cuantizadas. Estas series de muestras serán transformadas a una señal analógica continua mediante el procesamiento de estas a través de un filtro paso bajas con un ancho de banda máximo de 4 KHz (integración), removiendo de esta forma todas las altas frecuencias originadas durante el muestreo y consecuentemente obteniendo una señal de salida abajo de los 4 KHz, más aún, obteniendo la señal originalmente enviada.

- Codificación de bits

Con el objeto de transmitir señales digitales, se desarrollaron los códigos de transmisión. Los códigos de transmisión óptimos deben de cumplir con las siguientes características :

* La componente promedio de DC introducida en la línea debe de ser de 0 volts DC, ya que esto amplía enormemente la capacidad de cobertura del sistema, tanto en ancho de banda del canal como en la eliminación de la pérdida de potencia de la señal transmitida. Situación que se vuelve crítica para transmisiones a larga distancia (más de 1 Km).

* El *bit* del reloj debe ser enviado al receptor, ya sea mediante el empleo de un separador de distribución de reloj o mediante transiciones frecuentes en la señal.

A continuación se describirán algunos códigos más utilizados (Fig. 1.12):

NRZ :

No retorno a cero. En este código de transmisión un "0" puede ser representado mediante un voltaje negativo y un "1" mediante un voltaje positivo. Normalmente se emplea durante la transmisión de datos en distancias cortas, en un ambiente con un sistema de distribución de reloj separado. Un ejemplo de este caso es la transmisión de información en el interior de una central.

AMI :

Inversión de marcas alternadas. Ya que el código NRZ no es adecuado para la transmisión de datos en distancias grandes, el código AMI ha sido desarrollado para cubrir la transmisión a largas distancias. El objetivo de este código es reducir el nivel de DC en la línea a un valor de 0 volts. En este código un "0" será representado por 0 volts y un "1" por una tensión positiva alternada con una negativa.

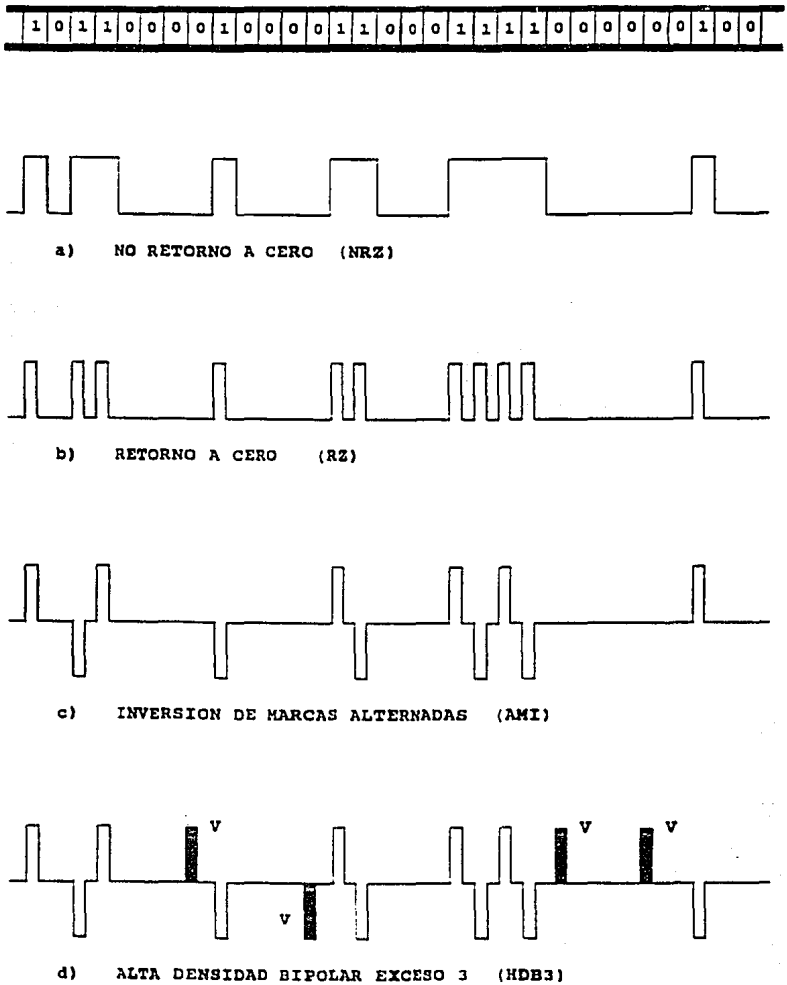


Figura 1.12 Códigos de transmisión[6]

Mediante la inversión de la polaridad de marcas consecutivas, el promedio de la componente de DC en la línea cae al valor de 0 volts. Por resultado, este código es adecuado para cubrir distancias muy grandes. No obstante lo anterior existe un problema que no ha sido aún resuelto: este código no transmite el reloj del sistema. El receptor debe reconocer y extraer la tasa de reloj de entrada mediante la exploración de la transición en el tren de pulsos de entrada. Si se tiene una serie de *bits* que son iguales a "0", el receptor no podrá reconocer la señal del reloj, debido a que se tiene un nivel continuo de DC (0 volts) en la línea. Con el objeto de resolver este problema otro código ha sido desarrollado y se denomina: *High Density Bipolar Excess 3 (HDB3).*

HDB3 :

High Density Bipolar Excess 3. Este código inserta un pulso denominado de violación cuando más de tres "0" llegan sucesivamente. El lado transmisor inserta los pulsos, los cuales pueden detectarse en el receptor. El receptor borrará estos pulsos nuevamente. Este código es por resultado un código de muy buena calidad, el cual requiere cierta circuitería *(Hardware)* adicional, responsable de la inserción y extracción de los pulsos de violación.

Multiplexaje[6]

Objetivo del multiplexaje: Los costos ahorrados mediante el empleo de las técnicas de multiplexaje son enormes, resultando un incremento rápido del uso de estos sistemas.

En la década de los 30's con la introducción de las comunicaciones de larga distancia se encontró otro problema de transmisión. Aunque los amplificadores (repetidores analógicos) habilitaron a los sistemas de transmisión, para poder compensar la atenuación característica en los sistemas de transmisión de larga distancia, la transmisión fue de muy baja calidad. Esta deficiencia de calidad fue causada por el "ruido de transmisión" introducido en los sistemas. Los cambios en la amplitud de la señal son amplificados en cada etapa hasta que la señal se convierte a una señal totalmente inaudible. Fue entonces cuando las grandes compañías dedicadas a las telecomunicaciones comenzaron a hacer investigaciones sobre un nuevo sistema de transmisión, el cual podría eliminar el "ruido de transmisión".

Mediante la combinación de una nueva técnica de multiplexaje, llamada multiplexaje por división de tiempo y el uso de la transmisión digital, nace la telefonía digital.

- **Multiplexaje por División de Tiempo (TDM)**

En los sistemas de transmisión de señales en banda base (audio) moduladas, las frecuencias de tales señales se transportan en forma continua sobre una portadora. Una pregunta interesante para éste método de transmisión es: ¿ se necesita transmitir la señal completa o es suficiente con transmitir muestras de la señal a intervalos regulares de tiempo?. La respuesta esta dada por el teorema de Nyquist con el que se prueba que es posible recuperar en el receptor la señal a transmitir, si se transmiten muestras de la misma a intervalos regulares de tiempo. Un Sistema de Multiplexaje por División de Tiempo (**TDM**) es un sistema de transmisión en el que varias señales de banda base se multiplexan sobre una misma portadora, esto es, viajan sobre la misma portadora asignando un canal a cada señal y a su vez se asigna una ranura de tiempo específica a cada canal; en la ranura de tiempo asignada al canal se transmite una muestra de la señal asignada a tal canal.

La aplicación de un sistema **TDM** a señales analógicas de banda base implica la conversión de dichas señales a un formato digital como se describe a continuación:

1) Se realiza un muestreo de dichas señales a intervalos regulares de tiempo como lo indica la figura 1.13 para cuatro señales analógicas, la generación de estas muestras se logra con un Modulador por Amplitud de Pulsos (PAM).

2) Las muestras obtenidas son cuantizadas empleando normalmente 8 *bits* para representar el valor de cada muestra, y a continuación los ocho *bits* se asignan a un canal que será multiplexado para su transmisión.

Las cadenas de *bits* que llegan al lado receptor y que representan los valores de las muestras de las señales analógicas se demultiplexan para obtener las muestras de dichas señales; básicamente se procede como si gue:

1) Hágase un análisis de la alineación de los bits tomando como referencia el canal "0" de la cadena recibida, el cual contiene un patrón específico que indica la sincronización del reloj al lado receptor.

2) A continuación se alojan las muestras de 8 *bits*, que representan una señal, en *buffers* de memoria individuales, y se convierten las muestras de 8 *bits* a las muestras de la señal analógica original. Finalmente se obtiene la señal analógica original a partir de las muestras recuperadas lo que se logra con un demodulador PAM.

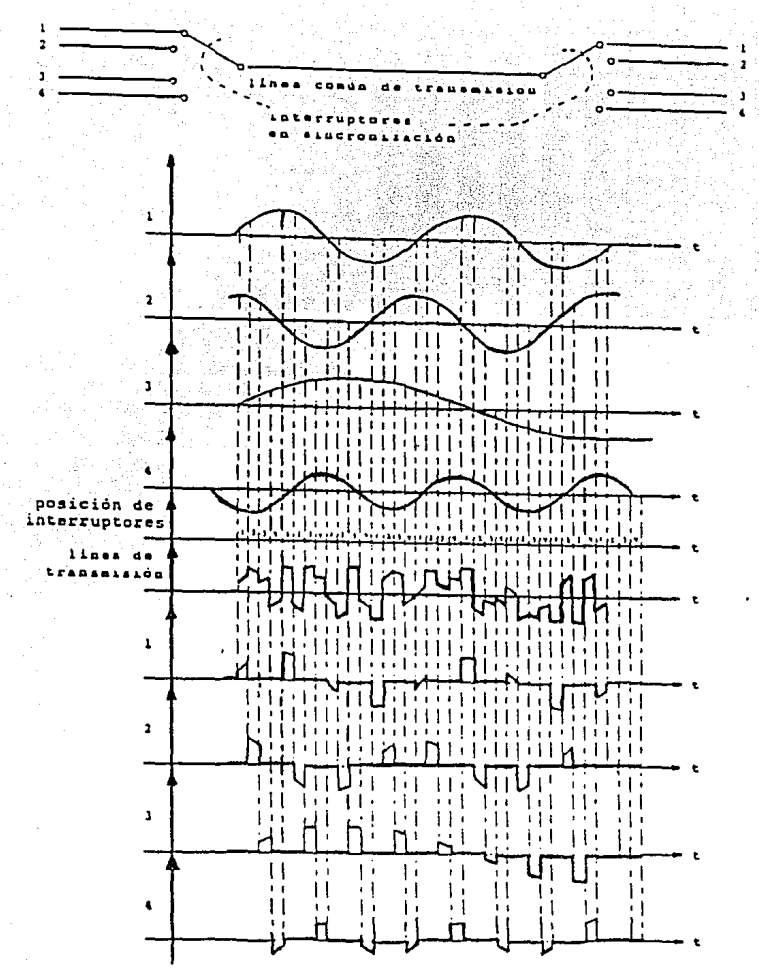


Figura 1.13 Señalización PAM multiplexada en el tiempo[6]

Estructura de una trama de canales[6]

Mediante el empleo de un sistema TDM, un conjunto de canales se monta en una sola portadora. Cada canal se representa como un tren de muestras, donde cada una de ellas se representa en forma de un código

digital. En Europa ha sido aceptado y estandarizado por la CCITT un sistema TDM de 32 canales. Cada canal transporta 8 *bits* a la vez. Esta estructura se llama **trama** y tiene 256 *bits*. Una llamada telefónica se asigna a un canal sobre tal trama, lo que significa que 8 *bits* pueden enviarse en cada trama y como una señal de un abonado se muestrea cada 125 μs (f_s=8000 Hz), un abonado puede enviar 8 *bits* cada 125 ls. Una trama toma exactamente 125 μs, y la duración de un canal tiene 125 μs/32 =3.906 μs.

La razón de transmisión de un tren PCM es de 256 *bits* en 125 μs, lo cual corresponde a 2.048 *Mbits*/seg, ver fig. 1.14.

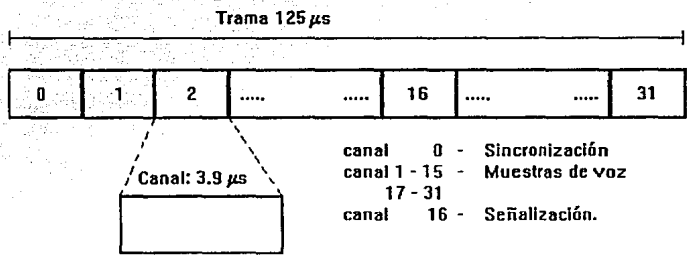


Figura 1.14 Formato PCM[6]

De un total de 32 canales únicamente 30 pueden usarse para la transmisión de voz y datos, razón por la cual en algunas ocasiones esta estructura se denomina de 30 canales. Cada canal usado para transmisión de datos y voz contiene 8 *bits*, de los cuales, el primero indica el signo y el resto la magnitud de la señal. En cada trama el mismo número de canal será entregado al mismo abonado.

Sistemas PCM de orden alto[6]

Mediante el empleo de un sistema PCM de 32 canales, se pueden transportar 30 canales de voz sobre una portadora. Con un ancho de banda de portadora suficientemente grande se puede transportar un número mayor de canales sobre la misma mediante el empleo de un PCM de orden alto, lo que significa hacer una reducción del tiempo necesario para el envío de un pulso, pudiéndose de esta manera enviar más pulsos en una misma unidad de tiempo. (Fig. 1.15).

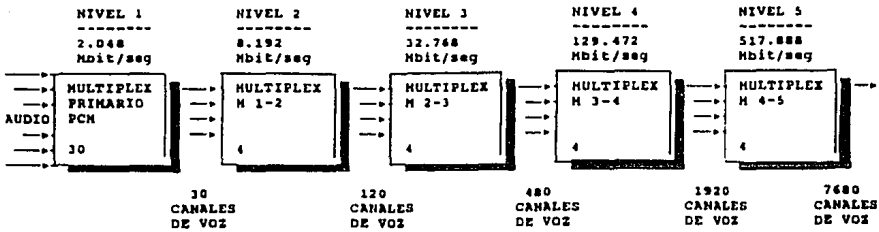


Figura 1.15 PCM de orden alto[6]

1.5 Sincronización

En una línea de transmisión digital se transporta una gran cantidad de *bits* a razón de 2*Mbits*/s. De tal manera que para lograr la interpretación de la información en el lado receptor, se requerirá de la sincronización.

Sincronización de bit[6]

En el lado receptor la información llega a una tasa de 2048 kHz. Si los datos se exploran entre la transición de 2 *bits*, la información muy probablemente será errónea. La información debe leerse a la mitad del *bit*. Por tanto, el primer problema que se presenta es cómo acertar a la mitad de los *bits* muestra, o sea, cómo sincronizar los *bits*. La sincronización de *bit* se puede lograr de dos maneras:

a) Enviando el *bit* de reloj a todos los puntos donde se recibe una comunicación por PCM. Esto requerirá de una conexión especial de reloj (distribución de reloj); y, b) Variar la señal de información lo suficiente con el objeto de lograr sincronización *bit* a *bit*.

Sincronización de trama[6]

Después de recibir los *bits* de información, estos tienen que ser ensamblados en *bytes* que representan una muestra de voz o datos de un enlace de comunicación particular. Aquí se necesita una referencia que identifique un cierto punto dentro de la trama, se tiene así la sincronización de trama, obtenida mediante la repetición de un patrón fijo en el canal 0 de cada trama.

Si un receptor no está sincronizado (p.ej. al encenderlo), éste intentará inicialmente lograr la sincronización de *bit*. Después de la sincronización de *bit* el receptor comenzará a buscar la muestra fija la cual se encuentra en el canal 0. En tramas con estructura de 32 canales, el canal 0 utiliza 8 *bits* para la alineación de la misma. El canal 0 utiliza el patrón **A0011011** para las tramas pares, mientras que las impares utilizan el patrón **B1CDEFGH**, donde sólo el segundo *bit* (**1**) es fijo y los otros *bits* (**B,C,D,E,F,G**, y **H**) pueden tomar diferentes valores. Los *bits* 2 al 8 del canal 0 en las tramas pares que contienen la alineación principal, y sólo el segundo *bit* del canal 0 alinea las tramas impares (ver fig. 1.16).

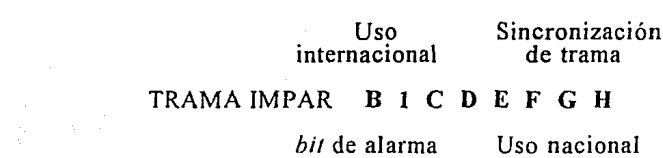


Figura 1.16 Sincronización de trama (A,B,...=id. de *bit*)[6]

En cada trama el primer *bit* del canal 0 se reserva para uso internacional (*bits* A y B). Estos *bits* toman el valor "1" cuando no se usan. Los *bits* D, E, F, G, y H de cada trama impar son reservados para uso nacional y no tienen significado en un enlace internacional, en este caso su valor será igual a 1. El *bit* C de la trama impar se utiliza como *bit* de alarma del enlace; éste será puesto a 1 para informar a la central origen y la central destino esta pérdida de alineación.

Sincronización de red[6]

Cuando se conmuta información por PCM, por acuerdo se establece que el tren de pulsos entrante debe estar en sincronía de *bit* con el reloj local del punto de conmutación. El tren de pulsos de entrada puede originarse en diferentes centrales donde estos han sido generados con los respectivos relojes locales de cada central a las que pertenecen.

Los diferentes casos que pueden considerarse son :

a) *Redes asíncronas*

En este tipo de redes los relojes de las centrales son independientes y el promedio de la tasa de transmisión del tren de pulsos entrante puede ser mayor o menor que aquel impuesto por el reloj local de una central determinada. Esto significa que en el proceso de adaptación de los *bits* de entrada al temporizador de la central, la información se perderá o repetirá ocasionalmente.

Para los relojes más estables y exactos, el de menor velocidad servirá para evaluar una razón de errores. Las redes que usan relojes muy exactos y estables son llamadas pleosíncronas (cercano a la sincronía).

b) *Redes síncronas* (maestro-esclavo)

En una red síncrona maestro-esclavo, un reloj es el maestro y el resto tendrá que sincronizarse a éste, mediante la regeneración del reloj, extraído del tren de pulsos entrante de la central maestra. En este tipo de redes la tasa de *bit* promedio es la misma para cada central, aunque evidentemente la fase del tren de pulsos entrante puede ser diferente debido a los retrasos de transmisión. En el proceso de adaptación de los *bits* a la fase del reloj local la información no se ve alterada. (Fig. 1.17).

c) *Redes mutuamente síncronas*

Aquí no existe un reloj maestro. Todos los relojes se sincronizan sobre el valor medio de todas las tasas de *bits* de entrada. De esta manera la red adopta también una tasa de *bit* uniforme. La mayoría de las centrales utilizan esta técnica ya que es la más eficiente. (Fig. 1.18).

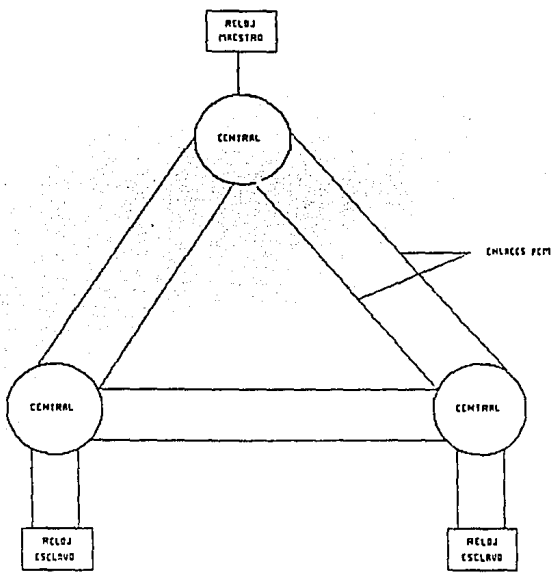


Figura 1.17 Red síncrona maestro-esclavo[6]

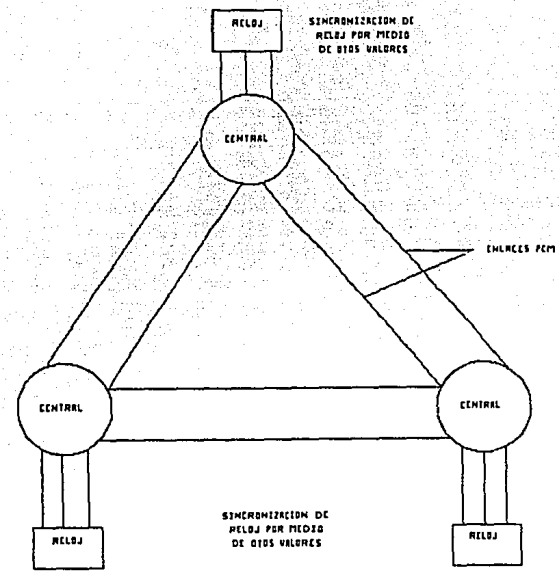


Figura 1.18 Red síncrona mutua[6]

1.6 Conmutación Digital

Evolución de la red telefónica[6]

- Red telefónica analógica

Hasta 1970 la red telefónica usada en todos los países consistía de conmutadores analógicos interconectados por sistemas de transmisión analógicos. (Fig. 1.19).

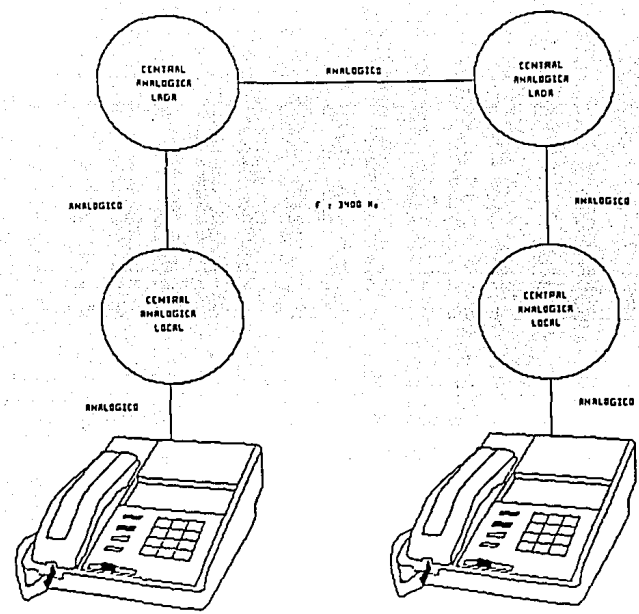


Figura 1.19 Red telefónica analógica[6]

- Red de telefonía híbrida

Desde la introducción de las redes de larga distancia en los años 30, surgío un problema nuevo dentro de la red telefónica: el ruido de transmisión reducía la calidad de la transmisión a niveles inaceptables. Como resultado las compañías telefónicas comenzaron a buscar sistemas que transmitieran datos sin introducir ruido.

Una solución a este problema se encontró en la década de los 70's: la introducción de la transmisión digital dentro de las redes telefónicas analógicas.

El primer sistema comercial de este tipo estuvo disponible a finales de los 60's y la red telefónica analógica fue lentamente convirtiéndose en una red híbrida, ver fig. 1.20.

Esta red telefónica híbrida consiste de :

a. Sistema de transmisión digital basada en un formato de trama PCM de 32 canales.

b. Puntos de conmutación analógica, conectando conversaciones mediante sistemas analógicos.

c. Convertidores de analógico a digital a nivel de troncales en cada central.

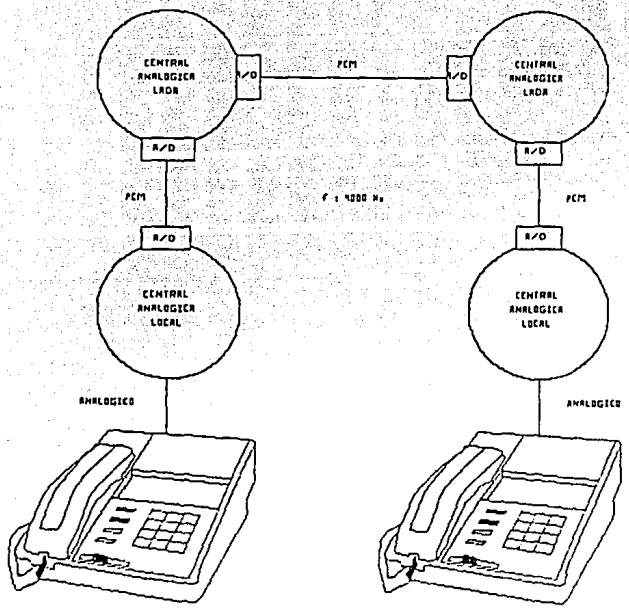


Figura 1.20 Red telefónica híbrida[6]

- Red digital integrada

Desde que los sistemas de transmisión digital han eliminado completamente el ruido de transmisión, la calidad de la señal se ha mejorado mucho.

Los sistemas de transmisión digital por sí mismos fueron una solución efectiva en costos. Sin embargo, el costo total de redes híbridas fue muy alto (convertidores A/D en cada punto de conmutación). De manera que para mejorar la efectividad en costos, las administraciones telefónicas han tratado de eliminar los convertidores intermediarios A/D. Por lo cual se desarrolló el conmutador digital basado en un sistema TDM, un ejemplo de éste es el conmutador 1240.

Con la comercialización de las centrales digitales en los años ochentas, el camino quedó abierto a la introducción de nuevas redes telefónicas, caracterizadas por un costo rentable de ellas mismas, las cuales consisten de Centrales digitales y Sistemas de transmisión digital.

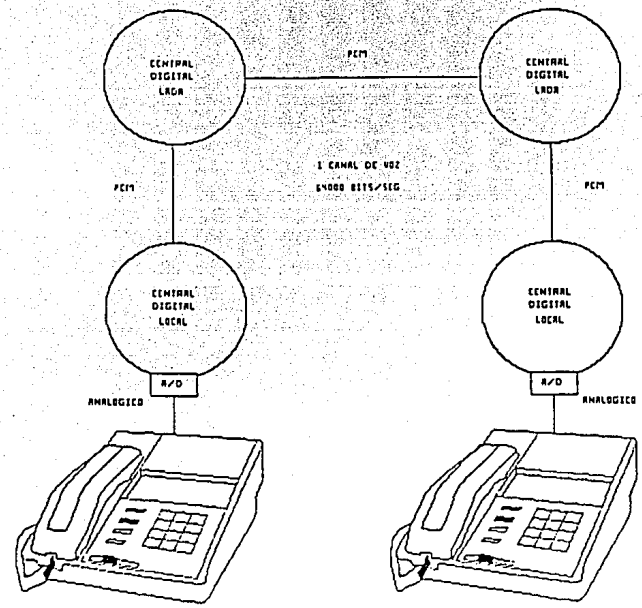


Figura 1.21 Red digital integrada[6]

Este tipo de red se denomina **red digital integrada** y es la solución más efectiva en costos para las redes telefónicas digitales actuales y serán, por tanto, introducidas en todo el mundo en las próximas décadas, ver fig. 1.21.

Principios de la conmutación digital[6]

- Conmutaciones de los trenes de pulsos PCM

En una central analógica se establecen conexiones físicas entre dos abonados utilizando algunos métodos de conmutación. Durante la fase de establecimiento de una llamada, los interruptores correctos son operados estableciéndose de esta forma la conexión entre dos aparatos telefónicos.

Durante la conversación los interruptores permanecen inmóviles y la trayectoria permanece fija hasta que alguno de los abonados libera la llamada.

En una central digital, un conmutador recibe un tren de pulsos correspondiente a una trama PCM de 32 canales y transmite un PCM a los puertos físicos. Una trama de pulsos PCM contiene 30 canales de usuarios diferentes como se vió en la Fig 1.14.

Un **Elemento de Conmutación Digital (DSE**: *Digital Switch Element* o *Switch Element*) tiene que dividir la información que llega a un puerto en 30 direcciones diferentes. Cada unidad de conmutación o **DSE** consiste de (Fig. 1.22):

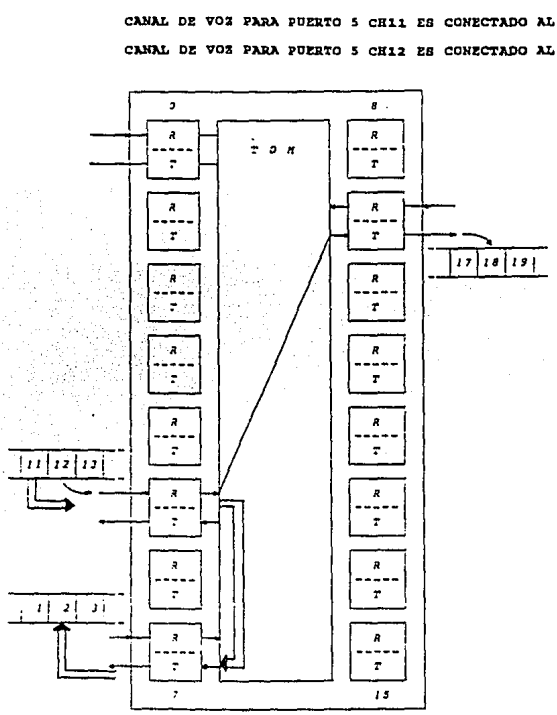


Figura 1.22 Elemento de Conmutación Digital DSE[6]

a. Un conjunto de puertos bidireccionales (16 en la mayoría de los casos). Cada puerto se construye con un puerto transmisor **T** y un puerto receptor **R**. El puerto transmisor transmite un PCM de 32 canales y el puerto receptor recibe un PCM de 32 canales.

b. Los puertos se interconectan por medio de un sistema de *bus* en paralelo (*bus* **TDM**) indicado con el rectángulo al centro de la figura 1.22.

En la fig. 1.23 se representa una estructura más detallada de un puerto transmisor y un puerto transmisor.

Al puerto receptor llegan grupos consecutivos de *bits* donde cada grupo forma un canal. Todos los *bits* dentro de un canal serán enviados al mismo destino, de tal modo que durante el tiempo de la conmutación, ésta se realizará canal por canal y no *bit* por *bit*.

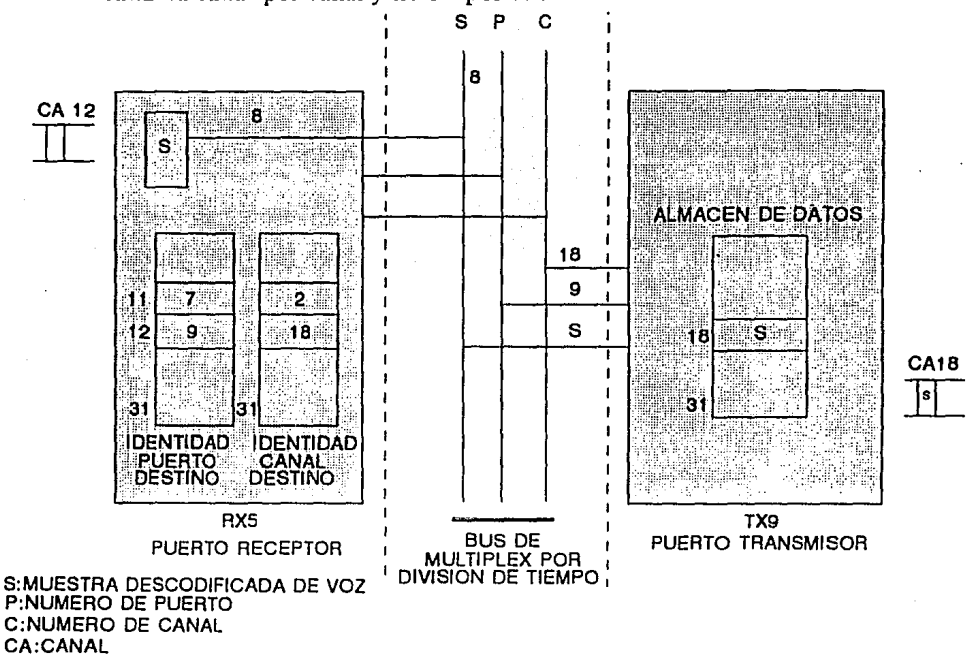


Figura 1.23 Puerto Transmisor (Tx) y Puerto Receptor (Rx) del Elemento de Conmutación SE[6]

El **puerto receptor** consiste de las siguientes partes :

- Un circuito *latch*, recuadro S en la figura 1.23, donde los *bits* muestra contenidos en un canal se almacenan hasta que estos se reciban completamente;

- Un *buffer*, rectángulos inferiores dentro del puerto receptor indicado a la izquierda en la figura 1.23, que contiene las identidades del puerto de destino y canal de destino para todos los canales que se estén usando.

El **puerto transmisor** envía los canales PCM. Este contiene un *buffer* para almacenar las muestras de datos, recuadro S inferior indicado dentro del puerto transmisor al derecha de la figura 1.23, hasta que estos puedan enviarse.

1.7 Señalización

Señalización en un ambiente MFC analógico[6]

- Señalización de línea y de registro

Para poder llevar a cabo las funciones de la conmutación, deben de ser tomadas ciertas acciones especiales. Se necesitará de una comunicación entre el subscriptor y la central y también entre las centrales, con el objeto de instruir a las centrales de como desarrollar sus funciones de conmutación. Esta fase de comunicación se llama **señalización**.

El primer tipo de comunicación se llama **señalización de línea** y corresponde a aquella información que se intercambian los circuitos de línea (o circuitos de troncales).

El objetivo principal de esta fase de señalización de línea es la de informar a la siguiente central de la intención de iniciar una llamada, o de liberar una llamada (ver fig. 1.24).

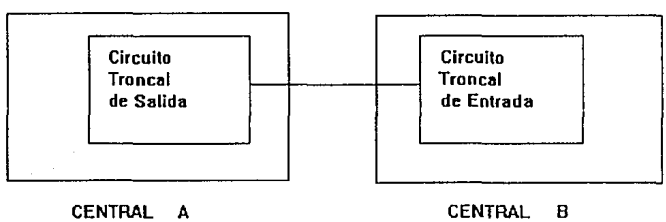


Figura 1.24 Señalización de línea[6]

Al tiempo que se establece una llamada, se toma una troncal; después de la toma, la información de selección será pasada entre un registro en la central de salida y el registro en la central de llegada.

La fase de intercambio de información es denominada **señalización de registro**, y hará uso de la misma troncal (trayectoria de conversación) la cual será utilizada posteriormente para conversación telefónica, ver fig. 1.25.

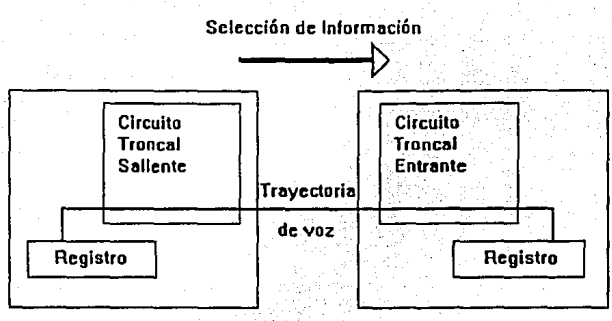


Figura 1.25 Señalización de registro[6]

En telefonía, se utiliza únicamente un escenario de señalización para establecer una llamada. Ver figura 1.26.

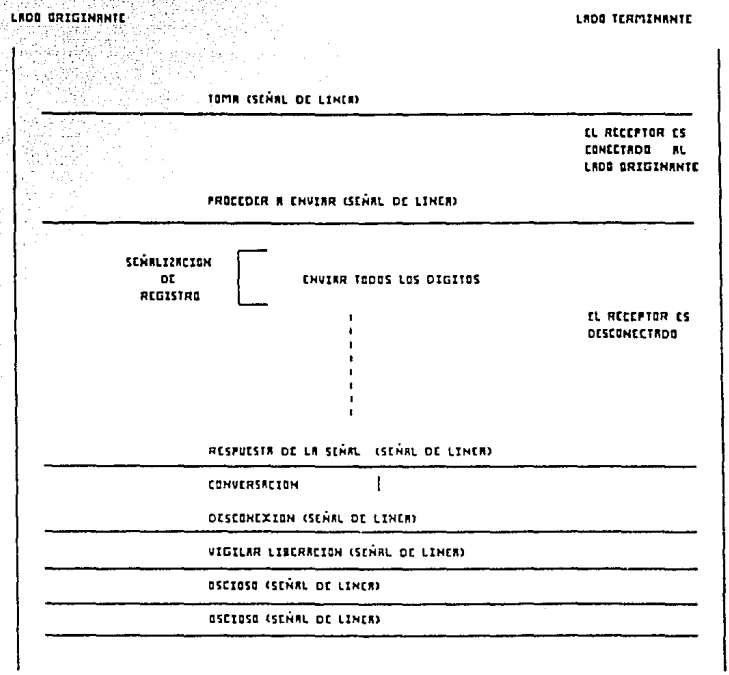


Figura 1.26 Escenario de señalización básica[6]

Para la liberación de una llamada, existen dos escenarios. Si el subscriptor llamante cuelga primero, una señal de borrado hacia adelante inicia la liberación de todos los dispositivos. Si el abonado llamado cuelga primero, una señal de borrado hacia atrás será enviada resultando en la activación de un temporizador. Cuando este temporizador expira o el abonado llamante cuelga, la llamada será liberada.

Señalización de Registro de MFC (*Multifrecuency Code*)[6]

Las señales de registro se utilizan para enviar información de direcciones y también transferir información de direcciones concernientes a las partes llamante y llamado. La señalización de registro de MFC consiste de la transmisión y la recepción de información sobre los canales de voz, mediante varias combinaciones de dos y sólo dos de N=6 frecuencias diferentes, como máximo, en la banda de voz, figs. 1.27.a, 1.27.b, y 1.27.c. Cada combinación de dos frecuencias formará una señal y cada señal representa información de una dirección. Los receptores de multifrecuencia detectan las señales y transfieren la información al equipo de control, el cual establece la conexión a través de los conmutadores.

FRECUENCIA EN Hz						
SEÑALES ADELANTE	1380	1500	1620	1740	1860	1980
SEÑALES ATRAS	1140	1020	900	780	660	540
1	X	X				
2	X		X			
3		X	X			
4	X			X		
5		X		X		
6			X	X		
7	X				X	
8		X			X	
9			X		X	
10				X	X	
11	X					X
12		X				X
13			X			X
14				X		X
15					X	X

Figura 1.27.a Tabla de códigos de multifrecuencia[6]

SEÑAL	GRUPO I (EN RESPUESTA A A1)	GRUPO II (EN RESPUESTA A A3/A5)		
	SIGNIFICADO	SIGNIFICADO		
1	DIGITO 1	ABONADO NORMAL	USO	NACIONAL
2	2	LLAMADA PRIORITARIA		
3	3	EQUIPO MANTENIM. LLAMADA		
4	4	DISPONIBLE		
5	5	LLAMAR OPERADOR		
6	6	LLAMAR TRANSMISION DE DATOS		
7	7	LLAMAR ABONADO	USO	INTERNACIONAL
8	8			
9	9	DISPONIBLE		
10	0	LLAMAR OPERADOR		
11	ACCESO A OPERADOR	DISPONIBLE		
12	ACCESO DE RETARDO A OPERADOR	DISPONIBLE		
13	ACCESO A EQUIPO DE MANTENIMIENTO	DISPONIBLE		
14	INSERTAR	DISPONIBLE		
	(UNICAMENTE PARA LINEAS MUY LARGAS)			
15	FIN DE PULSACION	DISPONIBLE		

Figura 1.27.b Tabla de señalización MFC hacia adelante (alto rango)[6]

SEÑAL	GRUPO A	GRUPO B	
	SIGNIFICADO	SIGNIFICADO	
1	ENVIAR SIGUIENTE DIGITO (N+1)	DISPONIBLE	
2	ENVIAR ULTIMO DIGITO (N-1)	ABONADO TRANSFERIDO	
3	CAMBIO A LA SERIE B	ABONADO OCUPADO	
4	CONGESTION	CONGESTION	
5	ENVIO NATURAL DE ORIGINADOR	ABONADO NO ASIGNADO/CONMUTACION DE NO ALAMBRADO	
6	CONDICIONES ESTABLECER VOZ	ABONADO LIBRE CON CARGO DE LLAMADA	
7	ENVIAR DIGITO (N-2)	ABONADO LIBRE SIN CARGO DE LLAMADA	
8	ENVIAR DIGITO (N-3)	LINEA DE ABONADO FUERA DE ORDEN	
9	DISPONIBLE	DISPONIBLE	USO NACIONAL
10	DISPONIBLE	DISPONIBLE	
11	ENVIAR INDICADOR DE TRANSITO INTER.	DISPONIBLE	USO INTERNACIONAL
12	ENVIAR LENGUAJE O DISCRIMINACION DE DIGITO	DISPONIBLE	
13	ENVIAR CODIGO CENTRAL DE TRANS. INTER.	DISPONIBLE	
14	DISPONIBLE (SUPRESOR DE ECO)	DISPONIBLE	
15	CONGESTION	DISPONIBLE	

Figura 1.27.c Tabla de señalización MFC hacia atrás (bajo rango)[6]

Señalización en un ambiente MFC digital[8]

- Señalización de línea

Principios de la señalización de canal asociado CAS

Con el objeto de transferir las condiciones de línea, se introdujo un nuevo sistema de señalización: **señalización por canal asociado** (*CAS: Channel Asociated Signaling*).

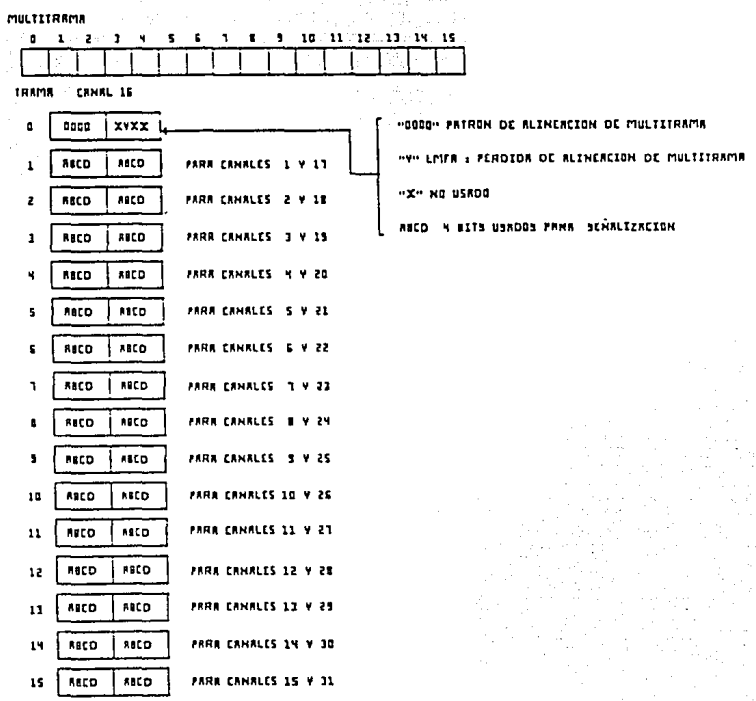


Figura 1.29 Señalización por canal asociado[6]

Este sistema codifica las señales de línea antiguas en señales digitales (*bits*) y transmite estos *bits* a través de posiciones fijas en un tren de pulsos. Estas posiciones dependerán de la estructura de la trama que sea utilizada (24 canales ó 32 canales). Los *bits* reflejarán un cierto estado de una línea. De esta manera el sistema estará habilitado para manejar todas las clases de señalización de línea que se genera durante los cambios de estado en la línea. Consecuentemente, CAS está restringida a la señalización de línea. Habiendo una sola excepción: señalización de registros decádicos que se envía mediante cambios de potencial en líneas de voz.

No todas las clases de señalización de registro que son hechas mediante señales voz-frecuencia son transmitidas usando CAS. La señalización CAS será transmitida a través de un canal de usuario normal.

Alineación de multitrama

Como no es posible enviar la información de la señalización de línea en cada trama, esta será enviada en varias tramas. Para asignar a estas tramas un número, se necesita una estructura de multitrama. El número de tramas en una estructura de multitrama depende de si se trata de tramas de 32 o 24 canales, en caso de una trama de 32 canales se utiliza una multitrama de 16 tramas. En tal caso son necesarios los números de trama sólo para el canal 16, el cual se usa por CAS como para la alineación de multitrama. La muestra para alineación de multitrama se envía en el canal 16 de la trama 0. Todas las otras tramas (1 a 15) usan el canal 16 para enviar información de señalización. El canal 16 de la trama 0 tiene, como cualquier otro canal, 8 *bits*. Sólo los 4 primeros *bits* del canal 16 serán utilizados por la muestra de alineación de multitrama. La muestra utilizada tiene el siguiente formato: "0000" , el quinto, séptimo, y octavo *bit* son *bits* cuyos valores pueden ser cualquiera (*don't care*). El sexto *bit* indica si el enlace correspondiente recibe el patrón de alineamiento de multitrama o no. En caso de una estructura trama de 24 canales se utilizan multitramas de 12 tramas, numeradas de la 1 a la 12. Cada trama tiene un *bit* que puede ser usado para la alineación tanto de trama como de multitrama. El *bit* de alineación en tramas impares se utiliza para alineación de tramas. El *bit* de alineación en tramas pares se utiliza para la alineación de multitramas. Hay únicamente 6 *bits* en cada multitrama para el patrón de alineación de multitrama. El patrón muestra el siguiente formato: "001110".

- Señalización por Canal Asociado en un PCM de 32 canales

En un medio ambiente TDM se usa todavía el sistema de señalización de registro MFC como el descrito anteriormente y se transfiere por la misma vía que la del canal de voz involucrado, sin embargo para transferir las condiciones de "línea" se introdujo un sistema nuevo de señalización, el cual fué mucho más eficiente: CAS (Channel Associated Signalling). Para señalización CAS en un medio ambiente TDM, se llevan 30 conversaciones en un enlace, dado que el canal cero se ocupa para la alineación de tramas y el 16 para señalización. Normalmente cada canal de voz también tiene que llevar su propia señalización de línea, la señalización CAS puede, sin

embargo, incrementar la eficiencia mediante la combinación de todas las señales de línea en el canal 16.

En condiciones de señalización (señalización de línea), se transfiere la condición del canal de voz entre los circuitos de troncales de dos centra les dadas. La condición de un canal se representa por una combinación de 4 *bits*, luego entonces se puede representar un total de 15 condiciones dioferentes para un canal de voz.

En una trama de 32 canales PCM, cada canal tiene 8 *bits*, incluyendo el canal 16. Luego el canal 16 de cada trama representa la condición de 2 canales de voz en el PCM: el L y el L+16 (L=1,..., 15); por lo tanto se necesitan 16 tramas para señalizar los 30 canales de voz del PCM; así el grupo de 16 tramas que contienen la señalización de las 30 conversaciones se llama multitrama, ver figuras 1.28 y 1.29.

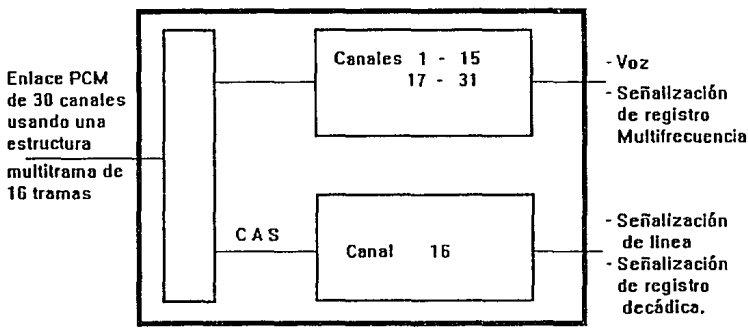


Figura 1.29 Enlace PCM de 30 canales, equipado con CAS[6]

Señalización de Registro

En un ambiente digital, también se aplica la señalización de registro MFC. Sin embargo los registros transmiten y reciben muestras binarias que representan un par de frecuencias.

- Señalización de Canal Común SCC

El objetivo de la señalización es pasar la información de una llamada de una central a la siguiente de la manera más eficiente, ésta información es la siguiente:

i) La intención de comenzar o finalizar una llamada,

ii) La selección de información,

iii) La identidad de la trayectoria de conversación que será utilizada.

En todos los sistemas de señalización como CAS, esto se lleva a cabo sobre la trayectoria de conversación seleccionada (iii), sobre la cual se hace primero la señalización de línea (i), para después enviar la señalización de registro (ii).

La eficiencia en la señalización podría incrementarse enormemente mediante el equipamiento entre ambas centrales de una conexión directa de señalización, sobre la cual toda la información se enviará directamente entre las centrales inteligentes de este modo tenemos un esquema de señalización que se llama señalización de canal común.

En la señalización de canal común, cada toma ó liberación será tratada como información. Por analogía, se dice que entre ambas centrales se proporciona un servicio de oficina postal que se responsabiliza de la recepción, transmisión y distribución de cartas que contenienen información de señalización. Dichas centrales postales se interconectan entre centrales vecinas mediante un enlace de señalización.

Cada vez que hay una toma, los circuitos de troncales (**TC** en la figura 1.30) escribirán una carta informando a los circuitos de la troncal de entrada de la otra central de los siguientes sucesos:

- Comenzará una llamada (señalización de línea),

- Se ha identificado un destino por información de selección (señalización de registro)

- Se ha seleccionado una troncal por la central de salida (identidad de la trayectoria de conversación)

La carta será direccionada al circuito de la troncal de entrada asociada y será suministrada a la oficina postal, la cual proveerá la carta a la oficina postal de la central destino. La oficina postal en esta central suministrará la carta al circuito de troncal correcto. (Fig. 1.30).

En el sistema de telefonía 1240 de ITT la oficina postal es un módulo o elemento de control adicional a los modulos de troncales que se llama Módulo de Señalización de Canal Cómun (CCSM), y dado que maneja tanto señalización de linea como señalización de registro tiene la ventaja sobre la señalización CAS, la cual solamente maneja señalización de linea, de ser más rápido sobre CAS. La señalización CCS puede centralizar las funciones de tasación, mantenimiento y administración. La duplicación del módulo es vista como desventaja pero es necesaria para dar confiabilidad al sistema.

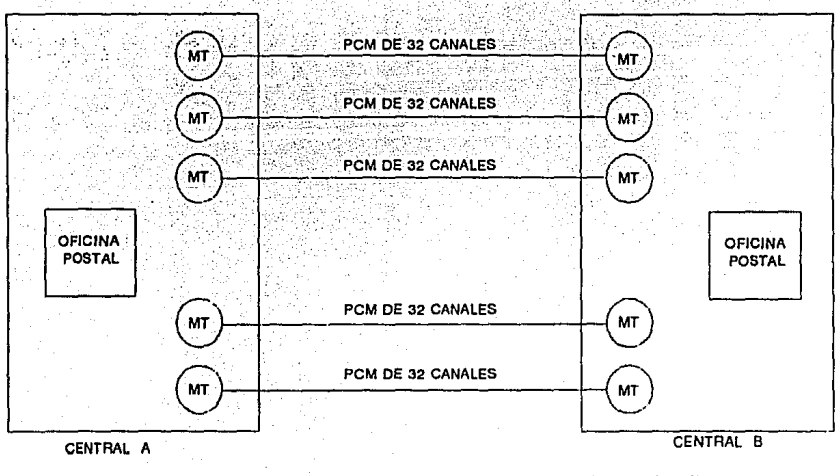


Figura 1.30 Señalización por canal común[6]

REFERENCIAS

[1] - Lathi B. P.
Sistemas de comunicación
Ed. Interamericana, México 1986

[2] - Schwartz M.
Transmisión de Información, Modulación, y Ruido.
Ed. Mc Graw Hill, 1987

[3] - Herbert Taub; Donald Schilling
Principles of Communication Systems
Ed. Mc Graw Hill, 1986

[4] - Simon Haykin
Digital Communications
Ed.John Wiley & sons, 1988

[5] - R. Bonami; J. M. Cotton
Central ITT 1240: Arquitectura
S. Das; K. Strunk
Central ITT 1240: Descripción de Equipo Físico
Revista Comunicaciones Eléctricas
Volumen 56 número 213. 1981

[6] - Bell Education Centre
Handout; Introduction to Digital Telephony
Edition 04, 1989

CAPITULO 2

CONCEPTOS FUNDAMENTALES DE LA COMUNICACION DIGITAL CODIFICADA

2.1 Introducción

Cuando se transmiten datos con formato digital sobre un canal ruidoso existe siempre la posibilidad de que los datos recibidos contengan errores. El diseñador de un sistema de comunicaciones establece generalmente una tasa de error arriba de la cual los datos recibidos no son utilizables. Si el dato recibido no cumple con los requisitos del límite establecido en la tasa de error, entonces se justifica[2] desde el punto de vista técnico utilizar la codificación para reducir los errores a un nivel al cual pueden ser tolerados. En los últimos veinte años se ha generalizado la utilización de códigos de corrección de errores para resolver este tipo de problemas.

La utilidad de la codificación se demostró con el trabajo de Shannon. En 1948 Shannon demostró que si la tasa de la fuente de datos es menor a una cantidad llamada capacidad del canal, entonces es posible la comunicación sobre un canal ruidoso con una probabilidad de error tan pequeña como se desee utilizando un esquema de codificación y descodificación apropiado. En esencia, el trabajo de Shannon establece que la señal de potencia, el ruido del canal y el ancho de banda disponible, establecen un límite solamente en la tasa de comunicación y no en la exactitud.

Es conveniente mencionar que en un sistema de comunicaciones, el límite real en la velocidad de comunicación se establece no por la capacidad del canal sino por el costo en la implantación de esquemas de codificación.

Las limitaciones en costo obligan a que la comunicación sea a velocidades sustancialmente menores que la capacidad del canal. En los últimos años se han dedicado bastantes esfuerzos a la búsqueda de esquemas de codificación prácticos y eficientes para diferentes tipos de canales ruidosos. La mayoría de la investigación dedicada al descubrimiento de estos esquemas prácticos de codificación se ha desarrollado en los últimos años[2], y ahora la codificación puede dar significativas mejorías de desempeño en muchas aplicaciones.

Existe un gran número de aplicaciones en donde se han construido equipos de codificación y han tenido una utilidad práctica exitosa. La creciente utilidad práctica de la codificación no se debe solamente a nuevos desarrollos dentro del campo de códigos de corrección de error sino también a los grandes avances en cuanto a la reducción del costo y tamaño de los dispositivos electrónicos de estado sólido. La aplicación manejada a través de éste trabajo, sobre la cual se sugiere utilizar un algoritmo de descodificación para códigos convolucionales, es una central telefónica de tecnología digital como la descrita en el capítulo 1, y es ejemplo de la aplicación real de tecnologías modernas.

2.2 Conceptos básicos de codificación

La codificación es esencialmente una técnica de procesamiento digital de señales[1] que se emplea para mejorar la **confiabilidad** de la comunicación de un sistema a través de un canal digital.

Aunque existen modelos individuales de codificación que toman diversas formas y tienen raíces en diversas disciplinas matemáticas, todas ellas tienen ingredientes comunes, uno es el uso de **redundancia**. En los mensajes digitales codificados siempre se encuentran símbolos adicionales al mensaje llamados **símbolos redundantes**. Estos símbolos se emplean para acentuar la singularidad de cada mensaje, y siempre se escogen para hacer poco probable que las perturbaciones del canal corrompan a los símbolos del mensaje y destruyan su unicidad. El segundo ingrediente es la **ponderación del ruido**[2]. Este efecto se obtiene al hacer que los símbolos redundantes dependan de un rango dado entre

varios símbolos de información, entre mayor sea el número de símbolos de información es más fácil detectar la singularidad del mensaje. Lo anterior implica que en un bloque de información transmitido la fracción de errores que debe ser corregida decrece conforme aumenta la **longitud** del mismo, entendiéndose por **longitud** el número de símbolos que contiene el mensaje; luego, el potencial de mejoramiento que proporciona la ponderación del ruido es en el sentido de reducir el número de errores en un bloque de información aumentando la longitud de éste. Se puede decir entonces, aplicando el concepto anterior, que los códigos con mayor longitud de bloque deberían de ser más efectivos que los códigos con longitud de bloque más corta.

Aún con el potencial de mejoramiento que ofrece la ponderación del ruido queda resolver el problema de corregir los errores en el bloque de información considerado, lo cual se lleva a cabo con ayuda de la **redundancia**.

En la representación de mensajes codificados por medio de **n** símbolos binarios, no se permite el uso de todas las 2^n secuencias posibles como mensajes legítimos. Ya que si todas las secuencias posibles recibidas de **n** símbolos fueran mensajes legítimos, no se tendrían bases suficientes para determinar si una secuencia es más válida que cualquier otra.

De lo anterior cabe preguntarse cuántos de los **n** símbolos de cada secuencia pueden formar parte del mensaje legítimo; la respuesta está en función del número de errores que se van a corregir en la secuencia recibida. Se puede demostrar[2][3] que para corregir **t** o menos errores, es necesario y suficiente que cada secuencia o mensaje legítimo difiera de cualquier otra secuencia o mensaje legítimo en por lo menos **2t+1** posiciones.

Por ejemplo, si en un código se desean corregir todos los símbolos erroneos simples y dobles, es necesario que todos los pares de secuencias codificadas difieran en por los menos cinco posiciones. Cualquier secuencia que contenga dos errores y por tanto difiera de la secuencia codificada correcta en dos posiciones exactamente, será diferente de cualquier otra secuencia codificada o mensaje legitimo en tres posiciones por los menos.

El número de posiciones en la cual dos secuencias difieren una de la otra se llama *distancia de Hamming* **d**, entre las dos secuencias. El menor valor para todos los pares de secuencias de cualquier código se llama *distancia mínima* del código, y se designa como d_{min}. De esta forma

d_{min} siempre debe ser uno más del doble del número de errores que se desean corregir[1][2] o sea:

$$t=\lfloor (d_{min}-1)/2 \rfloor \qquad (2.1)$$

donde el parámetro **t** indica que todas las combinaciones de **t** o menos errores, en alguna secuencia recibida, pueden ser corregidos y el símbolo [] representa el valor entero del cociente indicado.

Sea **Pe** la probabilidad de recibir un símbolo erroneo dentro de un patrón dado, la probabilidad de que ocurran **i** errores en ese patrón[2][4] es:

$$Pe^i(1-Pe)^{n-i} \qquad (2.2)$$

Donde **n** el número de *bits* de la palabra código o patrón recibido.

Se puede comprobar que para $Pe<1/2$:

$$(1-Pe)^n>Pe(1-Pe)^{n-1}> Pe^2(1-Pe)^{n-2} >... \qquad (2.3)$$

La desigualdad (2.3) indica que un patrón sin error (probabilidad $(1-Pe)^n$) es más probable que un patrón de simple error (probabilidad $Pe(1-Pe)^{n-1}$); un patrón de simple error (de probabilidad $Pe(1-Pe)^{n-1}$) es más probable que un patrón con error doble (probabilidad $Pe^2(1-Pe)^{n-2}$), ...etc. Esto significa que un descodificador, que recibe una secuencia en particular toma en cuenta la palabra código más "cercana" en distancia Hamming seleccionando así la palabra transmitida más probable (asumiendo que todas las palabras transmitidas son equiprobables). Un descodificador implementado con esta regla de descodificación es un descodificador de **máxima verosimilitud**, y este provee, bajo estas condiciones, mínima probabilidad de error[4]. En este sentido es un descodificador óptimo, el cual es frecuentemente empleado en códigos cortos ($n<10$). También este descodificador sirve como marco de referencia para comparar el desempeño de códigos subóptimos. En forma conceptual el descodificador de máxima verosimilitud es útil para ilustrar ciertas propiedades de los códigos de bloque si el descodificador se concibe con las secuencias código en forma tabular asociadas con las secuencias que no pertenecen al código.

El método tabular para describir los diferentes patrones de un código es conocido como la **tabla de descodificación** que a continuación se describe.

Se pueden encontrar diferentes formas de crear la tabla de descodificación. La tabla de descodificación se puede realizar con un código de secuencias de longitud **n**, asociando todas las secuencias recibidas posibles que no pertenecen al código con la secuencia o palabra código más "cercana" posible; cada una de estas asociaciones se escriben en forma de columna cuyo primer renglón será la palabra código debajo de la cual se escribirán todas las secuencias que difieren en una sola posición de ésta. A continuación de estas secuencias se escriben, también en forma de listado todas las secuencias posibles pero que difieren en dos posiciones con la palabra código que encabeza la columna, y así hasta listar todas las palabras que no son del código, la tabla tendrá tantas columnas como palabras contenga el código. Para ilustrar ésta descripción se seleccionó el código formado por las secuencias **00000**, **00111**, **11100**, y **11011** cada una de las cuales puede utilizarse para representar uno de cuatro mensajes posibles. En la figura 2.1 se ilustra el procedimiento mencionado.

Palabras código legítimas	**00000**	**11100**	**00111**	**11011**
Secuencias recibidas que difieren en una posición	10000	01100	10111	01011
	01000	10100	01111	10011
	00100	11000	00011	11111
	00010	11110	00101	11001
	00001	11101	00110	11010
Sec. recibidas que difieren en dos posiciones	10001	01101	10110	11010
	10010	01110	10101	01001
	.	.	.	.
	.	.	.	.
	.	.	.	.

Fig.2.1 Tabla de descodificación para un código de cuatro palabras de longitud n=5.

Nótese que el código abarca solamente una fracción de 2^n=32 secuencias de longitud n=5, y que ésta fracción implica poder seleccionar palabras código que difieren una de la otra en tres posiciones por los menos, así este código tiene una distancia mínima de tres y puede corregir solamente un error en cualquier posición.

Esta tabla se usa en el proceso de descodificación tomando como secuencia legítima a la salida del descodificador la palabra código más "cercana" a la secuencia recibida correspondiente a la columna de la tabla que contiene la secuencia recibida. Nótese que para cada secuencia recibida que forma parte de los dos últimos renglones de la tabla (fig 2.1), ésto es, las que difieren en al menos dos posiciones de las palabras código, no existe una forma única de asignación en la tabla lo cual no ocurre con las secuencias que difieren en una posición de las palabras código. Por ejemplo, la secuencia recibida 10001 podría asignarse a la cuarta columna pero también a la primer columna. Esta descripción corresponde en sí a la descodificación de máxima verosimilitud, desafortunadamente el tamaño de la tabla crece exponencialmente con la longitud del bloque codificado, de manera que el uso de la tabla de descodificación se vuelve impráctico para códigos largos.

Para un código de corrección de t errores, el número total de renglones (8 para el caso de la tabla anterior), **Ne**, en cada subconjunto, obedecen la desigualdad[2]:

$$Ne \geq 1+n+C(n,2)+...+C(n,t) \qquad (2.4)$$

donde **C(n,i)** es el coeficiente binomial i-ésimo, esto es,

$$C(n,i)= n!/[i!(n-i)!],\ i=1,..,t.$$

De esta inecuación se puede observar que existen **n** patrones que difieren en una posición de la correcta y **C(n,2)** patrones que difieren en dos posiciones, etc.

Se puede relacionar la cantidad de *bits* de redundancia de un código con el número de errores que se pueden corregir. Observando primero que existen 2^n secuencias posibles, y que cada columna de la tabla de descodificación contiene un número de **Nc** renglones corregibles de esas secuencias, de tal forma que el número de palabras código, **Nc**, debe obedecer la desigualdad:

$$Nc \leq 2^n/[1+n+C(n,2)+...+C(n,t)] \qquad (2.5)$$

Esta es la llamada **cota de Hamming**. Esta inecuación es aplicable únicamente para los códigos perfectos. Esto es códigos que pueden corregir solamente los patrones de t o menos errores. Existe un número muy reducido de códigos perfectos que satisfacen la inecuación anterior.

La **codificación** es el proceso por el cual la secuencia de información de **k** símbolos son mapeados con las secuencias código de **n** símbolos generándose secuencias de **k** símbolos de información y **n-k** símbolos redundantes. Los códigos así generados se les conoce como códigos **(n,k)**. Dado que la secuencia de k símbolos pueden tomar 2^k valores distintos, la inecuación anterior se puede reescribir como:

$$2^k \leq 2^n / [1+n+C(n,2)+...+C(n,t)] \qquad (2.6)$$

Una medida de la **eficiencia** empleada para un código en particular está dado por la razón

$$R = k/n \qquad (2.7)$$

Donde **R** se define como la **tasa del código**. La fracción de símbolos transmitidos que son redundantes es **1-R**.

2.3 Descripción general de un proceso de comunicación digital

En el diagrama a bloques de la figura 2.2 se describe el proceso de comunicación digital. Este modelo es general y la función de cada uno de sus bloques se presentará a continuación.

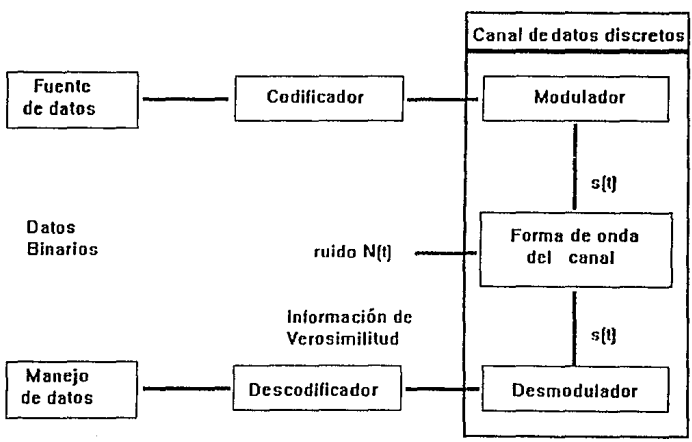


Fig.2.2 Modelo general de un proceso de comunicación digital

La fuente de datos[2]

A partir de ésta se generan datos en forma de símbolos binarios. Normalmente se asume que los datos se han procesado de tal forma que los "unos" y "ceros" individuales ocurren independientemente y con igual probabilidad. De cualquier forma, en situaciones del mundo real existen patrones de datos que ciertamente prevalecen más que otros, y esto puede en algunas ocasiones invalidar conclusiones que están basadas en datos completamente aleatorios.

El codificador[2]

Dos tipos de códigos se toman en cuenta dentro de éste trabajo: los **códigos de bloque** y los **códigos de árbol**. La diferencia principal entre los codificadores para estos dos tipos de código es la presencia o ausencia de memoria.

Conceptualmente el codificador que genera un código de bloque es un dispositivo **sin memoria**, el cual mapea una secuencia entrante de **k** símbolos en una secuencia de salida de **n** símbolos. El término **sin memoria** indica que cada bloque de **n** símbolos (secuencia codificada) dependen sólamente de un bloque específico de **k** símbolos (secuencia a codificar), y no porque el codificador carezca de circuitos de memoria. Los parámetros que caracterizan a un código de bloques son **n**, **k**, **R=k/n**, y $\mathbf{d_{min}}$. Los valores prácticos para **k** son el rango de 3 a varios cientos, para **R** de 1/4 a 7/8. Las secuencias de entrada y salida son por lo general símbolos binarios y muy raramente son símbolos de un alfabeto de mayor orden.

El codificador para un código de árbol es un dispositivo **con memoria** que acepta símbolos binarios en conjuntos de **m** elementos y genera símbolos binarios de salida en conjuntos de **n** elementos. Cada conjunto de **n** símbolos de salida es determinado por el conjunto de **m** símbolos de entrada corrientes y un rango de **m** símbolos de entrada precedentes. Se dice entonces que el rango de memoria del codificador es de **N=m+m**, es decir, la cantidad de símbolos codificados que son influenciados por un *bit* que pasa por el codificador. En el apartado (2.5) se verán conceptos sobre códigos de bloque y convolucionales en detalle.

Otra forma de clasificar los códigos es definiendolos como **códigos lineales** y **códigos no lineales**. Los códigos lineales por definición forman un espacio vectorial en el que cada palabra código es un elemento o vector del espacio, y tienen la propiedad fundamental que, mediante la adecuada definición de la suma sobre los elementos del código, dos palabras código cualquiera pueden ser sumadas y el resultado será siempre otra palabra del mismo código (propiedad de cerradura). En el caso particular de un código binario, esta operación es la suma exclusiva símbolo a símbolo de las dos palabras código. Esta propiedad tiene dos efectos importantes de gran alcance, el primero es que se simplifica el problema de codificación y descodificación porque permite expresar cualquier palabra código como combinaciones "lineales" de un pequeño grupo de palabras código de referencia conocidas como **vectores base** o **base** del espacio vectorial. El

segundo efecto es que también se simplifica significativamente el problema del cálculo del desempeño haciendo la distancia, entre dos palabras código, equivalente a la distancia entre la palabra toda cero y cualquier otra palabra código[3]. Así, cuando se calcula el desempeño del código, es necesario considerar únicamente el efecto de transmitir la palabra código toda cero[3]. Este cálculo de desempeño se facilita más aún sabiendo que la distancia de Hamming entre una palabra código dada y la palabra código toda cero es equivalente al número de elementos no cero en la palabra dada. Este número se conoce como **peso de Hamming** de la palabra código; se puede emplear una lista que contenga el número de palabras código con su respectivo peso, para estimar el desempeño del código mediante un parámetro llamado la **cota de unión**. Tal lista se le conoce como **estructura de peso** del código.

Casi todos los esquemas de codificación empleados en aplicaciones prácticas son códigos lineales. Los códigos lineales de bloque son llamados **códigos de grupo**, ya que las palabras código forman una estructura matemática llamada grupo. Los códigos lineales de árbol son llamados comunmente **códigos convolucionales**, ya que la operación de codificación puede ser modelada a través de una operación de convolución en tiempo discreto de la secuencia de entrada con la respuesta impulso del codificador.

Dado que los códigos no lineales carecen de aplicación práctica conocida no se menciona nada acerca de ellos, pero obsérvese que los códigos también pueden clasificarse como **códigos para corrección de errores aleatorios** y **códigos para corrección de ráfagas de errores**. Los primeros se diseñan en base a corregir errores en secuencias recibidas cuyas probabilidades de ocurrencia de los símbolos que las conforman son independientes entre sí y los segundos se diseñan para corregir errores en secuencias en las que la probabilidad de recibir un símbolo dado dentro de la secuencia depende del símbolo enviado y además de los simbolos restantes en la secuencia.

El modulador[2]

El modulador genera un conjunto de formas de onda de duración finita y hace un mapeo entre la salida del codificador y un conjunto de formas de onda. El objetivo básico de tal mapeo es adecuar la señal digital para que ésta sea capaz de viajar a través de un medio de transmisión dado bajo ciertas limitaciones de potencia en la transmisión y del equipo físico de recepción.

Para un esquema binario (digital) de modulación, cada símbolo de salida que viene del codificador se emplea para seleccionar una de dos posibles formas de onda.

En este trabajo no es de mucha utilidad hablar del modulador ya que aquí la señal a procesar no se va a transmitir sino que se va a almacenar en un sistema de memoria y posteriormente se va a recuperar de ella.

El canal de formas de onda[2]

El canal de formas de onda consiste de toda la circuitería o medio físico por el cual pasa la forma de onda que va desde la salida del modulador hasta la entrada del desmodulador. El sistema de comunicación no tiene que ser forzosamente un sistema en tiempo real, puede tratarse de un sistema de memoria de datos o de un sistema de grabación también. En el caso común, la salida $s_0(t)$ del canal es una réplica escalada de la entrada $s_i(t)$, a la cual se le agrega una perturbación aleatoria **n(t)**. Las perturbaciones pueden causar la supresión de la señal o que la amplitud de la señal recibida varíe o el canal varíe con el tiempo. La perturbación **n(t)** puede ser simple ruido recibido el cual puede ser modelado como un proceso aditivo Gaussiano o pudiera ser ruido urbano. En el ejemplo analizado se tiene que el canal es la memoria del procesador de una central telefónica en la cual se almacenarán señales digitales cuyo ruido adicionado en la misma puede consistir en retardos de tiempo provocados ya sea por las características de tiempo de acceso a la misma memoria o porque el sistema de memoria pierde su fidelidad debido a que la información almacenada en la memoria se va degradando con el tiempo.

El desmodulador[2]

El desmodulador es un dispositivo que estima cuál de los símbolos posibles fue transmitido en base a observaciones de la señal recibida. La probabilidad de que esta estimación sea correcta depende de la razón de la potencia de la señal a la potencia del ruido en el ancho de banda de datos, a la cantidad de distorsión de la señal causada por los efectos del filtrado, aunado a efectos no lineales, y al esquema de detección que se está utilizando. En un sistema con codificación el desmodulador cumple una segunda función. Esta función es la de abastecer información al descodificador que a su vez da confiabilidad en la decisión de cada símbolo recibido. Esta información se obtiene de diferentes maneras, y la técnica utilizada depende de la naturaleza de la perturbación **n(t)**.

El descodificador[2]

El descodificador es el dispositivo que invierte la operación del codificador. Dado que la secuencia de símbolos que son generados por el desmodulador pueden contener errores, el descodificador efectúa un mapeo mucho más complejo que el realizado por el codificador. Para hacer factible la operación de descodificación se debe concebir el procedimiento de cálculo para realizar este proceso. Los algoritmos de descodificación se pueden clasificar según las técnicas aplicadas, ya sea a los códigos de bloques o a los códigos de árbol. Las dos clases principales de algoritmos de descodificación de bloques son:

1.-El procedimiento basado en estructuras elementales de código, incluyendo información de descodificación y,

2.-el procedimiento algebráico, basado en estructuras algebráicas de códigos resolviendo básicamente ecuaciones algebraicas.

Las cuatro clases principales de algoritmos[2] para la descodificación de códigos de árbol son:

1.- El algoritmo de Viterbi (descodificación de máxima verosimilitud)

2.- algorítmo de descodificación secuencial, que es subóptimo respecto al de Viterbi

3.- descodificación de umbral.

4.- descodificación por tablas de estimación.

Más adelante se analizarán los dos primeros algoritmos.

2.4 El canal de datos discreto

Desde el punto de vista del codificador y el descodificador de la figura 2.2, el recuadro con líneas punteadas encierra lo más importante. El canal se caracteriza por los símbolos de entrada, símbolos de salida y probabilidades de transición.

En el caso más simple, la probabilidad de transición **P(j/i)** es invariante en el tiempo y su valor es independiente símbolo a símbolo, y define al canal conocido como **canal discreto sin memoria (DMC)**[1][2]; en la probabilidad **P(j/i)**, **i** denota un símbolo a la entrada del modulador, **j** denota un símbolo a la salida del desmodulador y **P(j/i)** denota la probabilidad de recibir el símbolo **j** dado que se envio el símbolo **i**.

El caso comunmente encontrado de **(DMC)** es el llamado **canal binario simétrico (BSC)**. Este canal se representa frecuentemente como en la figura 2.3:

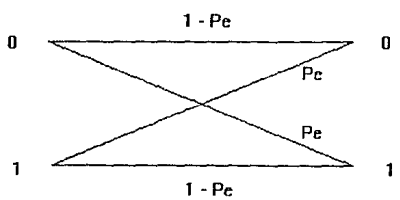


Fig.2.3 El canal binario simétrico.

En casi cualquier modelo de un canal de datos discreto, la probabilidad de error casi siempre varía inversamente con la relación señal a ruido. Esto obliga hacer una observación muy importante: el empleo de redundancia no es libre[2]. Si los símbolos de redundancia son incluidos en una transmisión digital y simultáneamente la tasa de transmisión de información y la potencia permanecen constantes, entonces la energía asociada con cada símbolo transmitido se ve reducida y consecuentemente, la tasa de error se ve incrementada. Por ejemplo, para transmitir un mensaje de 12 *bits* a una tasa de uno por segundo utilizando 1 W de potencia, cada símbolo binario utilizará 1/12 de joule; sin embargo, si este mensaje fuese codificado con un código de longitud 24 (50% de símbolos redundantes) y transmitido a la misma tasa que los símbolos del mensaje, entonces cada símbolo binario transmitido utilizaría 1/24 de Joule; obviamente la probabilidad de un símbolo erroneo para el caso del mensaje codificado es mayor que para el mensaje sin codificar. Si el esquema de codificación tiene la intención de producir una ganancia neta en el desempeño, este debe mejorar la tasa de error lo suficiente para estar por encima de las pérdidas a las que se incurrieron en primer lugar al introducir redundancia. Dado que el mejoramiento depende en gran parte del número de errores a corregir, es importante corregir tantos errores como sea posible para una tasa del código de longitud de bloque fija.

2.5 Códigos de bloque y códigos convolucionales

En este apartado se retoman los conceptos tratados inicialmente en el apartado 2.3 cuando se habló del codificador actuando en el proceso de comunicación digital codificada. Aquí se profundiza más acerca de los códigos de bloque y códigos convolucionales describiendo brevemente las bases teóricas sobre las que se sustentan y desarrollan; los conceptos aquí explicados serán útiles además para comprender ciertos aspectos teóricos mencionados en los capítulos 3 y 4.

Códigos de bloque[2][4]

En un código de bloque una palabra código consta de **n** dígitos $c_1, c_2,...,c_n$, y una palabra de datos consta de k dígitos $d_1,d_2,...,d_k$. En virtud de que la palabra código y una palabra dato son de **n** y **k** elementos respectivamente, entonces se dice que son vectores de dimensión **n** y **k** respectivamente. Se usarán matrices renglón para representar estas palabras.

$$\underline{c}=(c_1,c_2,...,c_n) \qquad (2.8)$$

$$\underline{d}=(d_1,d_2,...,d_k) \qquad (2.9)$$

Para el caso general de los códigos de bloque, los **n** dígitos de $\underline{c}$ se forman mediante combinaciones lineales (suma módulo-2) de **k** dígitos de datos. La figura 2.4 muestra un posible codificador para este código que utiliza un registro de desplazamiento de k dígitos y **m** sumadores módulo 2.

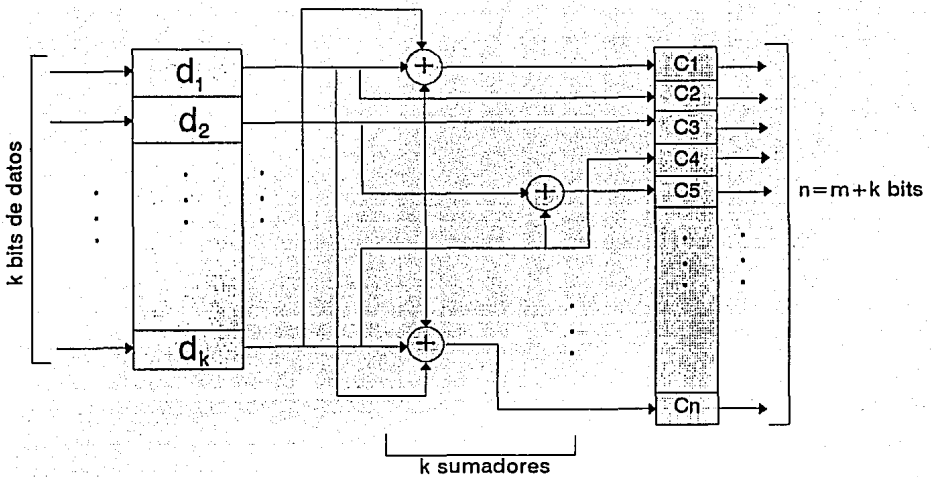


Fig.2.4 Un codificador para códigos de bloque.

Matriz generadora y Matriz de Verificación de Paridad[2]

En particular cuando $c_1 = d_1$, $c_2 = d_2$, ... , $c_k = d_k$ y los dígitos restantes del código c_{k+1} a c_n son combinaciones lineales de d_1, d_2,..., d_k el código generado se llama **código sistemático**. De esta manera, en un código sistemático, los primeros k dígitos de una palabra código son los dígitos de datos, y los últimos m = n-k dígitos son los dígitos de verificación de paridad, que se forman mediante combinaciones lineales de dígitos dato d_1, d_2,..., d_k como se indica a continuación:

$$
\begin{aligned}
c_1 &= d_1 \\
c_2 &= d_2 \\
&\vdots \\
c_k &= d_k \\
c_{k+1} &= h_{11}d_1 \,\theta\, h_{21}d_2 \,\theta \ldots \theta\, h_{k1}d_k \\
c_{k+2} &= h_{12}d_1 \,\theta\, h_{22}d_2 \,\theta \ldots \theta\, h_{k2}d_k \\
&\vdots \\
c_n &= h_{1m}d_1 \,\theta\, h_{2m}d_2 \,\theta \ldots \theta\, h_{km}d_k
\end{aligned}
\qquad (2.10)
$$

donde θ indica la suma módulo 2.

o bien: $\underline{c} = \underline{d} * \underline{G}$ (2.11)

en donde:

$$\underline{G} = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & \cdots & 0 & h_{11} & h_{12} & \cdots & h_{1m} \\ 0 & 1 & 0 & \cdots & 0 & h_{21} & h_{22} & \cdots & h_{2m} \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 & h_{k1} & h_{k2} & \cdots & h_{km} \end{array}\right] \qquad (2.12)$$

$\underline{I}_k(k*k)$ $\qquad$ $\underline{P}(k*m)$

La matriz $\underline{G}(k*n)$ se llama **matriz generadora**, y se puede particionar en una matriz identidad $\underline{I}_k(k*k)$ y una matriz $\underline{P}(k*m)$. Donde todos los elementos de $\underline{P}$ son 0 ó 1. La palabra código se pude expresar como:

$$\begin{aligned} \underline{c} &= \underline{d}*\underline{G} \\ &= \underline{d}*[\underline{I}_k,\underline{P}] \\ &= [\underline{d},\underline{d}*\underline{P}] \\ &= [\underline{d},\underline{cp}] \end{aligned} \qquad (2.13)$$

en donde $\underline{cp}$ es la matriz renglón de m dígitos de verificación de paridad:

$$\underline{cp} = \underline{d}*\underline{P}. \qquad (2.14)$$

Si se conocen los dígitos de datos, se pueden calcular los dígitos de verificación con la ecuación (2.14).

Escribiendo (2.14) explicitamente:

$$\underline{cp} = (c_{k+1}, c_{k+2}, c_{k+3}, \ldots, c_{k+m}) \qquad (2.15)$$

donde el digito de verificación de paridad c_{k+j} (donde j=1,...,m) se puede escribir, según (2.10), de la siguiente forma:

$$c_{k+j} = \sum_{i=1}^{k} h_{ij}\, d_i, \qquad j=1, 2, \ldots, m \qquad (2.16)$$

y dado que en un código sistemático $c_i = d_i$ (i=1,...,k), la ecuación (2.16) se convierte en

$$c_{k+j} = \sum_{i=1}^{k} h_{ij}\, c_i \qquad (2.17)$$

o bien:

$$\sum_{i=1}^{k} h_{ij}\, c_i - c_{k+j} = 0 \qquad (2.18)$$

tomando en cuenta (2.16) para cada uno de los **m** elementos de verificación de paridad se obtiene la siguiente ecuación :

$$[c_1\ c_2\ \cdots\ c_n] * \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ h_{k1} & h_{k2} & \cdots & h_{km} \\ -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 \end{bmatrix} = [0\ 0\ \ldots\ 0]; \qquad (2.19)$$

reescribiendo (2.17): $\underline{c} * \begin{bmatrix} \underline{P} \\ -\underline{I}_{n-k} \end{bmatrix} = \underline{0}, \qquad (2.19')$

donde $\underline{P}$ es la misma matriz contenida en $\underline{G}$, según ec. (2.12), y $\underline{0}$ es una matriz renglón de **m** componentes de valor cero.

Sea $\underline{H} = \begin{bmatrix} \underline{P} \\ -\underline{I}_{n-k} \end{bmatrix} = \begin{bmatrix} \underline{P} \\ \underline{I}_{n-k} \end{bmatrix}$;

como en aritmética binaria $-\underline{I}_{n-k} = \underline{I}_{n-k}$:

se concluye que $\underline{c} * \underline{H} = \underline{0}. \qquad (2.20)$

De lo anterior se sigue que $\underline{c}$ es una palabra código si y sólo si se verifica la ecuación (2.20), y $\underline{H}$ se llama **matriz de verificación de paridad.**

El Síndrome[2][3]

En el apartado 2.2 se ilustró mediante un ejemplo la descodificación de máxima verosimilitud construyendo la tabla de descodificación, también llamada **arreglo estándar**, de un código dado. La construcción del arreglo estándar muestra que la descodificación de máxima verosimilitud requiere de una capacidad de memoria que crece exponencialmente conforme crece la longitud del código.

Ahora se hace uso, en ésta discusión acerca de los códigos de bloques, del arreglo estándar descrito en el apartado 2.2 (figura 2.1), combinandolo con un cálculo relativamente simple para lograr una reducción significativa de los requerimientos de memoria en el descodificador con una pérdida insignificante en la velocidad de descodificación. Para esto se necesita el concepto de **síndrome de una secuencia** dado a continuación.

Para la secuencia recibida $\underline{c}'$ y la correspondiente palabra código transmitida $\underline{c}$, el síndrome se define como:

$$\underline{s} = \underline{c'} * \underline{H}; \qquad (2.21)$$

dado que la diferencia entre $\underline{c}$ y $\underline{c'}$ es la secuencia de error $\underline{e}$, esto es, $\underline{c'} = \underline{c}$ $\underline{H}$ $\underline{e}$; entonces

$$\underline{s} = \underline{c'} * \underline{H} = (\underline{c} \theta \underline{e}) * \underline{H} \qquad (2.21')$$

y finalmente, en virtud de la ecuación (2.18) :

$$\underline{s} = \underline{e} * \underline{H}. \qquad (2.22)$$

La ecuación (2.22) implica que el síndrome no es más que la aplicación de las ecuaciones de verificación de paridad de la palabra de error; ésto por sí mismo no representa ningún beneficio, pero el siguiente resultado le da al síndrome un valor muy útil: dos secuencias recibidas $\underline{c'}_1$ y $\underline{c'}_2$ se encuentran en el mismo renglón del arreglo estándar si y solo si sus síndromes son iguales. Así, si se recibe una secuencia $\underline{c'}$ proveniente de un canal ruidoso, y suponiendo que no se ha excedido la capacidad de corrección de errores del código, entonces se puede conocer la identidad de la secuencia de error al calcular el síndrome de $\underline{c'}$. Una vez que se conoce el valor de $\underline{e}$, se efectúa la suma exclusiva $\underline{e}$ θ $\underline{c'}$ y se obtiene la palabra código transmitida.

La clave de la reducción de la memoria requerida es que en lugar de almacenar un renglón completo del arreglo estándar, solamente se necesita almacenar la secuencia de error $\underline{e}$, junto con su síndrome[3].

Utilizando el síndrome, aún el arreglo más modesto se reduce en tamaño apreciablemente. En el arreglo del apartado 2.2 hay 2^5=32 elementos (28 secuencias de error y cuatro palabras código). En cambio, el nuevo requerimiento de memoria implica una capacidad únicamente para las palabras código, más siete secuencias de error y siete síndromes dando un total de 18 elementos.

En el capítulo 4 se apreciará la utilidad práctica del síndrome en el código de bloques ahí utilizado.

Códigos convolucionales[1][2][3][4]

Los códigos convolucionales (o recurrentes) difieren de los códigos de bloque por lo siguiente: en un código de bloque, el bloque de **n** dígitos generados por el codificador en cualquier unidad de tiempo depende solamente del bloque de **k** dígitos de datos de entrada dentro de esa unidad de tiempo. En un código convolucional, por otra parte, el bloque de **n** dígitos de código generados por el codificador en una unidad de tiempo particular dependen no sólo del bloque de **k** dígitos de mensaje dentro de esa unidad de tiempo, sino también del bloque de dígitos dato dentro del lapso previo de **N-1** unidades de tiempo (**N>1**).

La generación de las secuencias codificadas se realiza mediante una operación lineal entre los símbolos de entrada al codificador, esto es, la

convolución discreta de la secuencia o mensaje de entrada con la respuesta al impulso del codificador, por esta razón este tipo de códigos son conocidos como códigos convolucionales.

Similiarmente a la convolución de funciones continuas de variable continua (como por ejemplo, el voltaje y la corriente como funciones del tiempo), la convolución de dos secuencias digitales puede efectuarse como sigue:

1. Inviértase una de las secuencias dadas,

2. Hágase coincidir el primer dígito (derecho) de la secuencia invertida con el primer dígito (izquierdo) de la otra secuencia,

3. Multiplique los dígitos que están alineados, sume los productos formados por la alineación de secuencias, y tómese la suma de productos obtenida como un elemento de la convolución de las dos secuencias,

4. Desplace la secuencia invertida una posición relativa a la otra secuencia,

5. Repita los pasos 3 y 4 hasta que toda la secuencia invertida se halla desplazado a lo largo de la secuencia fija.

En el caso de secuencias binarias, el producto para cada posición es "0" o "1", y las sumas son sumas módulo 2 o sumas exclusivas, denotadas por el símbolo θ. Se supondrán secuencias binarias para discutir los códigos de convolución. A continuación se da un ejemplo de la operación de convolución sobre dos secuencias binarias:

Se desea obtener la convolución de las secuencias binarias **1101** y **10011**. Se acostumbra denotar la operación como **(1101)*(10011)**. Manteniendo fija la secuencia **10011** e invertiendo la secuencia **1101**, quedan ambas secuencias como sigue:

1 0 0 1 1

1 0 1 1

Para poder apreciar como procede el cálculo, hágase referencia a los cuatro primeros pasos mencionados anteriormente. Supóngase además que todos los dígitos que preceden y anteceden a ambas secuencias son ceros.

(1)	(2)	(3)	(4)
1 0 0 1 1	1 0 0 1 1	1 0 0 1 1	1 0 0 1 1
1 0 1 1	1 0 1 1	1 0 1 1	1 0 1 1

Luego los resultados obtenidos en cada uno de los cuatro pasos mostrados se dan a continuación:

Paso (1): $1*1 = 1$

Paso (2): $1*1 \ \theta \ 0*1 = 1$

Paso (3): $1*0 \oplus 0*1 \oplus 0*1 = 0$

Paso (4): $1*1 \oplus 0*0 \oplus 0*1 \oplus 1*1 = 0$

Y continuando con la operación entre ambas secuencias, como se ha descrito, la convolución en tiempo discreto es (primer digito a la izquierda) la secuencia binaria **11000111**.

En la codificación convolucional, se ve en general a los 1's de la secuencia fija como derivaciones de un registro de desplazamiento y a los ceros como etapas del registro que no tienen efecto sobre los *bits* de la secuencia que se desplaza, mientras que la secuencia que se desplaza es la secuencia correspondiente a los *bits* de datos que se mueven a través del registro de desplazamiento. La codificación toma lugar introduciendo uno o más *bits* a la vez en uno o más registros de desplazamiento. Sea n_0 el total de símbolos del canal a la salida de los registros de desplazamiento por cada k_0 *bits* de entrada, entonces la tasa del código de convolución es:

$$R = k_0 / n_0 \qquad (2.23)$$

Obviamente que un *bit* de entrada (para $k_0=1$) puede influir en la salida del registro de desplazamiento tanto como cantidad de etapas tenga el mismo. Así se refiere a **N** (número de etapas del registro) como la **longitud límite** del código convolucional. Esta observación es muy importante para el desarrollo del algoritmo de descodificación propuesto en este trabajo y que será descrito en el capítulo 3. También se hace notar que existe una relación entre las derivaciones de salida del registro de desplazamiento del codificador y la representación matemática del código.

La implantación de la codificación se realiza fácilmente mediante registro de corrimiento y una cantidad de v sumadores módulo 2. La figura 2.5 muestra un codificador para el caso de N=4 y v=3.

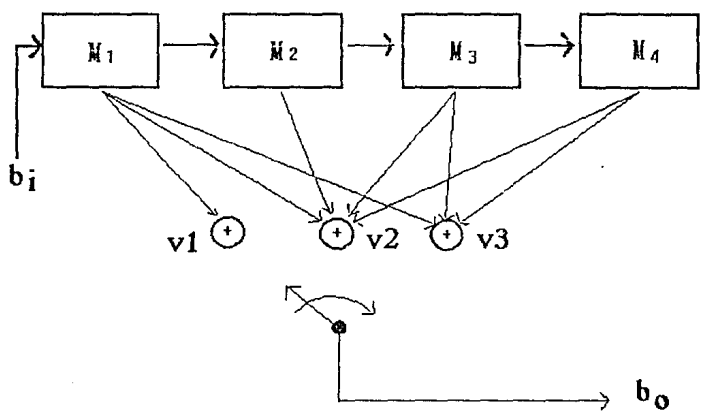


Fig.2.5 Un codificador de convolución.

La secuencia de *bits* b_j del mensaje se aplican a la entrada del registro de corrimiento. La secuencia de *bits* codificados b_0 se obtiene a la salida del conmutador. El conmutador muestrea los v sumadores módulo 2 en secuencia. Nótese que el sumador conectado a la etapa M_1 es una conexión directa de los datos de entrada a la salida del conmutador por lo que en realidad v=2, la frecuencia de muestreo del conmutador de salida es una vez durante cada intervalo de *bit* de entrada por lo que el código generado tendrá una tasa R=1/3. Las salidas v_1, v_2, v_3 de los sumadores en la figura son :

$$v_1 = s_1$$
$$v_2 = s_1 \oplus s_2 \oplus s_3 \oplus s_4 \qquad (2.24)$$
$$v_3 = s_1 \oplus s_3 \oplus s_4$$

La operación del codificador se explica de la siguiente manera: inicialmente el registro de corrimiento se borra; para el codificador mostrado los *bits* de datos entran de izquierda a derecha uno a la vez; el primer *bit* de datos de entrada se guarda en la etapa M_1, y durante este intervalo el conmutador muestrea las salidas del sumador v_1 v_2 y v_3 produciendo un solo *bit* de entrada tres *bits* de salida codificados; el codificador es por lo tanto de tasa 1/3. El siguiente *bit* entra en la etapa M_1, mientras que el *bit* inicialmente en M_1 se transfiere a M_2, y el conmutador una vez más muestrea todas las v=3 salidas del codificador. Este proceso continua hasta que eventualmente el último *bit* del mensaje ha sido introducido en M_1. A partir de ahí y con el objeto de que cada *bit* del mensaje pueda recorrer completamente cada etapa del registro de corrimiento, y por lo tanto ser involucrado en el proceso completo de codificación, se agregan suficientes ceros al mensaje para transferir el último *bit* del mensaje a través de la etapa M_4 y por lo tanto salir del registro de corrimiento. El registro de corrimiento entonces se encuentra en su condición inicial de borrado nuevamente.

Representación gráfica de códigos convolucionales[4]

Las secuencias de salida del codificador convolucional pueden representarse convenientemente en un **diagrama de árbol**. El árbol para el codificador de la figura 2.5 se muestra en la figura 2.6.

Cada **rama** del árbol representa la salida correspondiente a un conjunto particular de *bits* que se encuentran en el registro de desplazamiento, ésto es, los *bits* que fueron desplazados a la derecha del registro (fig. 2.5) durante la codificación en el intervalo de tiempo del bit de entrada anterior, y al *bit* que está siendo introducido al registro por la etapa M_1.

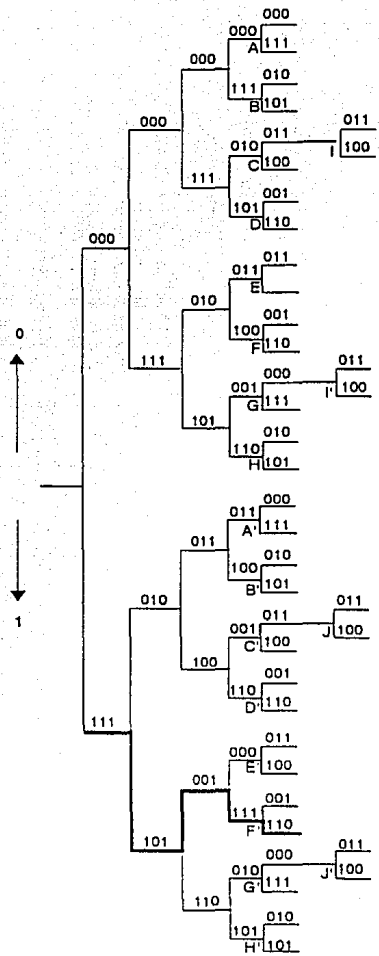


Figura 2.6 Arbol del código correspondiente al codificador de la figura 2.5.

Leyendo el árbol (fig. 2.6) de izquierda a derecha, los símbolos del canal aparecen en cada rama en el orden en que los transmite el conmutador de salida del codificador. Cada *bit* de entrada nuevo corresponde a un **nodo** del

árbol. El número de ramas posibles que emanan de un nodo del árbol es igual al número de símbolos del alfabeto con el que se representan los datos de entrada; aquí se trabaja con un alfabeto binario y por tanto emanan dos ramas de cada nodo del árbol. Por convención, las ramas superior e inferior de un nodo del árbol corresponden a los valores cero y uno de los *bits* de datos respectivamente. Una secuencia de entrada particular al codificador corresponde a una trayectoria única a través del árbol. Por ejemplo la secuencia de datos **11011...** (siendo el primer digito de entrada el de la izquierda) corresponde a la trayectoria mostrada con línea remarcada en la figura 2.6.

Obsérvese ahora algunas propiedades importantes del árbol. Primero: cualquier sucesión de ramas del árbol mostrado puede generarse suministrando el conjunto correspondiente de *bits* de entrada al codificador de la figura 2.5; segundo: de las dos ramas que emanan de cualquier nodo, todos los *bits* de una son el complemento binario o el conjugado (1 en lugar de 0 y 0 en lugar de 1) de los *bits* de la otra rama. La razón de lo anterior es la siguiente: cada *bit* de entrada tiene influencia en tantos *bits* de salida como número de etapas tenga el registro de desplazamiento del codificador; así, al cambiar la última entrada al registro de desplazamiento cambian las $(n_0-1)=2$ sumas módulo 2 que determinan los digitos de verificación.

Obsérvese también que comenzando con la quinta rama de cualquier secuencia hay conjuntos idénticos de ramas en las mitades superior e inferior del árbol, lo cual implica que ciertos pares de nodos podrían juntarse y ser así combinados; lo anterior es cierto, por ejemplo, para los nodos marcados con A y A' en la fig. 2.6. Lo mismo ocurre para los pares de nodos B-B', C-C', D-D', E-E', F-F', G-G', y H-H'. En vista de esta propiedad, se puede ver a los nodos A y A' como un mismo nodo, al igual que a cada uno de los pares de nodos B-B', C-C',..., H-H', etc. También los nodos subsecuentes I e I', J y J', etc., podrían asociarse de lamisma forma que los nodos A y A', etc., ésto es, si se asocian los pares de nodos A-A', ..., H-H' de la manera ya descrita, el nodo I coincide con el nodo J, y el nodo I' coincide con el nodo J'. De manera que para cada profundidad o número de nodos de una trayectoria del árbol dada, cuyo valor excede la longitud límite del código, el número de ramas subsecuentes puede reducirse a la mitad sin que la cantidad de información que pueda obtenerse del árbol del código también se reduzca. Combinando los nodos del árbol del código tal y como se indicó anteriormente, se obtiene una estructura con un patrón repetitivo cuya apariencia es la del nombre que lleva: **enrejado**. En el enrejado se utilizan los términos nodo y rama en el mismo sentido que en el diagrama de árbol. Para la descodificación de un código convolucional utilizando el algoritmo de Viterbi el enrejado prueba ser una herramienta útil para ilustrar de una manera compacta las secuencias codificadas como se verá más adelante.

Como cada *bit* que recién entra al registro del codificador causa que un *bit* "desaparezca" en el otro extremo del registro, entonces el número de *bits* dentro del codificador que tienen influencia sobre el triple de *bits* de salida del codificador, además del *bit* recién ingresado al codificador, es de N-1 (en el ejemplo dado N-1=3), se dice entonces que en cualquier instante el codificador tiene 2^{N-1} estados posibles sobre un conjunto de símbolos de salida correspondientes a cada *bit* de entrada elegido. Para el enrejado de la figura 2.7 (que corresponde al codificador de la figura 2.5), cada uno de los

ocho estados del codificador, mostrando el *bit* más reciente a la derecha, se lista al margen y en la línea correspondiente de los nodos en el enrejado, mientras que cada conjunto de los símbolos de salida aparecen en la rama que conecta al estado actual y su posible estado sucesor.

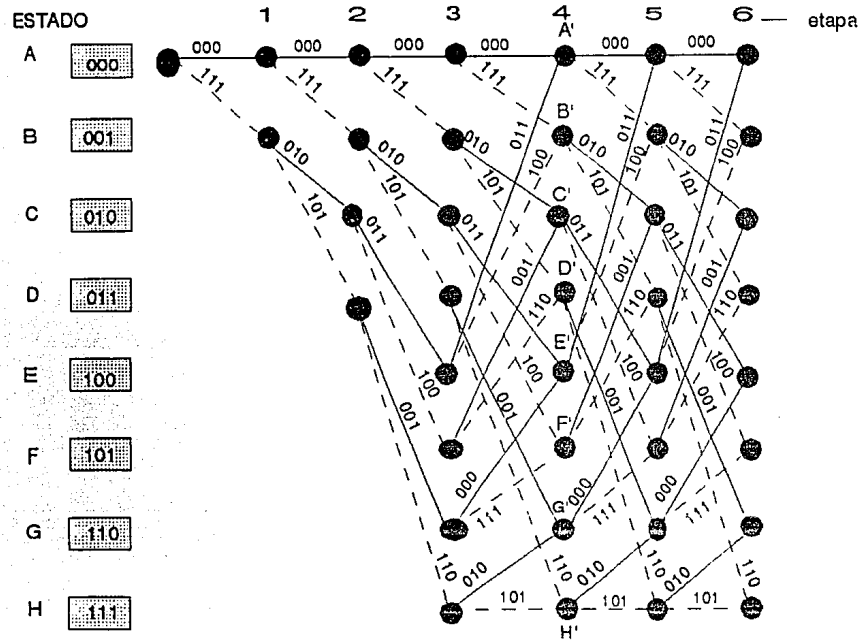


Figura 2.7 Enrejado correspondiente al código generado por el codificador de la figura 2.5.

Aquí, como en el diagrama de árbol, se elige de manera convencional y arbitraria que las ramas superiores e inferiores que emanan de un nodo correspondan a las entradas cero (línea firme) y uno (línea punteada)

respectivamente. La asignación de los estados del codificador junto con el enrejado sugiere una tercer representación gráfica de la operación de cualquier código convolucional. Se trata del **diagrama de estado**. Esta tiene un conjunto de estados definidos, un conjunto de entradas, y un conjunto de salidas cada una de las cuales depende del estado y de la entrada. Además la secuencia de estados del codificador es un proceso en que cada transición de estado depende del estado corriente y no de cualquier estado previo; ésto es, la forma en que llegó el codificador a su estado presente no tendrá efecto en estados futuros.

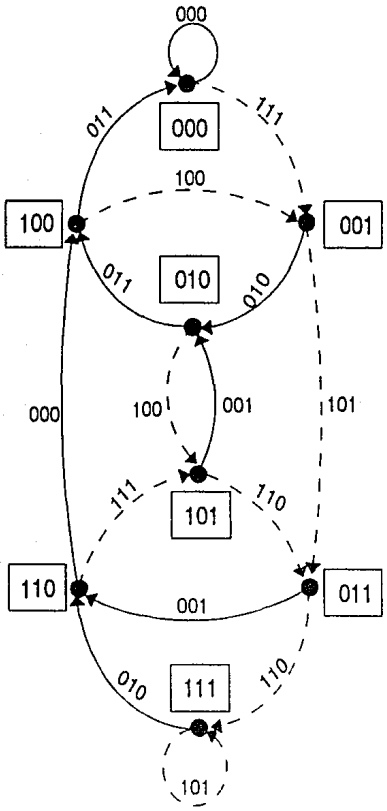


Figura 2.8 Diagrama de estados correspondiente al codificador de la figura 2.5.

La figura 2.8, corresponde al código que se ha discutido con las figuras 2.5 a 2.7. Cada nodo del diagrama de estados representa un estado único del codificador. Las transiciones de estado se indican mediante segmentos de líneas dirigidas que van de un nodo a otro.

Nótese que para los estados **000** y **111**, el codificador puede permanecer en el mismo estado, dado por la entrada apropiada. En todos los casos el segmento de línea se etiqueta con la salida dada por el codificador para la entrada, indicada por el tipo de línea (punteada=1, firme=0), y la transición de estado correspondientes. Como anteriormente, el *bit* más reciente en el registro es el que está a la izquierda en los *bits* que designan al estado.

Representación analítica de los códigos convolucionales[4]

Además de las representaciones gráficas que se han mostrado anteriormente para códigos convolucionales, existen dos representaciones analíticas ampliamente utilizadas. Ellas son a saber: el **operador de retardo polinomial** y la **matriz de dimensión infinita** (o semi infinita).

Para la representación mediante el operador de retardo, tanto las conexiones del codificador como las secuencias de datos a codificar, se representan por polinomios en **D** (el operador de retardo).

Por ejemplo, la secuencia de entrada **1101001**, siendo el último *bit* el de la izquierda, se escribe mediante el operador de retardo como:

$$\mathbf{I(D)} = 1 \oplus \mathbf{D} \oplus \mathbf{D}^3 \oplus \mathbf{D}^6,$$

siendo el exponente de la variable **D** el número de unidades de tiempo discreto que se retarda el *bit* respecto al origen de tiempo elegido, que por costumbre se toma como el primer *bit* de entrada al descodificador. En general, la secuencia de entrada:

$$\mathbf{I} = i_0\, i_1\, i_2\, i_3\, \ldots$$

tiene la representación polinomial:

$$\mathbf{I(D)} = i_0 + i_1\mathbf{D} + i_2\mathbf{D}^2 + i_3\mathbf{D}^3 + \ldots,$$

ésta representación también se llama **transformada** de **I**.

Para el codificador, una conexión de una etapa del registro de desplazamiento del mismo hacia los sumadores módulo 2, significa un coeficiente igual a 1 en los términos correspondientes del polinomio que representa al codificador, mientras que en este mismo las etapas del registro que no están conectadas significa un coeficiente igual a 0.

Así, los conjuntos de conexiones de las etapas $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$ del codificador de la figura 2.5 hacia los sumadores módulo 2 son respectivamente:

$$G_1(D) = 1$$

$$G_2(D) = 1 \oplus D \oplus D^2 \oplus D^3 \qquad (2.25)$$

$$G_3(D) = 1 \oplus D^2 \oplus D^3$$

En estas ecuaciones el grado más bajo de los polinomios corresponde a la primer etapa en que se encuentra la cadena de *bits* que entra al codificador. La ecuación (2.25) está constituida por los llamados **polinomios generadores** para calcular la secuencia de salida de cada sumador del codificador simplemente se multiplican los polinomios generadores con el polinomio de la secuencia de entrada. Así, los polinomios se pueden ver también como una transformada en tiempo discreto de la respuesta al impulso del codificador conocida también como secuencia generadora.

Por ejemplo, sea la secuencia de entrada al codificador **1101...**, cuya transformada es $I(D)= 1 \oplus D \oplus D^3 \oplus ...$; las secuencias a la salida de cada sumador estarán dadas por:

$$V_1(D) = I(D)G_1(D) = I(D)$$

$$V_2(D) = I(D)G_2(D) = (1 \oplus D \oplus D^3 \oplus ...)(1 \oplus D \oplus D^2 \oplus D^3)$$

$$V_2(D) = 1 \oplus D_3 \oplus D^5 \oplus D^6 ...$$

$$V_3(D) = I(D)G_3(D) = (1 \oplus D \oplus D^3 \oplus ...)(1 \oplus D^2 \oplus D^3)$$

$$V_3(D) = 1 \oplus D \oplus D^2 \oplus D^4 \oplus D^5 \oplus D^6 ...$$

obsérvese también que la transformada de la secuencia de salida es el producto de las transformadas, de la secuencia de entrada con la secuencia generadora, tal como ocurre en la teoría de la transformada z, de Fourier, y de Laplace.

En la representación de códigos convolucionales por medio de la matriz de dimensión infinita se utilizan vectores y matrices que tienen un conjunto de elementos infinito. Por analogía con los códigos de bloque ésta representación será de la forma $\underline{c}=\underline{d}*\underline{G}$, luego las matrices renglón o vectores dato son expresiones de la forma:

$$\underline{d} = (d_1\ d_2\ d_3\ ...),$$

donde los elementos del vector $\underline{d}$ continúan indefinidamente hacia la derecha. Sea el vector transmitido de la forma:

$$\underline{c} = (c_{11}\ c_{12}\ c_{13}\ c_{21}\ c_{22}\ c_{23}\ c_{31}\ c_{32}\ c_{33}\ c_{41}\ c_{42}\ c_{43}...).$$

Analizando ahora la forma $\underline{c}=\underline{d}*\underline{G}$ para obtener el contenido de $\underline{G}$. Basándose en que la secuencia $\underline{d}$ ingresa al codificador de la figura 2.5, las expresiones para los tres primeros símbolos transmitidos son:

$$c_{11} = d_1$$

$$c_{12} = d_1$$

$$c_{13} = d_1,$$

ahora se desplaza $\underline{d}$ una etapa del registro de desplazamiento para obtener:

$$c_{21} = d_2$$

$$c_{22} = d_1 \theta d_2$$

$$c_{23} = d_2 ,$$

y haciendo un nuevo desplazamiento de la secuencia de entrada se obtiene:

$$c_{31} = d_3$$

$$c_{32} = d_1 \theta d_2 \theta d_3$$

$$c_{33} = d_1 \theta d_3 ;$$

finalmente se obtiene el estado del descodificador en que todas sus etapas contienen *bits* de información:

$$c_{41} = d_4$$

$$c_{42} = d_1 \theta d_2 \theta d_3 \theta d_4$$

$$c_{43} = d_4 \theta d_2 \theta d_1$$

Se debe notar que en este instante de tiempo discreto en el que todas las N=4 etapas del registro contienen *bits* de información, se considera que el codificador está en una etapa llamada "estado estable", mientras que el intervalo de (N-1)=3 instantes de tiempo que precedieron esta etapa se le conoce como "transitorio inicial". En general, la salida de cada registro, para el estado estable, está dada por:

$$c_{m1} = d_m$$

$$c_{m2} = d_m \theta d_{m-1} \theta d_{m-2} \theta d_{m-3} \qquad (2.26)$$

$$c_{m3} = d_m \theta d_{m-2} \theta d_{m-3}$$

Sea $\underline{A_1}$ una submatriz de $\underline{G}$ que involucre la ecuación (2.26) y por tanto satisfaga la ecuación (2.27):

$$|d_{m-3}\ d_{m-2}\ d_{m-1}\ d_m| * \underline{A}_1 =$$

$$|d_m,\ d_m \theta d_{m-1} d_{m-2} \theta d_{m-3},\ d_m \theta d_{m-2} \theta d_{m-3}| \qquad (2.27)$$

entonces $\underline{A}_1$, por inspección de (2.27), está definida como:

$$A_1 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Ahora queda analizar el transitorio inicial donde se observa que d_1, ..., d_4 se desplazan por el registro de corrimiento y que cuando d_1 ingresa al mismo registro, solamente la etapa M_1 influye sobre la salida del codificador. En el siguiente desplazamiento M_1 y M_2 influyen sobre la salida del codificador, y así hasta que al ingresar d_4 las cuatro etapas pueden influir sobre la salida del registro de corrimiento.

Por lo anterior se definen las submatrices $\underline{T}_1$, $\underline{T}_2$, ..., $\underline{T}_{N-1}=\underline{T}_3$ que describen este comportamiento transitorio (compárense los renglones de $\underline{T}_i$, i=1,2,3, con los de $\underline{A}_1$):

$$\underline{T}_1 = \begin{bmatrix} 111 \\ 000 \\ 000 \\ 000 \end{bmatrix} \qquad \underline{T}_2 = \begin{bmatrix} 010 \\ 111 \\ 000 \\ 000 \end{bmatrix} \qquad \underline{T}_3 = \begin{bmatrix} 011 \\ 010 \\ 111 \\ 000 \end{bmatrix}$$

De esta forma se concluye que $\underline{G}$ está definida como:

$$\underline{G} = \begin{bmatrix} \underline{T}_1 & \underline{T}_2 & \underline{T}_3 & \underline{A}_1 & 000 & 000 & 000 & \\ & & & & A_1 & 000 & 000 & \ldots \\ & & & & & \underline{A}_1 & 000 & \\ & & & & & & \underline{A}_1 & \end{bmatrix}$$

o bien:

$$\underline{G} = \begin{bmatrix} 111 & 010 & 011 & 011 & 000 & 000 & \\ 000 & 111 & 010 & 011 & 011 & 000 & \\ 000 & 000 & 111 & 010 & 011 & 011 & \\ 000 & 000 & 000 & 111 & 010 & 011 & \ldots \\ & & & & 111 & 010 & \\ & & & & & 111 & \end{bmatrix}$$

donde los óvalos grandes indican elementos cero, y la matriz $\underline{G}$ se extiende indefinidamente hacia abajo y hacia la derecha. En la práctica $\underline{G}$ representa una matriz generadora en el mismo sentido que para los códigos de bloque. Esto es, cualquier secuencia codificada (transmitida) puede ser representada como una combinación lineal de dos o más renglones de $\underline{G}$. Luego entonces, con la matriz generadora dada, se genera el árbol del código.

El proceso de generación del árbol del código puede invertirse dado que existe una correspondencia uno a uno entre $\underline{G}$ y el árbol del código, lo cual abre la posibilidad de obtener $\underline{G}$ por un método diferente al ya descrito. De la figura 2.6 se observa fácilmente que al seguir por la trayectoria del árbol del código que representa el mensaje de puros ceros, cada una de sus ramas tiene el triple **000** asociado, y que si en cualquier etapa del árbol se elige por fin una rama correspondiente al *bit* dato 1, a ésta rama siempre le corresponderá el triple **111**.

Por ejemplo, la secuencia de datos **1010** se representa en el árbol con la trayectoria cuyos triples son **111 101 001 111**; la secuencia de datos **01010** se representa en el árbol con la trayectoria cuyos triples son **000 111 101 001 111** que resulta ser la misma secuencia anterior pero desplazada tres *bits* a la derecha. Se tendrán más desplazamientos similares al anterior si se codifican las secuencias de datos **001010**, **0001010**, etc. Las secuencia despalazadas que se obtienen, **000 000 111 101 001 111** y **000 000 000 111 101 001 111**, en conjunto con las dos anteriores sugieren ser utilizadas como renglones de $\underline{G}$, al acomodarse de la siguiente manera:

111 101 001 111
000 111 101 001 111
000 000 111 101 001 111
000 000 000 111 101 001 111,

y en vista de que todas las secuencias del código representadas en el árbol pueden producirse a partir de un pequeño conjunto de vectores renglón.

Para determinar la válidez de lo dicho en el párrafo anterior se tomará como vectores base el conjunto de secuencias código descritas anteriormente, ésto es:

$\underline{a}$ = 111 101 001 111 ... ...

$\underline{b}$ = 000 111 101 001 111 ... ...

$\underline{c}$ = 000 000 111 101 001 111 ... ...

$\underline{d}$ = 000 000 000 111 101 001 111 ...,

donde los puntos suspensivos indican que los elementos que continuan son ceros.

Haciendo referencia a la fig. 2.6, se hacen las siguientes asignaciones: llámese la trayectoria que va del origen del árbol al nodo A: trayectoria #1; llámese la trayectoria que va del origen del árbol al nodo B: trayectoria #2;, ..., llámese la trayectoria que va del origen del árbol al nodo H': trayectoria #16. Ahora tómese los primeros n_0N=12 *bits* de cada uno de los vectores **a**, **b** , **c**, y **d** y compárese contra cada una de las dieciseis trayectorias del árbol a las que se les asignó un número. Se observa que existen las siguientes igualdades:

a = trayectoria #11

b = trayectoria #6

c = trayectoria #3

d = trayectoria #2.

Bajo la misma mecánica se comprobará si las combinaciones lineales de los vectores **a**, **b**, **c**, y **d** producen las once trayectorias restantes en el árbol (diferentes de la trayectoria cero), ésto es, si tiene validez el hecho de tomar trayectorias desplazadas, en el sentido visto ante- riormente, como renglones de la matriz **G**. Se tiene así que las combinaciones lineales de los vectores base truncados hasta el doceavo *bit* son:

a θ **b** θ **c** θ **d**	= 111 101 001 000
a θ **b** θ **c**	= 111 101 001 111
a θ **b** θ **d**	= 111 101 110 010
a θ **c** θ **d**	= 111 010 011 100
b θ **c** 0 **d**	= 000 111 101 001
a θ **b**	= 111 101 110 101
a θ **c**	= 111 010 011 011
a θ **d**	= 111 010 100 110
b θ **c**	= 000 111 101 110
b θ **d**	= 000 111 010 011
c θ **d**	= 000 000 111 101.

Al comparar las once secuencias resultantes de las once combinaciones lineales de los vectores base con las once trayectorias restantes en el árbol se tiene que corresponden respectivamente a las trayectorias #13, #14, #15, #10, #7, #16, #9, #12, #8, #5, y #4, con lo que concluye la prueba.

Propiedades de distancia de Códigos Convolucionales[4][5]

Para códigos de bloques se define el concepto de distancia entre dos palabras código como una forma de distinguir las palabras codificadas y las palabras que no pertenecen al código. La distancia entre dos palabras código es también un caso particular de la métrica del código la cual se define de la forma más conveniente para medir el dempeño del código. La métrica más popular para códigos de bloque es la distancia Hamming. En el apartado 2.1 se vió que la distancia mínima del código, que es la ditancia Hamming mínima entre dos secuencias código, guarda una relación directa con la capacidad de corrección del código, ecuación (2.1), y de esta relación se sigue que es deseable que la $d_{mín}$ tenga un valor máximo posible.

La situación anterior prevalece para códigos convo- lucionales dado que también se definen para éstos dos tipos de distancia entre dos secuencias código igualmente útiles e importantes que para el caso de la codificación por bloques;la diferencia estriba en que las distancias se definen para dos secuencias de una longitud dada cuyos *bits* de información inicial i_0 son diferentes uno respecto del otro.

Por un lado está la distancia definida para secuencias codificadas de longitud n_0xN *bits* que corresponden a una secuencia de información que tiene una longitud límite **N**. El menor valor de todas las distancias posibles entre las diferentes secuencias es la distancia mínima **dmín** del código convolucional.

Por otro lado se desarrolló un concepto de distancia llamado **distancia libre** que es particularmente útil para códigos convolucionales en diferentes técnicas de descodificación convolucional. La distancia libre se define como el número mínimo de *bits* diferentes entre dos secuencias código arbitrariamente largas, posiblemente infinitas, de las cuales una de ellas tiene su *bit* de información inicial i_0=0 y la otra tiene su *bit* de información inicial i_0=1 y que se unen en una sola trayectoria lo más rápido posible como puede verse en el diagrama de enrejado (figura 2.7, pág. 79); aún siendo ambas trayectorias infinitas la distancia libre puede ser finita como se verá más adelante. Definir la distancia libre y la distancia Hamming entre dos palabras código tomando en cuenta el valor del *bit* de información inicial es conveniente dado que uno de los valores del *bit* inicial representa el valor descodificado correcto, mientras que el complemento será el valor descodificado incorrecto.

Sin pérdida de generalidad, se elige a i_0=0 como el valor correcto. Así se define a S_c como el subconjunto correcto y a S_i como el subconjunto incorrecto, y ambos forman el conjunto de todas las secuencias codificadas cuyo inicio corresponde a i_0=0 e i_0=1 respectivamente. Por linealidad se sigue que una secuencia que resulta de la diferencia entre una secuencia que pertenece a S_c y una secuencia que pertenece a S_i, pertenece también a S_i ya que corresponde a una secuencia cuyo *bit* i_0 es igual a 1.

De lo anterior se deriva que para determinar la distancia mínima y la distancia libre de un código basta con considerar las secuencias pertenecientes a S_i. De lo anterior y de lo mencionado en el apartado 2.2 es

fácil ver que la distancia mínima y la distancia libre son el número de elementos 1's en secuencias codificadas de S_i con longitudes n_0xN e infinita respectivamente. Como la distancia libre se define en función de una palabra codificada de longitud infinita, se vuelve un parámetro particularmente importante para el desempeño de códigos convolucionales que actúan conjuntamente con algoritmos de descodificación que hacen búsquedas en el árbol del código más allá de una longitud límite. Frecuentemente, en un código convolucional la distancia mínima es igual a la distancia libre aunque ésto no es forzosamente cierto para todos los códigos convolucionales dado que en general se cumple la desigualdad:

$$d_{mín} \leq d_{lib} \qquad (2.28)$$

Ejemplo: para ilustrar los conceptos de distancia libre y distancia mínima en la codificación convolucional se utilizará el diagrama de árbol de la figura 2.6 que es generado por el codificador de la figura 2.5, pág. 76.

De acuerdo a lo mencionado anteriormente para determinar $d_{mín}$ se enlistan a continuación los pesos de las secuencias de longitud N=4 ramas que corresponden a S_i, es decir, las que inician en el origen del árbol y terminan en los nodos A', B', C', D', E', F', G', y H' de la figura 2.6. Sea w(*) el símbolo para indicar el peso de la secuencia * del código, se observa de la figura 2.6 que:

$$w(A') = 8,$$

$$w(B') = 7,$$

$$w(C') = 6,$$

$$w(D') = 7,$$

$$w(E') = 6,$$

$$w(F') = 9,$$

$$w(G') = 8,$$

$$w(H') = 9;$$

de donde $d_{mín}=6$.

Ahora se calcula la distancia libre del código para lo cual cabe hacer la pregunta: ¿Puede obtenerse fácilmente?; cuya respuesta es: en éste caso sí. Observando que una secuencia código de longitud infinita cuya trayectoria correspondiente a partir de la rama N, contada a partir del origen del árbol, mantiene el valor de su peso constante dado que a partir de la rama N ésta toma el rumbo y coincide con la trayectoria cuyas ramas tienen asociado el triple 000 lo que equivale a decir que a partir de la rama N la trayectoria de

la secuencia código es la secuencia cero. En el caso de la figura 2.6 se observa que la trayectoria que cumple con ésto es la que se compone de la trayectoria A' y se continua al infinito por la rama superior que sale del nodo A'.

Sea $w(A',\infty)$ el peso mínimo que corresponde a cualquier trayectoria de longitud infinita que parte del nodo A', entonces $w(A',\infty)=0$.

Por otro lado los pesos mínimos $w(B',\infty)$, $w(C',\infty)$, ..., $w(H',\infty)$ que corresponden a las trayectorias infinitas que comienzan en los nodos B', C', ..., H' respectivamente, se calculan fácilmente tomando como referencia la figura 2.7 y haciendo corresponder los nodos B' a H' del árbol de la figura 2.6, con sus nodos equivalentes en la etapa cuatro del enrejado de la fig. 2.7; de ésta manera se calcula:

$$w(B',\infty) = w\{\ 010, 011, 011\} = 5,$$

$$w(C',\infty) = w\{\ 011, 011\} = 4,$$

$$w(D',\infty) = w\{\ 001, 000, 011\} = 3,$$

$$w(E',\infty) = w\{\ 011\} = 2,$$

$$w(F',\infty) = w\{\ 001, 011, 011\} = 5,$$

$$w(G',\infty) = w\{\ 000, 011\} = 2,$$

$$w(H',\infty) = w\{\ 010, 000, 011\} = 3.$$

A partir de éstos se calcula el peso mínimo de cada trayectoria siguiendo la trayectoria cuyas ramas tienen el menor número de 1's en sus correspondientes triples, y que se dirigen lo más directamente posible hacia la trayectoria cero, esto es, la trayectoria cuyas ramas tienen como triple **000**.

Sumando estos pesos mínimos a los correspondientes pesos de las trayectorias que van del origen del árbol a los nodos A', B',..., H' ya calculados, se obtiene el peso total de cada trayectoria:

$$w(A') + w(A',\infty) = 8 + 0 = 8,$$

$$w(B') + w(B',\infty) = 7 + 5 = 12,$$

$$w(C') + w(C',\infty) = 6 + 4 = 10,$$

$$w(D') + w(D',\infty) = 7 + 3 = 10,$$

$$w(E') + w(E',\infty) = 6 + 2 = 8,$$

$$w(F') + w(F',\infty) = 9 + 5 = 14,$$

$$w(G') + w(G',\infty) = 8 + 2 = 10,$$

$$w(H') + w(H',\infty) = 9 + 3 = 12.$$

Así el peso mínimo de la secuencia de longitud infinita que pasa por A' y E' el peso mínimo es w = 8 y que para otras trayectorias $W \geq 8$ por lo que d_{lib} =8. En este código se cumple la condición dada en (2.28) para la desigualdad.

Según la ecuación (2.1) es deseable que la $d_{mín}$ de un código tenga un valor que sea el máximo posible, y como en general su búsqueda resulta un trabajo tedioso y complicado peor aún si éste se realiza como en el ejemplo anterior, se han encontrado métodos sistemáticos e inteligentes para reducir el esfuerzo en la búsqueda de estos códigos, y támbien para reducir el número de códigos para examinar y obtener el código óptimo en el sentido de tener $d_{mín}$ o d_{lib} máximos.

Cotas de distancia para Códigos Convolucionales[4]

Las cotas superior e inferior de distancia mínima para un códigos convolucional con tasa $R=k_0/n_0$ y longitud límite n_0N se obtienen por fórmula y sus características son similares a las de los códigos de bloque.

Desafortunadamente, las cotas superior e inferior no son en general muy confiables, por lo que su utilidad se reduce a dar una idea de que tan bueno es un código en función de que tanto se aproxima su distancia mínima a los valores dados por las cotas, ésto es, en el código convolucional del ejemplo anterior tiene una $d_{mín}$ = 6; ésto implica que una cota superior es buena si es mayor que $d_{mín}$ y entre mas próxima sea la cota superior a $d_{mín}$ será mejor, en tanto que una cota inferior debe estar por debajo del valor de $d_{mín}$ pero ésta será mejor si su valor es muy próximo a $d_{mín}$.

A continuación se dan tres cotas válidas para códigos binarios y se calcula el valor de cada una para el codificador de la fig 2.5:

A. Cota de Gilbert[4] *(inferior)*

Sea H(x) la entropía para un alfabeto binario que se evalúa como:

$$H(x) = -x\log_2 x-(1-x)\log_2(1-x),$$

Sea el código de tasa R, longitud límite n_0N y d el número entero más grande posible que satisface

$$H(d/n_0N) \leq 1-R \qquad \textbf{(2.29)}$$

Entonces existe al menos un código convolucional binario con distancia mínima d para la cual (2.29) es válida.

B. Cota de Plotkin[4] *(superior)*

Sea el código binario de tasa $R=1/n_0$; $d_{mín}$ satisface la desigualdad:

$$d_{mín} \leq [(1/2)(n_0N + n_0)] \qquad \textbf{(2.30)}$$

donde [ ] indica la parte entera de la cantidad que contiene.

C. Cota de Heller[4][6] *(superior para* d_{lib}*)*

Sea el código de longitud límite n_0N, donde n_0 y N son parámetros fijos de un código dado, la cota de Heller está dada por la expresión:

$$d_{lib} \leq \min_{j \geq 1} [(n_0/2)(2^j/2^j-1)(N+j-1)], \qquad (2.31)$$

aquí también [ ] indica que se toma la parte entera de la cantidad que encierra, y min indica el valor mínimo sobre algun valor de $j \geq 1$.

Como ejemplo, para el códificador de la figura 2.5 se tienen los siguientes valores: $n_0=3$, $N=4$, $R=1/3$, $d_{mín}=6$, y $d_{lib}=8$ con los que se calculan las tres cotas anteriores:

A. Sustituyendo en (2.29): $H(d/12) \leq 2/3$

de valores ya calculados[6] se tiene:

$$H(0.1735) = 0.6656 \text{ y,}$$

$$H(0.1740) = 0.6668$$

de donde : $(d/12) \leq 0.1740$

y : $d \leq 2.088$

luego el mayor valor entero de d que satisface (2.29) implica que $d_{mín} \geq 2$, luego la cota inferior es 2.

B. Sustituyendo en (2.30): $d_{mín} \leq [(1/2)(12+3)]$

$$d_{mín} \leq 7$$

como está por encima y próxima a $d_{mín}$ calculada ==> buena cota superior.

C. tabulando (2.31) para j=1, 2, ..., 5:

j	[*]
1	12
2	10
3	10
4	11
5	14

de donde: $d_{lib} \leq 10$

Propagación de errores[4]

El desempeño de un código convolucional no depende solamente del esquema de descodificación que se esté utilizando en el receptor sino también de las propiedades de distancia como se ha dicho en los párrafos anteriores.

Por otro lado, la representación de un código convolucional mediante el operador de retardo permite representar las conexiones del registro de desplazamiento del codificador a los sumadores de salida del mismo mediante polinomios llamados polinomios generadores que además muestran ciertas características del código que definen.

Es deseable en algunos esquemas de descodificación que la labor del codificador favorezca el proceso de decisiones correctas sobre las secuencias recibidas que se están descodificando. Contrario a esto, la naturaleza de algunos códigos causa que las decisiones que toma el descodificador sobre las secuencias recibidas sean incorrectas, y más aún que al recibir secuencias correctas se decida que estas son erroneas; los códigos que tienen ésta propiedad son conocidos como **códigos catastróficos**. Existe una prueba para saber si un código convolucional es catastrófico y está dada en función de las propiedades de los polinomios generadores: un código convolucional es catastrófico si y sólo si sus polinomios generadores contienen un factor común de grado mayor o igual a 1. De lo anterior se observa que los codigos sistemáticos no son catastróficos ya que uno de sus generadores es de grado cero(figura 2.9).

El Algoritmo de descodificación de Viterbi[3][4]

El análisis del Algoritmo de Viterbi se hará en base al codificador convolucional sistemático de R=1/3 y N=4 que se muestra en la figura.

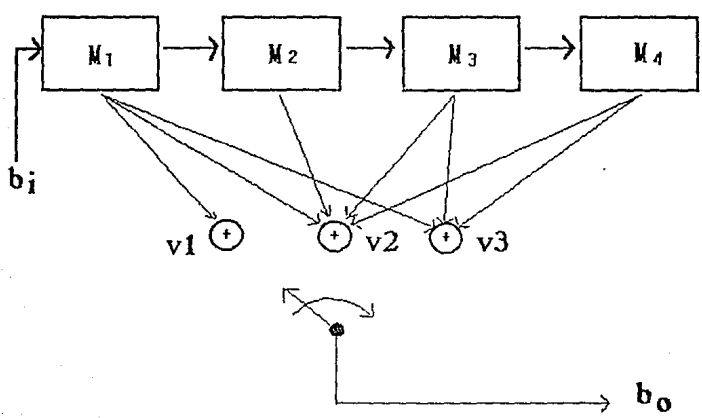


Figura 2.9 Códificador convolucional sistemático de R=1/3 y N=4.

El algoritmo de descodificación de Viterbi es una técnica de descodificación rápida, poderosa y de fácil implantación para descodificar códigos convolucionales. Además, su simulación y análisis es relativamente fácil de realizar. Para este algoritmo la cantidad de cálculos para la descodificación es independiente del ruido adicionado en los símbolos del canal que entran al receptor.

También se reproduce nuevamente su enrejado en la figura 2.10 porque el algoritmo de Viterbi se basa en encontrar la trayectoria transmitida más probable trazada en el enrejado. Obsérvese que el enrejado caracteriza el hecho de que no existe dependencia entre símbolos codificados que distan uno del otro más alla de una longitud límite.

Por ejemplo, si la secuencia de *bits* **11011**... entra al codificador de la figura 2.9 entonces los primeros cuatro *bits* de la secuencia de entrada producirán la siguiente salida del codificador **111 101 001 111**; luego, al desplazar el quinto *bit* de la secuencia de entrada hacia el codificador, éste quedará en el estado **011** y el triple de salida **110** producido por el codificador debido al desplaza-miento del quinto *bit* de la secuencia de entrada al codificador no depende de que el valor del primer *bit* de la secuencia de entrada haya sido 0 o 1. Dicho de otro modo, el nodo del enrejado, indicado por una flecha en la figura 2.10, se alcanza por cualquiera de las siguientes secuencias de entrada **00011**, **01011**, **10011**, y **11011**.

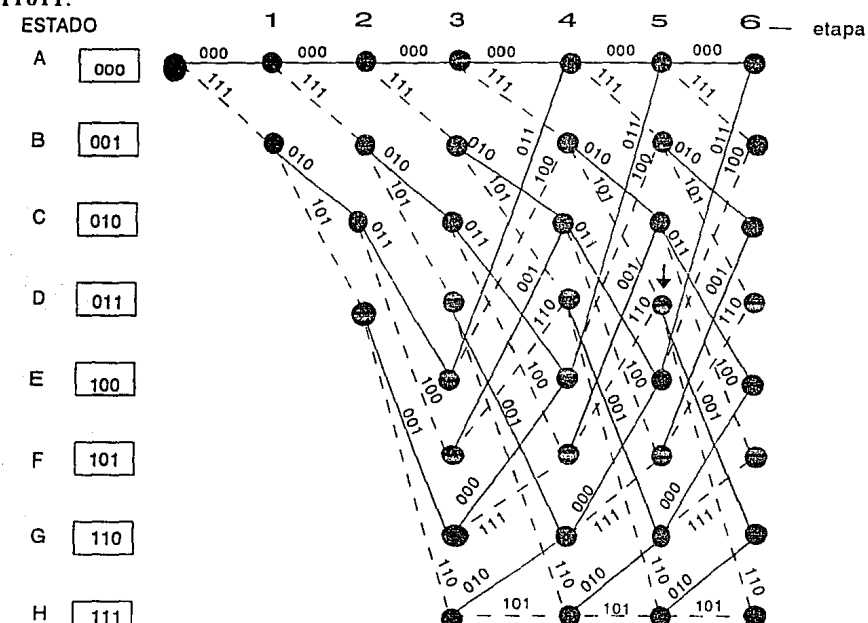


Figura 2.10 Enrejado del codificador convolucional de R=1/3 y N=4.

Ahora el objetivo es determinar la secuencia transmitida que es similar en algún sentido a la secuencia recibida como se hace en el caso de los códigos de bloque.

Operación del descodificador de Viterbi[4][5]

El descodificador de Viterbi tiene en común con otras técnicas de descodificación generar una secuencia supuesta cuya versión codificada es muy similar a la secuencia recibida. Para el análisis se utilizará como medida de similitud entre dos secuencias la distancia de Hamming. Supóngase ahora que se transmitió una secuencia de ceros y que la secuencia que se recibió fué la siguiente:

010 000 100 001 000

Trátese de descodificar la secuencia transmitida utilizando el enrejado del codificador y la distancia de Hamming, teniendo en mente que se trabaja con un código sistemático. Por conveniencia, se ha redibujado en la figura 2.10 el enrejado para indicar la distancia entre los triples correspondientes de las secuencias de prueba y la secuencia recibida mostrando esta última en la parte inferior del enrejado.

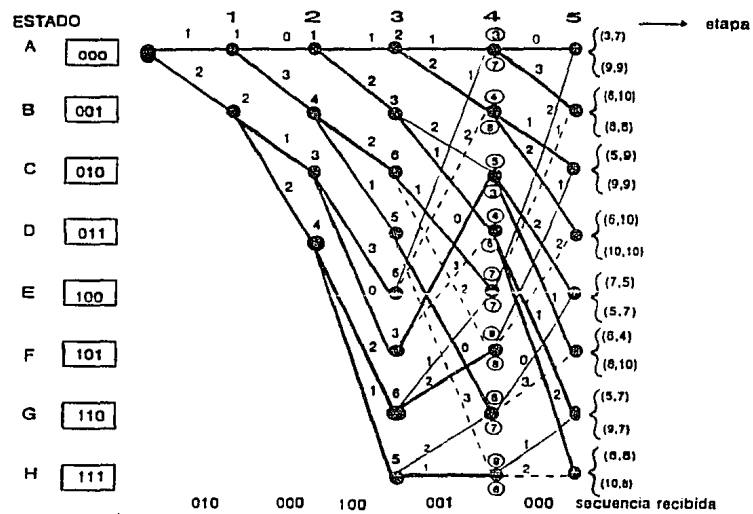


Figura 2.11 Enrejado del codificador de la figura 2.9 indicando las distancias acumuladas.

La figura 2.11 muestra el enrejado indicando en los nodos de cada etapa las distancias acumuladas entre la secuencia recibida (hasta esa etapa) y la secuencias supuestas cuyas trayectorias correspondientes en el árbol parten del origen del mismo y terminan en los nodos de la etapa que se está observando. Como se puede observar a partir de la etapa cuatro del

enrejado hay más de una trayectoria posible, que partiendo del origen, pueden alcanzar un mismo nodo. Para caracterizar este hecho se han encerrado en un círculo las dos distancias acumuladas posibles para cada nodo de la etapa cuatro y puesto entre paréntesis las cuatro distancias posibles para cada nodo de la etapa cinco.

De acuerdo a lo anterior nótese en la figura 2.11 que las primeras ramas supuestas **000** y **111** están a una distancia de la "rama" recibida **010** en 1 y 2 respectivamente; las distancias entre el segundo triple recibida **000** y los cuatro posibles triples o ramas supuestas **000, 111, 010,** y **101** son 0, 3, 1, y 2 respectivamente y las cuatro secuencias **000 000, 000 111, 111 010, 111 101,** de *bits* digitos de longitud cada una, posiblemente transmitidas están a distancias 1, 4, 3, y 4 respectivamente de los primeros seis digitos recibidos; para la tercer "rama" recibida **100** hay ocho posibles ramas supuestas **000, 111, 010, 101, 011, 100, 001,** y **110** y que cada una está a las correspondientes distancias 1, 2, 2, 1, 3, 0, 2, y 1, indicadas de arriba a abajo sobre cada rama que está entre las etapas 2 y 3 del árbol en la fig.2.11, de la tercer "rama" recibida, además cada una de las ocho secuencias posiblemente transmitidas de nueve digitos de longitud: **000 000 000, 000 000 111, 000 111 010, 000 111 101, 111 010 011, 111 010 100, 111 101 001, 111 101 110,** están respectivamente a distancias 2, 3, 6, 5, 6, 3, 6, y 5 de los primeros nueve digitos recibidos, indicadas en los nodos de la etapa 3 fig. 2.11. Hasta esta tercer etapa existe solamente un camino o trayectoria para llegar a cada nodo y hasta aquí se han analizado todas las secuencias de tres ramas (en general **N**-1 ramas) posiblemente transmitidas. Para nodos de la siguiente etapa, **N**=4, se observa algo que ya se mencionó y es que los nodos pueden ser alcanzados por más de una trayectoria, lo que produce en general que haya dos distancias acumuladas distintas para cada nodo de la etapa cuatro, una distancia para cada trayectoria que va desde el origen del enrejado al nodo. Al llegar a la cuarta etapa el descodificador tiene que hacer por primera vez una decisión sobre que trayectoria elegir para continuar más adelante; de no hacer esta decisión en la etapa **N** o en las etapas posteriores a ésta, el número de trayectorias posibles crecerá exponencialmente al igual que el número de distancias acumuladas como se puede apreciar en la figura 2.11 y en consecuencia se tendrá que los requerimientos de memoria y cálculos de descodificación crecerán en la misma proporción. Sin embargo, este crecimiento estaría limitado por un retardo en la descodificación, que es la longitud de una secuencia procesada antes de que el descodificador lleve a cabo una decisión.

En base a la discusión anterior, ahora se abordará el punto clave del descodificador de Viterbi. Comenzando con la **N**-ésima rama contando a partir del origen, en el ejemplo dado aquí es la cuarta rama, el descodificador retendrá para cada nodo solamente una de las ramas que dejan al nodo de la etapa 3. La rama que se conserva será la que dé la menor distancia acumulada, con respecto a la trayectoria recibida, de la trayectoria que va del origen del enrejado hasta el nodo de la etapa 3 por el que sale dicha rama, esto es, se escoge la que tenga la **dmin** entre la trayectoria supuesta y la recibida. Este proceso de descartar una de dos ramas, en cada nodo de una etapa, a partir de la etapa 3 mantendrá constante e igual a 2^{N-1} (=8 en el ejemplo) el número de trayectorias que se extienden a la siguiente etapa. Además, Viterbi demostró que este proceso es óptimo en el sentido de producir una decisión de máxima verosimilitud respecto a la trayectoria correcta. Para el ejemplo, en la figura 2.11 se

indica con línea gruesa aquellas trayectorias que serán retenidas para llegar de la etapa 3 a la etapa 4, lo mismo de la etapa 4 a la etapa 5 del enrejado. Viterbi llamó **sobrevivientes** a estas trayectorias. Obsérvese lo siguiente: las trayectorias que llegan al nodo (E,4) tienen una distancia acumulada idéntica: 7, lo mismo ocurre con el nodo (F,4) en el que la distancia acumulada es de 8 para las dos trayectorias que llegan a él; en estos casos Viterbi elige arbitrariamente cualquiera de las dos trayectorias que llegan al nodo en cuestión dado que para ambas trayectorias la distancia acumulada en la siguiente etapa será la misma.

Ahora se analiza lo que ha realizado el descodificador hasta aquí. A la etapa 4 existen ocho sobrevivientes con las distancias acumuladas respecto a la secuencia recibida y el estado del codificador correspondiente que se muestran:

Secuencia supuesta	distancia	estado (etapa 4)
000 000 000 000	3	A
000 000 000 111	4	B
000 000 111 101	4	D
000 111 010 011	7	E
000 111 101 001	5	G
111 101 001 111	8	F
111 101 110 101	6	H
111 010 100 001	3	C

Es interesante observar que una de las secuencias que tiene la menor distancia viene a ser la secuencia realmente transmitida.

Continuando con el proceso de descodificación las ramas que llegan a la etapa 5 como sobrevivientes se indicaron con línea gruesa. Obsérvese también que los sobrevivientes de los estados E, F, G, y H en la etapa 4 no sobreviven para la etapa 5. En cambio los sobrevivientes de los estados A, B, C, y D en la etapa 4 producen dos sobrevivientes cada uno para la etapa 5. Hasta esta etapa los detalles de distancias acumuladas son:

Secuencia supuesta	distancia	estado (etapa 5)
000 000 000 000 000	3	A
000 000 000 000 111	6	B
000 000 000 111 010	5	C
000 000 000 111 101	6	D
000 000 111 101 001	5	G
000 000 111 101 110	6	H
111 010 100 001 011	5	E
111 010 100 001 100	4	F

Las dos tablas anteriores muestran que hay dos cantidades que el descodificador debe "recordar": la trayectoria sobreviviente y su distancia acumulada respecto a la secuencia recibida para cada nodo y en cada nivel del enrejado.

Ahora surje la siguiente pregunta: ¿como obtener las mejores estimaciones acerca de que *bits* fueron realmente transmitidos? Una forma es producir una decisión en cada etapa del enrejado comenzando con la N-

ésima etapa (cuarta para el ejemplo). Una vez que se ha tomado la decisión, el descodificador puede borrar de su memoria las 2^{N-1} ramas en la posición del *bit* ya descodificado en el enrejado; mediante éste método la salida obvia del descodificador es el *bit* más antiguo del enrejado que tiene la distancia más pequeña respecto a la secuencia recibida, en la tabla anterior estos *bits* se han marcado en negrita sobre la trayectoria óptima. O sea, la de distancia mínima, dando por resultado la secuencia descodificada **00000** qué es la originalmente transmitida. Nótese que en el ejemplo se analiza un código sistemático, pero la descodificación de Viterbi no representa dificultades mayores para un código no-sistemático ya que se etiqueta cada rama del árbol de acuerdo a si fué producida por un *bit* de información cero o uno.

Otro modo de descodificar con este algoritmo es no tomar ninguna decisión hasta que el mensaje codificado se haya recibido por completo; este método depende por supuesto de la disponibilidad de memoria en el descodificador. En este segundo método el codificador y el descodificador se "limpian" insertando N-1 ceros sucesivos en el codificador después de que éste haya codificado el mensaje a transmitir, lo cual obliga al descodificador a regresar a la trayectoria cero (cuyas ramas son puros ceros) conforme la secuencia insertada se va procesando. Esta operación asume que el descodificador conoce el número total de *bits* a ser transmitidos lo cual resulta ser a menudo de interés práctico.

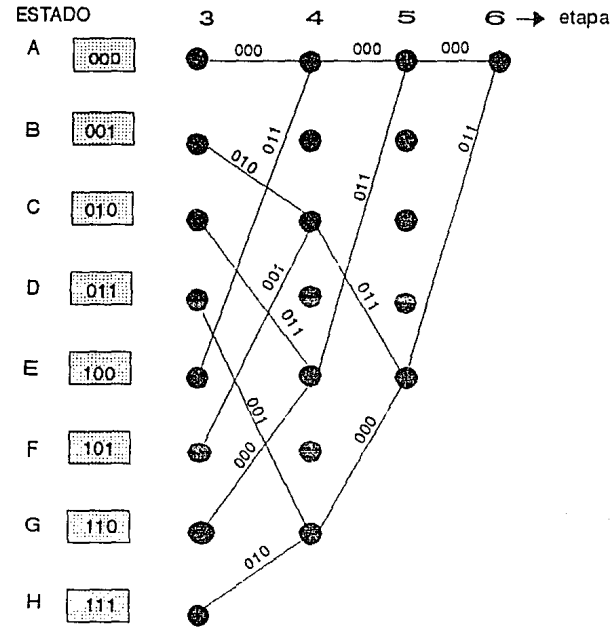


Figura 2.12 Enrejado truncado por la entrada 000 al codificador del ejemplo.

El enrejado del ejemplo aquí tratado se trunca, como se indica en la figura 2.12, conforme los tres ceros de limpiado ingresan al codificador; nótese que el efecto de cada cero ingresado al codificador es reducir a la mitad el número de ramas de la etapa correspondiente.

Parámetros del Descodificador de Viterbi[4][5]

De la discusión anterior se concluye que el descodificador recorre el enrejado a una velocidad fija. Para operaciones en tiempo real se necesita una velocidad de operación del descodificador 2^N veces mayor que la tasa de datos del canal, ya que el descodificador debe procesar 2^N ramas en cada etapa y descartar la mitad de ellas. Dependiendo del retardo de descodificación y tomando en cuenta que se tienen N=4 ramas por cada trayectoria de longitud límite y que se deben retener 2^{N-1} trayectorias de prueba, entonces deberá tenerse una cantidad de memoria suficiente para almacenar algún múltiplo de $N*2^{N-1}$ =32 pares (en el ejemplo anterior) de ramas y valores de distancia.

Con los parámetros anteriores es posible percatarse de la rapidez que tiene y la cantidad de memoria que necesita el algoritmo de máxima verosimilitud para códigos convolucionales, parámetros que son importantes en las consideraciones que se deben hacer al realizar la implantación física del descodificador.

REFERENCIAS

[1] - Taub, Herbert; Schilling, Donald
Principles of Communication Systems
Ed. Mc Graw Hill, 1986

[2] - Clark, George Jr; Cain, J. Bibb
Error Correction Coding for Digital Communications
Aplication of comm. Theory Series Editor: R. W. Lucky
Bell Laboratories, 1981

[3] - Haykin, Simon
Digital Communications
Ed.John Wiley & sons, 1988

[4] - Wiggert, Djimitri
Error-Control Coding and Applications
Artech House, INC. Dedham, Massachusets

[5] - Heller, Jerrold A. and Jacobs, Irwin Mark
Viterbi Decod. for Satellite and Space Communication
IEEE: Transactions on Communication Technology.
Vol COM-19, no. 5 oct 1971
pp. 835-847

[6] - Fano R. M.
Transmission of Information
The MIT Press, Cambridge, MA., and John Wiley & Sons
New York, 1961

CAPITULO 3

UN ALGORITMO DE DESCODIFICACION

3.1 Introducción

En este capítulo se presenta la operación del algoritmo de descodificación secuencial y el desarrollo, definición y justificación del algoritmo que representa una alternativa al anterior para descodificar secuencias de dígitos binarios, que provienen de un codificador convolucional. Se mostrarán los resultados experimentales de sus capacidades de corrección. El algoritmo se ha desarrollado para el caso particular en que el codificador agrega dos *bits* de redundancia, pudiéndose extender a codificadores más complejos.

En cualquier transmisión digital siempre existe la posibilidad de que se cometa el error de tomar un símbolo por otro. Por ejemplo, en el caso de transmisión de dígitos binarios, es posible que al enviarse un "1" el detector considere que se envió un "0". Para reducir la probabilidad de introducir un error de éste tipo, existen diversas técnicas, todas ellas basadas en agregar información redundante.

Cuando la información redundante se agrega por medio de algún procedimiento lógico se habla de codificación para protección contra ruido.

La descodificación consiste en eliminar la información redundante agregada por el transmisor, a modo de recuperar la información original con un porcentaje de errores menor al que se hubiera obtenido al transmitir sin símbolos de redundancia. Tradicionalmente se utiliza un descodificador secuencial, cuando se tiene codificación convolucional, el cual reduce la probabilidad de *bit* erróneo P_B, siendo esta última una función de la longitud límite N del código[4].

El descodificador secuencial es subóptimo respecto al descodificador de máxima verosimilitud, sin embargo, del apartado 2.5 se deduce que el algoritmo de Viterbi tiene limitada su capacidad de corrección debido a que los cálculos de descodificación que debe ejecutar se incrementan exponencialmente conforme crece la longitud límite **N** del código. En contraste, la complejidad del descodificador secuencial es independiente de la longitud límite **N** del código por lo que se hace factible, con este método, descodificar códigos con longitudes límite muy grandes. Así, para un código en particular el descodificador secuencial es subóptimo pero puede utilizar valores de **N** muy grandes[2][3].

Como se verá más adelante en detalle, la descodificación secuencial es un método intuitivo de prueba y error para determinar una trayectoria del árbol del código que sea, en algún sentido, similar a la secuencia transmitida. En esta búsqueda el descodificador recorrerá el árbol del código moviéndose sobre una rama que está hacia adelante o hacia atrás de un nodo a la vez. La decisión de moverse en una de las dos direcciones se determina por la variación de la métrica de descodificación a lo largo de la trayectoria seguida por el descodificador. Aquí, la medida de similitud entre secuencias está dada por la métrica de descodificación que, dependiendo de la técnica utilizada, puede ser la distancia Hamming o una probabilidad a posteriori[4].

La técnica alternativa que se propone presume ser más rápida para ejecutar la descodificación que la técnica secuencial, dado que no efectúa una búsqueda avanzando o retrocediendo sobre las ramas del árbol sino que hace una estimación de las ramas supuestas y las compara con las ramas recibidas que están probablemente en error, y en base a un patrón de comparación se contabilizan las diferencias acumuladas entre las trayectorias supuesta y recibida; posteriormente propone cambiar las ramas supuestas que detectó con error por las que, a su criterio, son las correctas, obteniéndose de una vez la trayectoria que se parece más a la trayectoria enviada por el transmisor. El criterio del descodificador se basa en la naturaleza del mismo para propagar errores como consecuencia de tomar una decisión errónea sobre una rama supuesta. Este criterio permite descodificar secuencias erróneas a una velocidad constante por lo que se hace comparable en este sentido al descodificador de Viterbi.

También parece tener ventajas respecto a la memoria requerida dado que no necesita almacenar información de varias trayectorias y sus respectivas métricas de descodificación como el algoritmo de Viterbi ni tampoco información de trayectorias supuestas ya recorridas y desechadas como lo hace el descodificador secuencial; simplemente requiere de un acumulador de diferencias adicional a la trayectoria supuesta y un registro de **N** etapas para almacenar la secuencia probablemente errónea. Los requisitos de velocidad de operación respecto a la velocidad del canal y de cantidad de memoria del descodificador vistos como parámetros de comparación serán analizados en este capítulo limitando el objeto de estudio a el mismo código de tasa **R**=1/4 y longitud límite N=4 que se usó como ejemplo en el capítulo 2 y que resulta satisfactorio en cuanto a

capacidad de corrección requerida, según se verá, para aplicar el descodificador al sistema telefónico 1240 como se propone en el capítulo 4.

El algoritmo propuesto en este capítulo fue concebido en los 70's por el M. en I. Fernando Lepe Casillas, quién en ese entonces lo propuso a una revista técnica publicada en España mediante el artículo "Dos algoritmos de descodificación para codificadores convolucionales" el cual fue rechazado por los editores de la revista; éstas fueron sus razones:

- Las bases matemáticas del algoritmo no se explican con claridad (deducción de la tabla en la fig. 3.4).
- El número de pruebas de simulación no son suficientes para comprobar la efectividad del algoritmo.
- La bibliografía referenciada es insuficiente.
- No es claro como se produce el ahorro de memoria.

Los análisis que se realizan en este capítulo pretenden por un lado justificar analíticamente el algoritmo mediante la deducción de la tabla de la fig. 3.4, y efectuar pruebas del algoritmo con simulaciones por computadora de un canal digital que introduce errores aleatorios al mensaje transmitido. No se pretende que las pruebas sean exhaustivas, más bien que sean realmente una muestra representativa del desempeño del algoritmo para diferentes niveles de ruido en el canal digital. Se incluye además una comparación con el algoritmo de Viterbi, sometido a las mismas pruebas de simulación, para tener un marco de comparación en cuanto a parámetros de descodificación se refiere, tales como memoria requerida.

3.2 Notación y ecuaciones de codificación utilizadas

Haciendo referencia al codificador convolucional, que se describió en el capitulo 2 (fig. 2.5), su entrada está dada por un vector b_i de n dígitos binarios, los cuales se desplazan a través de **N**=4 registros de corrimiento M_N de k_0=1 etapa cada uno, en donde, al conectar estos a **m** sumadores módulo-2 se definen las ecuaciones de codificación v_n. El resultado de las ecuaciones se entrega a un conmutador que entrega en su salida **m**=3 *bits* en serie, esto es triples, por cada *bit* de b_i que entra al codificador; nótese que cada *bit* de b_i influye sobre cuatro triples que genera el codificador. Cuando el último *bit* de b_i entre a la primer etapa M_1 del codificador, se habrá generado a la salida del codificador un vector b_o compuesto de n triples codificados; éste será enviado al receptor a través de un canal digital, el cual agregará ruido en forma aleatoria, entregando al receptor un vector r_n compuesto de n triples, probablemente diferente al vector b_o enviado por el transmisor. Finalmente el descodificador recupera la secuencia de *bits* b_i, que es una estimación de la secuencia b_i a partir del vector recibido r_n. La fig. 3.1 servirá como referencia a los símbolos manejados por el codificador y el descodificador.

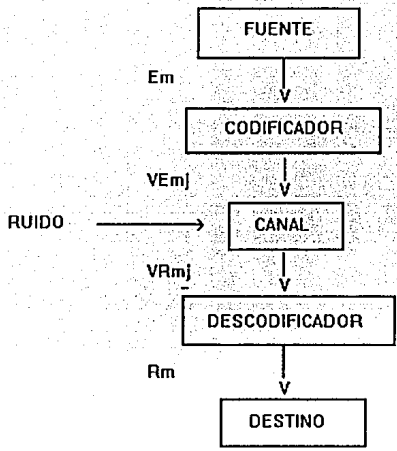


Figura 3.1 Símbolos manejados en el codificador y el descodificador.

Donde:

(3.1)...E_m: Es un elemento del vector b_i, cuyos n elementos binarios representan el mensaje antes de codificarlo.

(3.2)...$VE_{mj(j=1,2,3)}$: Triple que corresponde al *bit* E_m codificado, según las siguientes ecuaciones :

$$VE_{m1} = E_m \qquad \textbf{(3.3)}$$

$$VE_{m2} = E_m \oplus E_{m-1} \oplus E_{m-2} \oplus E_{m-3} \qquad \textbf{(3.4)}$$

$$VE_{m3} = E_m \oplus E_{m-2} \oplus E_{m-3} \qquad \textbf{(3.5)}$$

Nótese que a cada *bit* de información E_m, corresponde un triple de *bits* en el canal, al cual se denotará como:

$$VE_{m:1,2,3} \qquad \textbf{(3.6)}$$

es decir: $VE_{m:1,2,3} = \{VE_{m1}, VE_{m2}, VE_{m3}\}$ **(3.7)**

(3.8)...$VR_{mj(j=1.2.3)}$: Triple del vector r_n a la salida del canal y entregado al receptor que corresponde al triple enviado VE_{mj} pero con ciertas variaciones en su valor provocadas por el ruido del canal.

(3.9)...R_m: Es un elemento de un vector descodificado b_i y corresponde a la decisión final sobre el triple VR_{mj}.

La generación de *bits* de redundancia puede lograrse con el siguiente circuito:

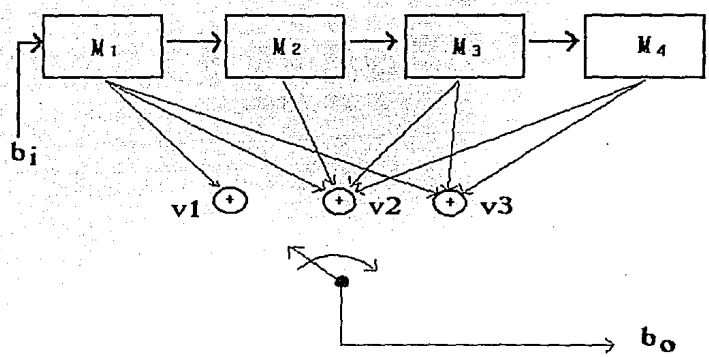


Figura.3.2 Un codificador convolucional[4].

Errores: si bien en un canal físico el ruido introducido se modela con algún proceso estocástico, en este caso de simulación sólo interesa si la detección fue correcta o incorrecta. Por lo cual se hacen las siguientes consideraciones simplificadoras:

$$P\{E_m=1\} = P\{E_m=0\} = 0.5 \qquad \textbf{(3.10)}$$

$$Pe = P\{error/0\} = P\{error/1\} \qquad \textbf{(3.11)}$$

donde Pe es la probabilidad de error en el canal.

La expresión (3.11) equivale a decir que la probabilidad de que $R_m=1$ dado que $E_m=0$ es igual al probabilidad de que $R_m=0$ dado que $E_m=1$.

$$R_m = E_m \text{ si } Pe = 0 \qquad \textbf{(3.12)}$$

$$R_m = E_m \text{ si } Pe = 1 \qquad \textbf{(3.13)}$$

3.3 El descodificador secuencial

Operación[8]

De las ecuaciones (3.3) a (3.5), resulta evidente que el *bit* E_L influye sobre los triples:

$$VE_{j;1,2,3},\ j=L,\ldots,L+3;$$

por lo tanto, en un primer método para identificar a R_L, se usan todas las combinaciones posibles de R_L, ..., R_{L+3} para generar la totalidad de los triples supuestos $VS_{L;1,2,3}$, ..., $VS_{L+3;1,2,3}$; los triples supuestos se construyen usando el mismo codificador de la figura 3.2 como parte del descodificador. El conjunto de datos supuestos R^*_L, ..., R^*_{L+3} que genera los triples supuestos más parecidos a los triples recibidos, se toma como el adecuado, y se admite que $R^*_L=\hat{E}_L$, lo que se define como descodificar el *bit* E_L. Este proceso requiere de memoria suficiente para almacenar todos los posibles triples supuestos, o si se dispone de tiempo, se debe de computar y conservar en memoria el conjunto R^*_L, ..., R^*_{L+3} resultante de las comparaciones entre triples conforme se vayan realizando.

El descodificador secuencial no genera todos los posibles triples supuestos, sino los que necesita para identificar a $\hat{E}_L=R_L$. Tal proceso se lleva a cabo alimentando los *bits* de los triples recibidos dentro del descodificador para generar los triples supuestos con los que se van a comparar los triples recibidos, lo cual evita el tener que guardar en memoria todos los triples del árbol generados por el primer método antes mencionado. Es precisamente en este punto donde se logra la reducción del empleo de espacio en memoria, ya que no se hace necesario almacenar en memoria un árbol cuyas dimensiones requieren de un amplio espacio en élla; además, reduce el tiempo de toma de decisión debido al proceso de búsqueda dentro del mismo. Para éllo, el secuencial hace decisiones consecutivas, comparando los triples supuesto y recibido:

en el nodo L:	$VS_{L;1,2,3}$	vs	$VR_{L;1,2,3}$	**(3.14)**
en el nodo L+1:	$VS_{L+1;1,2,3}$	vs	$VR_{L+1;1,2,3}$	**(3.15)**
en el nodo L+2:	$VS_{L+2;1,2,3}$	vs	$VR_{L+2;1,2,3}$	**(3.16)**
en el nodo L+3:	$VS_{L+3;1,2,3}$	vs	$VR_{L+3;1,2,3}$	**(3.17)**

donde al comparar $VS_{j;1,2,3}$ vs $VR_{j;1,2,3}$ se decide sobre R_j intentando establecer si es 0 o 1 y contabilizar al mismo tiempo las diferencias detectadas (distancia Hamming) entre los triples VS y VR. El parámetro j está en el rango $L<j<L+3$ y representa la toma de decisión j-ésima para los diferentes triples recibidos en el descodificador. En esta tarea se acumulan **diferencias** contables entre los triples supuestos, que se generan con las decisiones ya tomadas, y los triples recibidos.

Si después de la decisión L+3 (donde L representa la decisión L-ésima y L puede tomar cualquier valor en el conjunto de los números naturales) el número de diferencias es reducido, se acepta que $R_L=E_L$. Pero si al realizar la decisión j+1 ($L<j<L+3$) se encuentra que el total de discrepancias es mayor que un umbral **T**, se considera que la decisión anterior fue errónea y que es necesario cambiar R_j por su conjugado. Si al tomar la decisión j+1 se sobrepasa nuevamente el límite de diferencias, es necesario modificar la decisión j-1, para lo cual es indispensable que la decisión j se encuentre marcada como modificada.

Para aclarar como se lleva a cabo el marcaje en el caso general en que E_L influye sobre **N** triples, se da el siguiente ejemplo general tabulado fig. 3.3, en donde un cero indica una decisión ya tomada pero no marcada y un asterisco indica una decisión modificada:

Decisión			(j-1)	(j)	(j+1)		
tiempo	L	L+1	L+2	L+3	L+4 ...	L+N	**COMENTARIOS**
t1	0	0	0	0	0		Se sobrepasa el límite de diferencias permisible **T** en la decisión j+1
t2	0	0	0	*			Se modifica la decisión j y se marca L+3
t3	0	0	0	0	0		Igual que t1.
t4	0	0	*				Se modifica la decisión j-1 y se se marca L+2
t5	0	0	0	0	0		Igual que t1.
t6	0	0	0	*			Igual que t2.
t7	0	0	0	0	0		Igual que t1.
t8	0	*					Se modifica la decisión j-2 y se marca L+1
.							
.							
.							
tm	*						Se modifica la 1er decisión y se toma $R_L=\overline{VR}_{L,1}$

Fig.3.3 Tabla de marcaje de error detectado.

Así pues, el descodificador secuencial requiere de memoria para marcar los cambios de decisiones y para almacenar 4 triples recibidos sucesivamente. El tiempo que toma identificar R_L no es siempre el mismo, siendo está característica de variabilidad de hecho una limitante para el descodificador secuencial. Es difícil realizar una evaluación detallada del descodificador secuencial, pués a éste hay que asignarle ciertos umbrales (**T**), los cuales fijan el número de diferencias que se permiten al seguir por una rama del árbol de descodificación, y los umbrales que dan la eficiencia óptima varían con la probabilidad de error en el canal Pe. Sin embargo se puede utilizar un conjunto de umbrales seleccionados para cada Pe y observar el desempeño del algoritmo en un rango establecido de Pe.

Limitación básica del Descodificador Secuencial[1][4]

La operación del descodificador secuencial tal como se describió arriba no es sino un esquema general de descodificación secuencial de esquemas que ya existen y cuyas diferencias estriban en la medida de similitud entre la secuencia recibida y la secuencia supuesta, esto es, la métrica de descodificación que utilizan.

En el esquema de descodificación descrito anteriormente se tomó la distancia Hamming como métrica de descodificación; sin embargo, existen otros esquemas de descodificación secuencial que toman como métrica la verosimilitud de una secuencia recibida r_n dado que se envió el vector b_o, que para un canal discreto sin memoria corresponde al producto de verosimilitudes de todas y cada una de las ramas o triples que componen r_n.

Dependiendo de la métrica utilizada, el método de búsqueda del descodificador tendrá otra variante ya que la métrica se compara con un cierto umbral **T**, que se calcula en función de la longitud de la trayectoria supuesta y de las veces que el descodificador se haya regresado a un mismo nodo y por tanto el valor de **T** varía de una rama a otra del árbol para el caso del descodificador secuencial de Wozencraft-Reiffen[4]; o bien **T** puede tener un valor fijo y su variación, en función del ruido del canal, será también un incremento fijo de **T** tal y como lo hace el descodificador secuencial de Fano[4].

Para cada esquema de descodificación secuencial se debe tener presente una limitación importante que no depende del método de búsqueda del descodificador sino más bien de factores que están fuera del objetivo de éste trabajo como son las propiedades del canal, que se hacen presentes a través de la naturaleza del ruido N_0 que agrega el canal a la información transmitida, y las características físicas del medio de transmisión que afectan directamente la energía por cada dígito del canal recibido E_r. Se trata de la velocidad mínima de operación del descodificador respecto a la velocidad del canal llamada velocidad de corte de operación R_{comp}; debido a que los cálculos de descodificación varían y se incrementan al aumentar el nivel de ruido de la señal, la carga de cálculos se ejecuta a una cierta velocidad que de ser menor o igual a la velocidad del canal, la carga de

cálculos que efectúa el descodificador se incrementará hasta infinito según simulaciones efectuadas por computadora[4].

Parámetros del Descodificador Secuencial[3][4]

En el descodificador secuencial normalmente habrá una variación, función del tiempo, en el número de cálculos de descodificación necesarios para extender la secuencia supuesta una rama. La variación en el número de cálculos requeridos para descodificar un dígito, tiene un efecto en cuanto a costo y circuitería que es un factor considerable dado que afecta bastante la implantación del descodificador tanto en la simulación como en la práctica.

Las búsquedas prolongadas de la trayectoria óptima, en particular para secuencias recibidas con alto nivel de ruido, provocan la acumulación de *bits* a la entrada del descodificador, lo que obliga a incluir una memoria intermedia a la entrada del mismo para almacenarlos y evitar la pérdida de los *bits*, mientras se realiza la búsqueda.

Para minimizar el número de dígitos acumulados en la memoria intermedia durante una búsqueda, el descodificador necesita tener un **factor de velocidad** o una ventaja en velocidad del orden de diez veces la tasa de información, esto como regla general, ya que este factor no es una constante como en el esquema de descodificación de máxima verosimilitud o de Viterbi, visto en el capítulo 2.

Comercialmente están disponibles descodificadores que manejan altas tasas de información ($\geq 1/2$ *bit*/símbolo) y tasas de canal muy altas (10 M*bits*/s o mayores).

En síntesis, se tienen las siguientes ventajas y desventajas de la descodificación secuencial:

Las tasas de error disminuyen exponencialmente con la longitud límite **N** operando a tasas menores a la capacidad del canal.

La carga de cálculos está acotada por una constante que no depende de la longitud límite **N** sino de las propiedades del canal, ésta es, la velocidad de corte de cálculos $\mathbf{R}_{comp}$. Lo anterior implica el uso en la práctica de *buffers* de 200 *bits* como mínimo en sistemas de comunicaciones vía satelite[7].

Vista como una ventaja sobre el descodificador de Viterbi, la capacidad de memoria requerida crece linealmente con la longitud límite **N**. Y como una desventaja está la variación en el tiempo de cálculo para descodificar una rama.

3.4 Desarrollo del algoritmo de descodificación propuesto

Al igual que en el descodificador secuencial visto anteriormente, el primer paso es suponer un valor de R_m, por ejemplo $R_m = VR_{m,1}$, con el cual se genera $VS_{m;1,2,3}$. Si se encuentran dos diferencias entre los triples supuesto y recibido, se decide que el valor de R_m es el conjugado del supuesto inicialmente. A continuación se analizan las consecuencias de decidir erróneamente sobre R_L, esto es, tomar $R_L = E_L$, debido a que en $VR_{L;1,2,3}$ existe un error doble o triple. Se supone que las decisiones anteriores han sido correctas y los triples $VR_{L+1;1,2,3,...}$, $VR_{L+3;1,2,3}$, coinciden con los triples $VE_{L+1;1,2,3,...}$, $VE_{L+3;1,2,3}$, y que:

(3.18) $$VS_{L+1,1} = VR_{L+1,1} = R_{L+1} = E_{L+1}$$

(3.19) $$VS_{L+1,2} = R_{L+1} \oplus R_L \oplus R_{L-1} \oplus R_{L-2}$$

$$= E_{L+1} \oplus \overline{E}_L \oplus E_{L-1} \oplus E_{L-2} = \overline{VR}_{L+1,2}$$

(3.20) $$VS_{L+1,3} = R_{L+1} \oplus R_{L-1} \oplus R_{L-2}$$

$$= E_{L+1} \oplus E_{L-1} \oplus E_{L-2} = VR_{L+1,3}$$

entonces, se decide correctamente sobre R_{L+1} y se detecta una diferencia entre los triples; y para la posición L+2:

(3.21) $$VS_{L+2,1} = VR_{L+2,1} = R_{L+2} = E_{L+2}$$

(3.22) $$VS_{L+2,2} = R_{L+2} \oplus R_{L+1} \oplus R_L \oplus R_{L-1}$$

$$= E_{L+2} \oplus E_{L+1} \oplus \overline{E}_L \oplus E_{L-1} = \overline{VR}_{L+2,2}$$

(3.23) $$VS_{L+2,3} = R_{L+2} \oplus R_L \oplus R_{L-1}$$

$$= E_{L+2} \oplus \overline{E}_L \oplus E_{L-1} = \overline{VR}_{L+2,3};$$

se decide incorrectamente sobre R_{L+2} y se detectan dos diferencias entre los triples; y para la posición L+3:

(3.24) $$VS_{L+3,1} = VR_{L+3,1} = R_{L+3} = E_{L+3}$$

(3.25) $$VS_{L+3,2} = R_{L+3} \oplus R_{L+2} \oplus R_{L+1} \oplus R_L$$

$$= E_{L+3} \oplus \overline{E}_{L+2} \oplus E_{L+1} \oplus \overline{E}_L = VR_{L+3,2}$$

(3.26) $$VS_{L+3,3} = R_{L+3} \oplus R_{L+1} \oplus R_L$$

$$= E_{L+3} \oplus E_{L+1} \oplus \overline{E}_L = \overline{VR}_{L+3,3};$$

se decide correctamente sobre R_{L+3} y se detecta una diferencia; las expresiones (3.18) a (3.26) se pueden visualizar como sigue:

comparación triple supuesto vs triple recibido	decisión sobre R_{L+i} (i=1,2,3) y consecuencia	
$VS_{L+1:1,2,3} = VR_{L+1:1,\bar{2},3}$	$R_{L+1} = E_{L+1}$; DIF = 1	**(3.27)**
$VS_{L+2:1,2,3} = VR_{L+2:1,\bar{2},\bar{3}}$	$R_{L+2} = \bar{E}_{L+2}$; DIF = 2	**(3.28)**
$VS_{L+3:1,2,3} = VR_{L+3:1,2,\bar{3}}$	$R_{L+3} = E_{L+3}$; DIF = 1	**(3.29)**

Se observa que hasta este punto se han acumulado cuatro diferencias consecutivas por lo que se vislumbra ya un primer algoritmo al establecer la siguiente:

REGLA 1: *"Cuando en la decisión J se hayan acumulado 4 diferencias consecutivas, cámbiese R_{J-1} y R_{J-3} por sus respectivos conjugados; continúese con la decisión J+1".*

Obsérvese que en este esquema no se repite ninguna decisión. Cabe aclarar que como memoria adicional, sólo se requiere el contador de diferencias consecutivas, el cual debe renovarse después de cada corrección.

3.5 Definición del Algoritmo de Descodificación Propuesto

El primer paso es suponer que $R_L = VR_{L,1}$. Esto es debido a que las ecuaciones de codificación especifican que el primer *bit* de cada triple es en principio (si no se toma en cuenta el efecto del ruido, el cual puede cambiar ese *bit*) el *bit* codificado. Si se encuentran dos o tres diferencias entre los triples supuesto y recibido, se cambia a: $R_L = \overline{VR}_{L,1}$. Para este algoritmo, la posición relativa de estos cambios es la base para correcciones posteriores.

La consecuencia de decidir erróneamente sobre R_L, tomando $R_L=\bar{E}_L$, cuando las decisiones anteriores han sido correctas y los triples recibidos posteriores no contienen errores, es la generación de errores alternados en las decisiones subsecuentes, causados por los errores en los triples supuestos VS :

comparación triple supuesto vs triple recibido	decisión sobre R_{L+i}, (i=1,2,3) y consecuencia	
$VS_{L+1:1,2,3} = VR_{L+1:1,\bar{2},3}$	$R_{L+1} = E_{L+1}$; DIF = 1	**(3.30)**
$VS_{L+2:1,2,3} = VR_{L+2:1,\bar{2},\bar{3}}$	$R_{L+2} = \bar{E}_{L+2}$; DIF = 2	**(3.31)**
$VS_{L+3:1,2,3} = VR_{L+3:1,2,\bar{3}}$	$R_{L+3} = E_{L+3}$; DIF = 1	**(3.32)**
$VS_{L+4:1,2,3} = VR_{L+4:1,\bar{2},\bar{3}}$	$R_{L+4} = \bar{E}_{L+4}$; DIF = 2	**(3.33)**
$VS_{L+5:1,2,3} = VR_{L+5:1,2,\bar{3}}$	$R_{L+5} = E_{L+5}$; DIF = 1	**(3.34)**

El resultado de las observaciones anteriores abre las puertas a un conjunto numeroso de posibles estrategias de corrección, representando todas ellas variaciones sobre la misma idea:

REGLA 2: *"Cuando se detecten diferentes los bits 2 y 3 en los triples supuestos a los bits 2 y 3 en los triples recibidos, y esto ocurra en las posiciones L, L+2, ..., L+2N, resulta conveniente tomar el conjugado de los bits* R_{L-2}, R_L, ..., R_{L+2N}".

Antes de cualquier proposición definitiva se debe obtener:

1)Cuáles de los *bits* en las posiciones R_{L-2}, R_L,...,R_{L+2N} deben cambiarse.

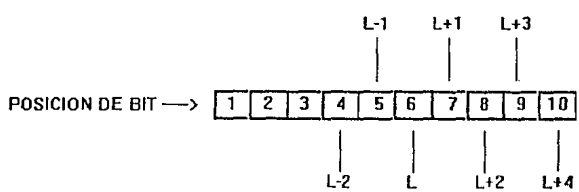
2)Cuál es el mejor valor de N (no confundir con la longitud límite N).

3)Qué condiciones, presentes en las posiciones impares, o sea L-1, L+1, L+3,...,L+2N-1 favorecen o no dicha operación.

Para ver cuales de los *bits* R_m deben cambiarse, se necesitan fijar dos condiciones, a saber:

- El mejor valor de N (inciso 2) para descodificación óptima y,
- Las posiciones L-2, L, L+2, ..., L+2N... más convenientes

La figura 3.4 (siguiente página) indica una tabla de resultados acerca del desempeño del algoritmo de descodificación obtenida a partir de un programa cuyo desarrollo se explica en los apartados 3.7 y 3.8. El programa para obtener la tabla se basa en la aplicación del teorema de Bayes y viene a ser el resultado del análisis que se hace de este algoritmo en el apartado 3.6. La evaluación de la tabla se realizó tomando como referencia un mensaje con longitud de 10 *bits* considerando que el canal puede variar su nivel de ruido con probabilidad de transición variando de 0.01 a 0.99. Cada uno de los 10 *bits* tienen una cierta posición que se asocia a las posiciones mencionadas en el algoritmo como sigue:



PMC	L-2 PM4DD	L PM6DD	L+2 PM8DD	L+4 PM10DD	L-2,L P4X6	L,L+2 P6X8	L-2,L,L+4 P4X6X8	L,L+2,L+4 P6X8X10
0.01	1.000000	1.000000	1.000000	0.000001	1.000000	1.000000	1.000000	0.000001
0.03	0.999998	0.999998	0.999999	0.000031	0.999997	0.999997	0.999996	0.000031
0.05	0.999983	0.999984	0.999991	0.000162	0.999975	0.999976	0.999967	0.000162
0.07	0.999926	0.999931	0.999961	0.000494	0.999894	0.999894	0.999859	0.000493
0.09	0.999766	0.999789	0.999873	0.001171	0.999671	0.999671	0.999564	0.001166
0.11	0.999397	0.999469	0.999666	0.002392	0.999163	0.999163	0.998894	0.002375
0.13	0.998631	0.998826	0.999225	0.004433	0.998134	0.998133	0.997539	0.004376
0.15	0.997168	0.997633	0.998364	0.007667	0.996202	0.996202	0.995005	0.007505
0.17	0.994529	0.995542	0.996775	0.012594	0.992782	0.992782	0.990532	0.012185
0.19	0.989996	0.992049	0.993987	0.019873	0.987008	0.987008	0.983006	0.018921
0.21	0.982529	0.986441	0.989299	0.030331	0.977648	0.977648	0.970845	0.028265
0.23	0.970723	0.977773	0.981738	0.044951	0.963057	0.963057	0.951954	0.040735
0.25	0.952856	0.964906	0.970071	0.064772	0.941236	0.941236	0.923805	0.056648
0.27	0.927118	0.946640	0.952926	0.090684	0.910087	0.910087	0.883786	0.075881
0.29	0.892120	0.922016	0.929108	0.123092	0.867962	0.867962	0.829916	0.097603
0.31	0.847618	0.890719	0.898065	0.161542	0.814435	0.814435	0.761845	0.120135
0.33	0.795152	0.853427	0.860333	0.204520	0.750957	0.750957	0.681701	0.141140
0.35	0.738136	0.811802	0.817619	0.249664	0.680896	0.680896	0.594118	0.158201
0.37	0.681120	0.768028	0.772353	0.294379	0.608702	0.608702	0.505149	0.169551
0.39	0.628556	0.724087	0.726866	0.336550	0.538569	0.538569	0.420537	0.174544
0.41	0.583766	0.681212	0.682702	0.374949	0.473375	0.473375	0.344350	0.173596
0.43	0.548544	0.639742	0.640368	0.409195	0.414348	0.414348	0.278538	0.167763
0.45	0.523339	0.599362	0.599543	0.439444	0.361391	0.361391	0.223309	0.158261
0.47	0.507672	0.559492	0.559517	0.466040	0.313682	0.313682	0.177810	0.146160
0.49	0.500505	0.519654	0.519654	0.489280	0.270232	0.270232	0.140734	0.132301
0.51	0.500486	0.479739	0.479739	0.509313	0.230263	0.230263	0.110712	0.117346
0.53	0.506086	0.440113	0.440138	0.526140	0.193383	0.193383	0.086500	0.101875
0.55	0.515705	0.401559	0.401740	0.539681	0.159586	0.159586	0.067046	0.086456
0.57	0.527797	0.365096	0.365732	0.549870	0.129120	0.129120	0.051490	0.071651
0.59	0.540981	0.331723	0.333269	0.556725	0.102300	0.102300	0.039131	0.057965
0.61	0.554155	0.302201	0.305197	0.560388	0.079336	0.079336	0.029394	0.045781
0.63	0.566532	0.276934	0.281905	0.561123	0.060235	0.060235	0.021800	0.035323
0.65	0.577628	0.255958	0.263323	0.559276	0.044793	0.044793	0.015946	0.026643
0.67	0.587213	0.239019	0.249029	0.555242	0.032638	0.032638	0.011490	0.019660
0.69	0.595234	0.225679	0.238392	0.549417	0.023302	0.023302	0.008146	0.014195
0.71	0.601749	0.215417	0.230707	0.542167	0.016295	0.016295	0.005672	0.010027
0.73	0.606878	0.207708	0.225288	0.533821	0.011148	0.011148	0.003871	0.006922
0.75	0.610763	0.202066	0.221528	0.524657	0.007448	0.007448	0.002583	0.004663
0.77	0.613548	0.198071	0.218918	0.514908	0.004846	0.004846	0.001679	0.003056
0.79	0.615371	0.195371	0.217055	0.504767	0.003057	0.003057	0.001059	0.001941
0.81	0.616355	0.193682	0.215630	0.494387	0.001860	0.001860	0.000643	0.001189
0.83	0.616609	0.192779	0.214417	0.483895	0.001083	0.001083	0.000374	0.000696
0.85	0.616228	0.192483	0.213252	0.473389	0.000597	0.000597	0.000206	0.000386
0.87	0.615294	0.192660	0.212026	0.462949	0.000307	0.000307	0.000105	0.000199
0.89	0.613880	0.193203	0.210670	0.452639	0.000143	0.000143	0.000049	0.000093
0.91	0.612050	0.194034	0.209144	0.442507	0.000059	0.000059	0.000020	0.000038
0.93	0.609858	0.195091	0.207429	0.432595	0.000020	0.000020	0.000007	0.000013
0.95	0.607353	0.196328	0.205524	0.422932	0.000005	0.000005	0.000002	0.000003
0.97	0.604580	0.197710	0.203436	0.413540	0.000001	0.000001	0.000000	0.000000
0.99	0.601577	0.199212	0.201183	0.404438	0.000000	0.000000	0.000000	0.000000

Fig.3.4 Tabla de probabilidades de error para el algoritmo propuesto (longitud de mensaje LTEDO=10).

En cada una de las posiciones del mensaje el algoritmo ha de tomar una decisón sobre el valor del *bit* a descodificar, y como ya se mencionó, es de interés saber cuales de las posiciones L-2, L, L+2, ..., L+2N deben cambiarse. Así las columnas de la tabla (fig. 3.4) indican valores de probabilidad definidos como sigue:

PM4DD: probabilidad de que esté mal el *bit* 4, dadas dos diferencias alternadas detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

PM6DD: probabilidad de que esté mal el *bit* 6, dadas dos diferencias alternadas, detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

PM8DD: probabilidad de que esté mal el *bit* 8, dadas dos diferencias alternadas, detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

P4x6: probabilidad de que estén mal el *bit* 4 y el *bit* 6, dadas dos diferencias alternadas, detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

P6x8: probabilidad de que estén mal el *bit* 6 y el *bit* 8, dadas dos diferencias alternadas, detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

P4x6x8: probabilidad de que estén mal los *bits* 4, 6, y 8, dadas dos diferencias alternadas, detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

P6x8x10: probabilidad de que estén mal los *bits* 6, 8, y 10, dadas dos diferencias alternadas, detectadas en las posiciones 6, 8, y 10 (LTEDO=10).

Para cada definición anterior se supone que no hay errores en los triples anteriores 1, 2, y 3.

Se observa en la tabla de la fig 3.4 que para valores de probabilidad de error en el canal **PMC**<0.18, los mejores valores de probabilidad se dan para las columnas de los eventos (L-2, L); (L, L+2); y (L-2, L, L+2) cuyas probabilidades son P4X6, P4X8, y P4X6X8 respectivamente, de los cuales se observa que los valores P4X6X8 decrecen más rápidamente que P4X6 y P6X8, pero al mismo tiempo es tan efectivo como P4X6 y P4X8 dentro del rango de **PMC** indicado y además incluye a ambos eventos. De lo anterior

se concluye que las posiciones más convenientes para invertir sus valores son L-2, L, L+2; ésto lleva implícito el hecho de que el mejor valor de N es igual a 1.

Por otro lado, las condiciones favorables de las posiciones intermedias L+1, L+3, ..., L+2N-1 son: recibir triples cuyas distancias a los triples supuestos sean de 0 ó 1 lo cual implica una decisión correcta según lo indica la definición del algoritmo y se pone de manifiesto en las ecuaciones 3.30, 3.32, y 3.34

Con las ecuaciones de codificación empleadas, el algoritmo propuesto es útil para probabilidades de error en el canal (Pe ó PMC) menores que 0.18, con N=1.

De la tabla también se observa que respecto al valor de N para el algoritmo propuesto, fueron mejores los resultados con N=1 que con N=2; una vez que se ha cometido una decisión errónea en la posición L, la probabilidad de detectar 2 diferencias en las posiciones L+2 y L+4 y ... y L+2N, disminuye con N, si Pe>0.

3.6 Justificación del Algoritmo

El árbol del código es útil para realizar un análisis que justifique la efectividad del algoritmo en cuestión dado que es posible visualizar los siguientes aspectos:

- Los errores introducidos en los triples recibidos.
- La propagación del error inherente a la característica propia del codificador.

Para lo anterior conviene suponer lo siguiente:

- El transmisor envía una secuencia codificada de ceros.
- Decidir por un 1 en un triple recibido implica una decisión errónea.
- Decidir por un 0 en un triple recibido implica una decisión correcta.

El árbol se ilustra en la figura 3.5 donde se indica la asignación de etapas pares **A** y etapas impares **B**. Las etapas **A** involucran las posiciones L, L+2, ..., L+2N mencionadas en la definición del algoritmo, mientras que las etapas **B** involucran las posiciones L+1, L+3, ..., L+2N-1.

Se considera además que las etapas **B** involucran todos los triples recibidos que tengan una o ninguna diferencia con el triple supuesto correpondiente, y que las etapas **A** involucran todos los triples recibidos que tengan dos con el triple supuesto correspondiente. Dado que sobre el mismo árbol se hacen consideraciones para aplicar el teorema de Bayes, se le llamará árbol bayesiano.

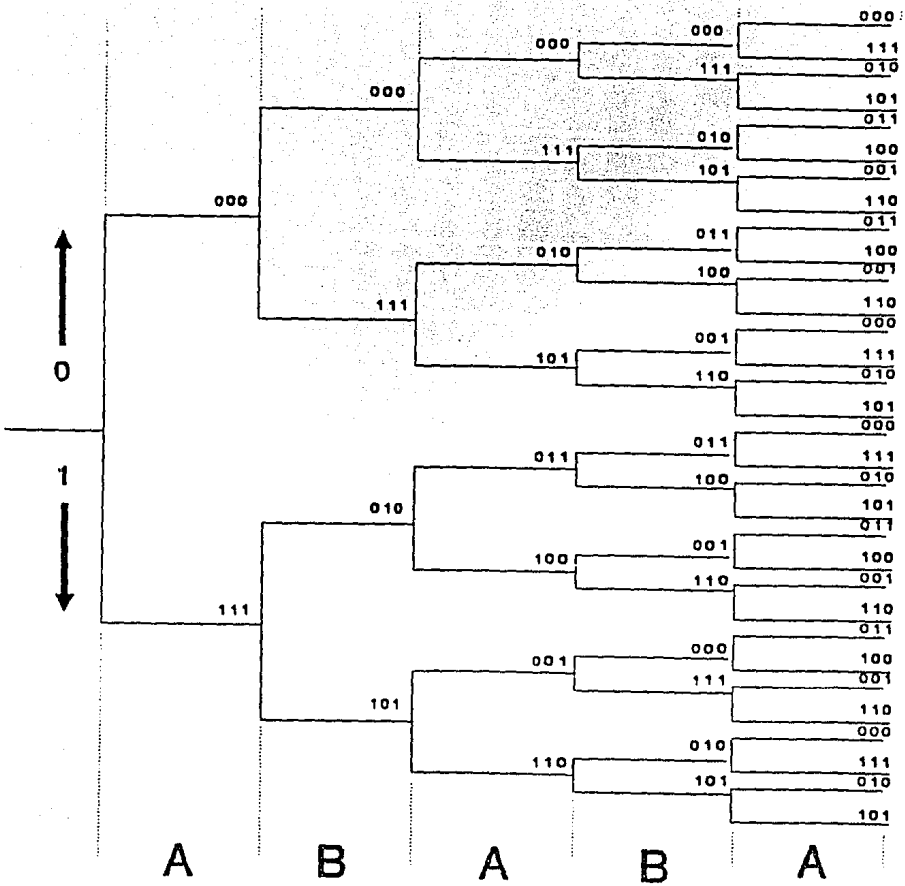


Figura 3.5 Arbol Bayesiano

Para cada decisión tomada dentro del árbol sobre una rama de las etapas **B**, se consideran todos los triples posiblemente recibidos dado que en cada nodo de cada una de estas etapas del árbol se tienen dos alternativas por decidir y cada alternativa asocia la mitad de los $2^3=8$ posibles triples recibidos menos el triple cuyo bit generador o primer bit de dicho triple implica generar un triple supuesto con dos diferencias (ver ejemplo más abajo), esto es, los triples que difieren en una o ninguna posición del triple recibido. Mientras que para una rama de las etapas **A** se preve la condición de recibir aquel triple que presente dos diferencias en los *bits* 2 y 3 con respecto al triple supuesto.

Nótese que el ruido introducido por el canal de transmisión en los triples enviados influirá en la toma de decisiones, lo que a su vez generará una toma de decisión errónea que será propagada en forma periódica por el descodificador debido a su característica propia, es decir, la indicada por las ecuaciones (3.30) a (3.34) (pag. 111). Al observar las ecuaciones (3.30) a (3.34) se puede decir que la lógica del descodificador parece simple, sin embargo, ésta lógica no es determinante para hacer efectiva la operación de descodificación para diferentes valores de probabilidad de error en el canal.

El objetivo del análisis es determinar los rangos de la probabilidad de error en el canal en los que el descodificador toma decisiones confiables dado que se puede pensar que para bajas probabilidades de error en el canal el descodificador es confiable y para altas probabilidades de error en el canal el descodificador es ineficiente.

Para evaluar los rangos de probabilidad de error en el canal se observa que las ecs. (3.30) a (3.34) ponen de manifiesto la propagación de una decisión errónea dentro de los triples supuestos generados por el descodificador.

Por ejemplo: sea el triple supuesto VS=**000** en una rama de cualquier etapa **B**, la probabilidad de decidir que éste fué el triple $\widehat{VE}$, esto es, el valor estimado de VE, equivale a decir que se recibió cualquiera de los siguientes triples:

000
001
010,

Nótese que el triple 100 no se incluye ya que no es posible generarlo a partir del primer bit (0) de VS y suponer que se puede tomar el primer bir de VS igual a 1 implica que VS=111. Por otro lado los tres triples recibidos se pueden ver como una ocurrencia de eventos y por tanto es posible calcular la probabilidad de la unión de tales eventos como sigue:

$$P(\text{decidir que } VE=VS) = P(000)+P(001)+P(010),$$

donde los 1's tienen una probabilidad p (probabilidad de error en el canal) y los 0's tienen una probabilidad q=1-p. Luego como los 0's y los 1's son independientes, cada una de las probabilidades anteriores se calcula como sigue:

$$P(000) = q^3$$

$$P(001) = P(010) = q^2p$$

así: $P(\text{decidir que } \widehat{VE}=\mathbf{000}) = q^3+2q^2p,$

observese que se tomó como base para éste cálculo los triples que difieren en una posición del triple supuesto ya sea en el bit 2 ó 3, o no difieren en ninguna.

Análogamente, para el resto de los triples supuestos 001, 010, 011, 100, 101, 110, 111 que ocupen una rama de cualquier etapa **B** se calcula la asignación de probabilidad de error de cada uno de ellos en una rama en cualquiera de esas posiciones:

para VS= **001**:

$$P(\text{decidir que } \widehat{VE}=VS) = P(001)+P(011)+P(000),$$

$$= pq^2 + p^2q + q^3$$

o bien: $P(\text{decidir que } \widehat{VE}=\mathbf{001}) = q^3 + p^2q + pq^2,$

para VS= **010**:

$$P(\text{decidir que } \widehat{VE}=VS) = P(010)+P(011)+P(000),$$

$$= q^2p + qp^2 + q^3$$

o bien: $P(\text{decidir que } \widehat{VE}=\mathbf{010}) = q^3 + p^2q + pq^2,$

para VS= **011**

$$P(\text{decidir que } \widehat{VE}=VS) = P(011)+P(010)+P(001),$$

$$= qp^2 + q^2p + q^2p$$

o bien: $P(\text{decidir que } \widehat{VE}=\mathbf{011}) = 2pq^2 + p^2q,$

Nótese que los triples supuestos 111, 110, 101, y 100 son el conjugado de los triples supuestos anteriores, y por tanto la probabilidad de cada uno de éstos se calcula fácilmente cambiando en cada una de las expresiones obtenidas anteriormente p por q y viceversa, así:

$P(\text{decidir que } \widehat{VE}=\mathbf{111}) = p^3 + 2p^2q,$

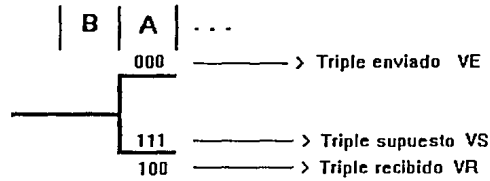
$P(\text{decidir que } \widehat{VE}=\mathbf{110}) = p^3 + p^2q + pq^2,$

$P(\text{decidir que } \widehat{VE}=\mathbf{101}) = p^3 + p^2q + pq^2$

$P(\text{decidir que } \widehat{VE}=\mathbf{100}) = 2p^2q + pq^2,$

Obsérvese que entre los triples supuestos y los recibidos en estas posiciones, existirá una o ninguna diferencia. Mientras que para los triples que ocupan las posiciones pares (etapas A del árbol bayesiano) se les asignará una expresión de probabilidad que ponga de manifiesto lo que representan las ecuaciones (3.31) y (3.33) (pag. 111).

Por ejemplo: sea el triple recibido VR=**100**,
el triple supuesto es VS=**111**



Al observar que entre el triple supuesto y el triple recibido hay dos diferencias, ésto es, $VS_{L;1,2,3}$ vs $VR_{L;1,2,3}$ implican dos diferencias; se establece que la lógica del descodificador decidirá por el triple **000** como el triple enviado VE que es el conjugado del triple supuesto lo que hace que se detecte una sola diferencia entre el conjugado del triple enviado y el triple recibido o sea:

$$VE = 000$$
$$VR = 100$$

lo anterior conduce a:

P(decidir que $\widehat{VE}=VS_{L;\bar{1},\bar{2},\bar{3}}$) =
P(tomar el conjugado del supuesto, $\widehat{VE}=000$) = q^2p.

Se sigue el mismo análisis para asignar una expresión de probabilidad al resto de los triples que ocupen las etapas A en el árbol bayesiano:

para VS = 100, $\widehat{VR}$=111 y:

P(decidir que $\widehat{VE}=VS_{L;\bar{1},\bar{2},\bar{3}}$) =
P(tomar el conjugado del supuesto, $\widehat{VE}=011$) = p^3,

para VS = 101, VR=110 y:

P(decidir que $\widehat{VE}=VS_{L;\bar{1},\bar{2},\bar{3}}$) =
P(tomar el conjugado del supuesto, $\widehat{VE}=010$) = p^2q,

para VS = 110, VR=101 y:

P(decidir que $\widehat{VE}=VS_{L;\bar{1},\bar{2},\bar{3}}$) =
P(tomar el conjugado del supuesto, $\widehat{VE}=001$) = p^2q,

Para los triples restantes se calcula su probabilidad, al igual que en los triples que ocupan etapas **B**, tomando en cuenta que son los conjugados de los primeros, se intercambian la p's por las q's y viceversa, obteniendo:

P(tomar el conjugado del supuesto, $\widehat{VE}$=111) = p^2q

P(tomar el conjugado del supuesto, $\widehat{VE}$=100) = q^3

P(tomar el conjugado del supuesto, $\widehat{VE}$=101) = q^2p

P(tomar el conjugado del supuesto, $\widehat{VE}$=110) = q^2p

Continuando con el análisis, cada una de las ramas del árbol bayesiano representan eventos independientes entre sí y cada una de las posibles trayectorias formadas por estos eventos representan la ocurrencia simultánea de ellos; así se puede decir que para evaluar la probabilidad de que ocurra una de las trayectorias del árbol se utiliza la regla de la multiplicación[5][6] la cual expresa que:

Sean los eventos independientes A_1, A_2, ..., A_n, entonces:

$$P(A_1 \cap A_2 \cap ... \cap A_n) = P(A_1)XP(A_2)X..XP(A_n) = \prod_{i=1}^{n} (A_i), \qquad \textbf{(3.35)}$$

ésto es, si en la expresión anterior se toma cada A_i como una rama que conforma una trayectoria del árbol, entonces se puede obtener la probabilidad de que ocurra dicha trayectoria.

Posteriormente se evalúa la probabilidad de detectar dos diferencias en cada rama de las etapas **A** y cero o una diferencia de las etapas **B** (evento DD) como la sumatoria:

$$P(DD) = \sum_{i=1}^{r} P(T_i), \qquad \textbf{(3.36)}$$

donde:

$r=2^L$ es el número total de trayectorias posibles que tiene el árbol y que depende de L (número de etapas del árbol).

$P(T_i)$ es la probabilidad de la i-ésima trayectoria del árbol calculada a partir de la ec. 3.35.

La probabilidad condicional de decidir erroneamente, ésto es, tomar R(I)=1 en una rama de cualquier etapa **A** = 4, 6, 8, 10 o cualquiera de las intersecciones (4∩6) o (6∩8) o (4∩6∩8), o (6∩8∩10) dado que se detectaron dos diferencias en las posiciones 6, 8, y 10 se evalúa en forma similar al ejemplo siguiente:

sea PM4 la probabilidad de la intersección de eventos

PM4= P{DD∩M4},

y sea PM4DD la probabilidad condicional:

$$PM4DD = P\{M4/DD\},$$

de donde $P(M4/DD)P(DD) = P(DD \cap M4)$

o bien: $$P(M4/DD) = \frac{P(DD \cap M4)}{P(DD)} = PM4DD$$

como $P(DD \cap M4) = P(M4) = PM4$:

$$PM4DD = \frac{PM4}{P(DD)} \qquad (3.37)$$

donde el cálculo de PM4 implica efectuar la suma de la probabilidad de todas las trayectorias (T_i) del árbol bayesiano que contienen un error en L=4, esto es, R(4) = 1 y además dos diferencias detectadas en las etapas pares; el valor de P(DD) se calcula con la expresión 3.36.

La expresión 3.37 es el valor de la probabilidad de error en L=4 dadas dos diferencias detectadas en las posiciones L=6, 8, 10; y se puede aplicar la misma expresión para evaluar la probabilidad de los demás eventos M6DD, M8DD, M10DD, 4x6DD, etc., tomando en consideración que las trayectorias por calcular son diferentes para cada evento.

Finalmente, habiendo obtenido las probabilidades para cada evento, es posible determinar la probabilidad de que las decisiones en las posiciones pares hallan sido o no erróneas tomando en cuenta el efecto del ruido en un canal binario simétrico más la propagación del error inherente a la característica del descodificador propuesto.

3.7 Diagrama de flujo del programa utilizado para calcular las probabilidades de detectar dos diferencias en una etapa A

En este inciso será posible indicar el cálculo paso a paso de las diferentes expresiones que determinan las probabilidad de error en cada uno de los triples y posteriormente el cálculo de la probabilidad de detectar dos diferencias en las posiciones pares (etapas 6, 8, y 10), haciendo referencia al diagrama de flujo, figura 3.6, a partir del cual se generó el programa que evalúa el desempeño del algoritmo. El diagrama de flujo se construyó en base al análisis realizado en el apartado 3.6. Para su construcción se hicieron las siguientes consideraciones:

- Se analiza un mensaje o secuencia binaria transmitida de 10 *bits* de longitud.
- La asignación de los *bits* del mensaje con sus posiciones es la misma que la descrita en el apartado 3.5 al describir la tabla de la figura 3.4

- La asignación de probabilidad asignada a un triple o rama de una trayectoria del árbol corresponde al símbolo:

FD si la rama está en una etapa **A** (posición par);

F si la rama está en una etapa **B** (posición impar)

- La probabilidad de una trayectoria del árbol es el producto de las probabilidades de sus correspondientes ramas según se vió en el apartado anterior, esto es, el producto de los factores (F o FD) asignado a cada una de las ramas que la componen y que en el diagrama de flujo se representa con la variable PROD.

Al inicio (1) del diagrama se indica la longitud de la secuencia de ceros para generar el árbol Bayesiano y evaluar el desempeño del código, ya sea 8 o 10.

A continuación (2) se asignan puros ceros a la secuencia R(I) de longitud indicada en el inicio; en los siguientes dos recuadros, (3) y (4), se evaluarán los factores de probabilidad que serán asignados a cada uno de los triples que forman el árbol, en el primero se evalúan las expresiones correspondientes a los triples alojados en las posiciones L, L+2, L+4 , etc., mientras que en el siguiente recuadro se evaluarán las expresiones correspondientes a los triples de las posiciones impares, ésto es, FD(I) representa las expresiones correspondientes a los triples de posición par, mientras que F(I) serán las expresiones correspondientes a los triples de posición impar.

Nótese que: $F(1+i) + F(8-i) + FD(i+1) + FD(8-i) = 1$; $i=0,1,2,3$

A continuación se entra en un bucle, indicado en la parte derecha del diagrama, en el cual se van a generar los diferentes triples que conformarán las ramas de una trayectoria del árbol Bayesiano, cada uno de los *bits* que conformarán los triples se representa por las variables IV1, IV2 e IV3 (5), la variable INT sirve como un índice a los factores F y FD, y se calcula en función del valor binario convertido a decimal más uno del triple generado VE. Enseguida se asigna la expresión de probabilidad a la rama cuyo triple se ha generado en función de la posición de ésta: (7) FD(INT) si la rama es de posición par, (8) F(INT) si la rama es de posición impar. (9) Cada factor calculado se va multiplicando a la varible PROD que indica la probabilidad de la trayectoria generada al terminar de procesar el bucle.

(10) SUM representa la probabilidad de detectar dos diferencias en L=6, 8, y 10 y cero o una diferencia en L=4, 5, 7, y 9 si LTEDO=10 (DD), o sea hacer un cambio de decisión en cuanto a la tercia supuesta; cada vez que se calcula la probabilidad de una trayectoria del árbol (PROD), que equivale a procesar el bucle descrito anteriormente, se acumula este valor a la probabilidad SUM dado que PROD involucra todas las diferencias dobles detectadas en su trayectoria correspondiente, de manera que una vez generadas todas las trayectorias que forman el árbol, el algoritmo obtiene el valor total de SUM.

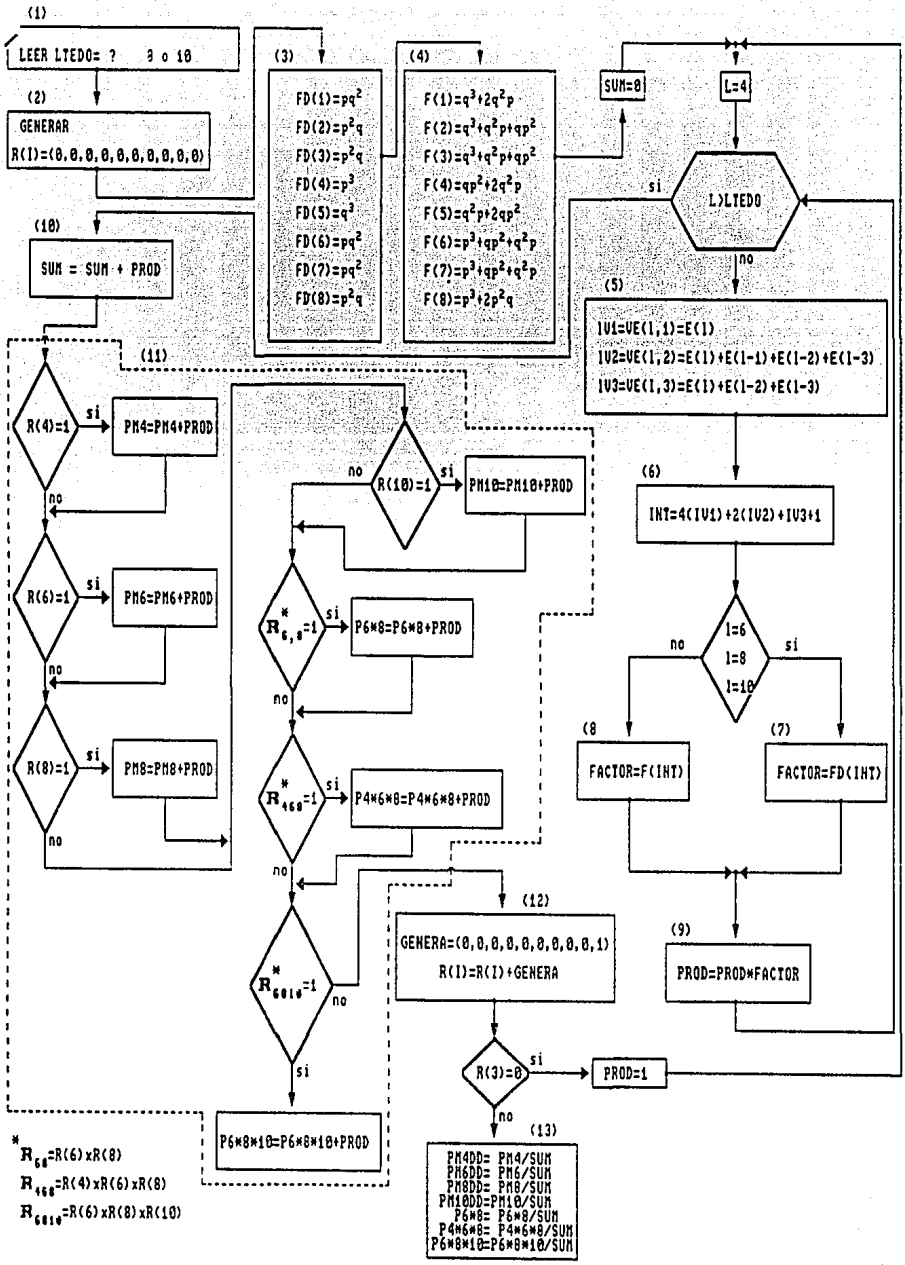


Fig. 3.6 Diagrama de flujo

(11) Cada diamante que cae dentro del área encerrada por la línea punteada a la izquierda del diagrama incluye la expresión de tomar una decisión erronea en una o varias posiciones pares de la trayectoria recién generada, ésta expresión se representa como R(I)=1 donde I indica la posición o posiciones sobre las que se tomó la decisión; si la expresión es verdadera, entonces la trayectoria recién generada incluye la condición y por tanto se procede a acumular la probabilidad dicha trayectoria, al igual que SUM, una vez generadas todas las trayectorias del árbol la suma final de estas probabilidades indicará la probabilidad total de cada una de las expresiones incluidas en cada diamante, por ejemplo: el valor final de PM4 indicará la probabilidad total de tomar la decisión erronea en L=4 o sea R(4)=1 y de que se detecten dos diferencias (cambio de triple supuesto por su conjugado) en L=6, 8, y 10..

(12) El algoritmo genera una nueva secuencia binaria R(I) que codificará dentro del bucle para generar una nueva trayectoria, por ejemplo: para una secuencia de longitud 10 la secuencia inicial R(I) consiste de puros ceros **0000000000**, en este punto el algoritmo genera la secuencia **0000000001** y continua su proceso en dirección del bucle, al llegar nuevamente a (12) el algoritmo genera ahora la secuencia **0000000010** que como se puede observar es el consecutivo binario de la secuencia anteriormente generada; de acuerdo a esto la siguiente secuencia por generar será **0000000011**.

Para generar en (12) una nueva secuencia en forma consecutiva el algoritmo se basa en la detección de unos consecutivos al final de la secuencia previamente generada, revisando a partir del cuarto *bit* el valor de cada uno de la misma. Bajo ésta condición se cambia el cero anterior a los unos detectados por un 1, y todos los unos por ceros cambiando siempre el último *bit*; así se tiene que para generar una nueva secuencia dada la secuencia previa **0000000111**, se cambia el séptimo *bit* (0) por un 1, el octavo y noveno *bits* (1) por ceros y finalmente el décimo *bit* (1) por un cero obteniendo **0000001000**.

Si al llegar a (12) la nueva trayectoria generada es **0010000000** entonces el algoritmo continua su proceso hacia (13) que es el punto donde se procede a evaluar la probabilidad condicional asociada a cada uno de los eventos definidos en el apartado 3.5 para poder determinar si lo más probable es que se tenga 1 error, dado que se han detectado dos diferencias en las posiciones L, L+2 y L+4 (si LTEDO=10) o en L y L+2 (si LTEDO=8), valores que son reflejados en la tabla de la figura 3.4. Recuérdese que R(I) representa la secuencia ya descodificada.

3.8 Listado del programa de desempeño óptimo del descodificador

El siguiente listado es el programa fuente que se puede obtener a partir del diagrama de flujo del apartado 3.7 y fué utilizado para generar la tabla 3.4 variando la probabilidad de error en el canal desde 0.01 hasta 0.99 y como se aprecia está escrito en Fortran IV;

éste programa se corrió en una computadora personal utilizando un compilador Fortran para DOS de microsoft y fué escrito originalmente por el M. en I. Fernando Lepe Casillas utilizando una laboriosa rutina (PSUBG) para el cálculo de los factores de probabilidad de las ramas de posición impar la cual fué sustituida en el listado aquí incluido por las expresiones que calculan directamente dichos factores, esto con fines de comprobación. Un objetivo del programa es obtener la decisión para descodificación óptima del algoritmo. Es decir, que decisiones muy probablemente contienen error cuando se han detectado dos diferencias y deben cambiarse por su complemento binario.

```
C     CALCULO  DE  LA PROBABILIDAD DE ERROR DADAS DOS
C     DIFERENCIAS ALTERNADAS
C     DEFINICION DE PROBABILIDADES DADAS
C     LAS 3 DESCODIFICACIONES ANTERIORES.
C     PMC=PROERR EN EL CANAL; SUM=PROBABILIDAD DE DOS
C     DIFERENCIAS EN LOS TRIPLES DE     POSICIONES 6,8,10
C     LTEDO=LIMITE DEL CICLO DO
       INTEGER R(10), RP
       DIMENSION F(8), FD(8)
       COMMON/PMC/PMC
       TYPE 205
205    FORMAT(2X,'DAME 8 O 10')
       ACCEPT 206,LTEDO
206    FORMAT(I2)
       PMC=0.01
97     DO109I=1,10
109    R(I)=0
       IF(PMC.GT.1.)GOTO98
       TYPE302,PMC
302    FORMAT(15X,'PMC=',F8.6)
C      COMIENZA EL CALCULO DE LOS FACTORES
       FD(1)=PMC*(1.-PMC)**2
       FD(2)=PMC**2*(1.-PMC)
       FD(3)=FD(2)
       FD(4)=PMC**3
       FD(5)=(1.-PMC)**3
       FD(6)=FD(1)
       FD(7)=FD(1)
       FD(8)=FD(2)
       Q=1.-PMC
       F(1)=Q**3+2*Q**2PMC
       F(2)=Q**3+Q**2*PMC+Q*PMC**2
       F(3)=F(2)
       F(4)=PMC**2*Q+2*PMC*Q**2
       F(5)=PMC*Q**2+2*PMC**2*Q
       F(6)=PMC**3+PMC**2*Q+PMC*Q**2
       F(7)=F(6)
       F(8)=PMC**3+2*PMC**2*Q
       DO27LUN=1,8
C FTST DE TEST=PRUEBA
27     FTST=FTST+F(LUN)
```

```
D       TYPE304,FTST
D 304 FORMAT(15X,'FTST=',F8.4)
C CALCULO DE LA PROB. EN UNA RAMA DEL ARBOL BAYESIANO
        PM4=0
        PM6=0
        PM8=0
        PM10=0
        P6X8=0
        P4X6X8=0
        P6X8X1=0
        SUM=0
70      PROD=1
        DO80I=4,LTEDO
        CALLSMOD2(R(I),R(I-1),R(I-2),R(I-3),IV1,IV2,IV3)
        INT=IV1*4+IV2*2+IV3+1
        IF(I.EQ.6.OR.I.EQ.8.OR.I.EQ.10)GOTO101
        FACTOR=F(INT)
        GOTO81
101     FACTOR=FD(INT)
81      PROD=PROD*FACTOR
80      CONTINUE
        SUM=SUM+PROD
        IF(R(4).EQ.1)PM4=PM4+PROD
        IF(R(6).EQ.1)PM6=PM6+PROD
        IF(R(8).EQ.1)PM8=PM8+PROD
        IF(R(10).EQ.1)PM10=PM10+PROD
        IF(R(6)*R(8).EQ.1)P6X8=P6X8+PROD
        IF(R(4)*R(6)*R(8).EQ.1)P4X6X8=P4X6X8+PROD
        IF(R(6)*R(8)*R(10).EQ.1)P6X8X1=P6X8X1+PROD
C GENERACION DE UNA RAMA DEL ARBOL BAYESIANO
        DO90J=3,LTEDO-1
        RP=1
        DO91L=J+1,LTEDO
91      RP=RP*R(L)
90      IF(RP.EQ.1)R(J)=1-R(J)
        R(LTEDO)=1-R(LTEDO)
        IF(R(3).EQ.0)GOTO70
C APLICANDO LAS ECUACIONES DE BAYES
        PM4DD=PM4/SUM
        PM6DD=PM6/SUM
        PM8DD=PM8/SUM
        PM10DD=PM10/SUM
        P6X8=P6X8/SUM
        P4X6X8=P4X6X8/SUM
        P6X8X1=P6X8X1/SUM
C ESCRITURA DE RESULTADOS
D       TYPE215,PM4DD,PM6DD,PM8DD,PM10DD
D215    FORMAT(2X,'PM4DD=',F8.6,'PM6DD=',F8.6,'PM8DD=',F8.6,
D       1' PM10=',F8.6)
D       TYPE 301,P6X8,P4X6X8,P6X8X1
D301    FORMAT(2X,'P6X8=',F8.6,'P4X6X8=',F8.6,'P6X8X1=',F8.6)
        TYPE3001,SUM
```

```
3001   FORMAT(2X,'SUM= ',F10.6)
       PMC=PMC+.02
       GOTO97
98     CALLEXIT
       END
```

3.9 Gráficas de la probabilidad de desempeño del descodificador

La gráfica de la figura 3.7 indica el desempeño del algoritmo en función de PMC y corresponde a los valores de la tabla de la figura 3.4.

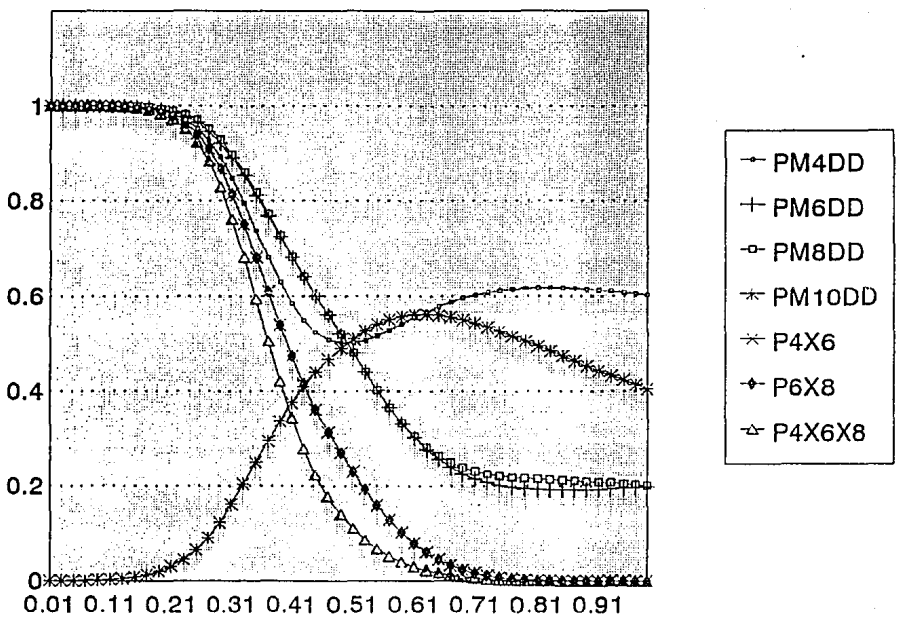


Figura 3.7 Desempeño del Algoritmo en función de PMC (PMC = Probabilidad de Error en el Canal)

3.10 Pruebas de Simulación

Se describe en éste apartado la simulación software del desempeño del algoritmo analizado a lo largo de éste capítulo decodificando secuencias recibidas a la salida de un canal digital, el cual agrega ruido en forma aleatoria a las correspondientes secuencias transmitidas que a su vez representan un mensaje codificado.

El desarrollo del programa simulador fué concebido originalmente por el M. en I. Fernando Lepe Casillas cuyo programa fuente está escrito en lenguaje FORTRAN IV; el programa de simulación que se presenta en éste apartado toma como base el programa escrito en FORTRAN IV que en principio es traducido al lenguaje PASCAL/VS para ejecutarse en una computadora IBM mainframe modelo 9121 y que con el fin de obtener una simulación confiable se modificó básicamente la generación de patrones de error, esto es, la simulación del ruido del canal y se aumentó el número de pruebas para cada valor de PMC (probabilidad de error en el canal) en el rango de 0.01 a 0.99.

Esquema de simulación.

La figura 3.8 muestra gráficamente las diferentes acciones que constituyen la simulación:

(1) La fuente genera un mensaje digital que para fines de simulación será una secuencia de 40 *bits* (todos cero) al que se la agregan cuatro bits de inicialización del codificador y tres bits de limpieza del codificador al inicio y al final de la secuencia respectivamente obteniendo un mensaje de 47 *bits* a la salida de la fuente.

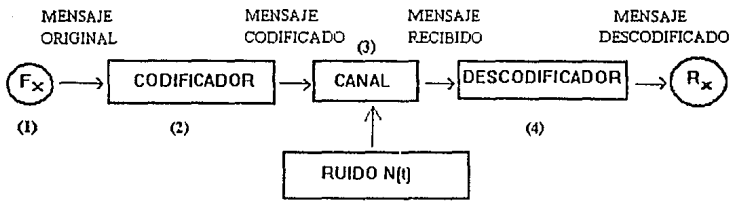


Figura 3.8 Diagrama a bloques de la simulación.

(2) El codificador aplica las ecuaciones 3.3, 3.4 y 3.5 al mensaje original para obtener el mensaje codificado compuesto de 47 triples, que en el programa simulador es el arreglo VE.

(3) El canal agrega mediante una suma or-exclusiva el ruido N al mensaje VE que en la figura está indicado como una función del tiempo pero que en la simulación no se conoce como tal, más bien es una varible aleatoria que en el programa escrito en FORTRAN IV tiene una distribución uniforme. Debido al cambio del lenguaje y el hardware utilizados la generación del ruido se hizo como sigue:

- Genera el número de errores a introducir en función de PMC tomándola como una probabilidad nominal sobre la longitud del mensaje codificado. Por ejemplo si PMC=0.01 y la longitud del mensaje codificado es de 120 *bits* entonces se debe tener un porcentaje de errores de 1% sobre los 120 *bits* del mensaje codificado.

- Las posiciones del mensaje en que se introducen los errores serán asignados aleatoriamente para lo cual se debe utilizar una función aleatoria confiable. El PASCAL/VS parece contar con una función RANDOM que genera números psuedoaleatorios confiablemente dado lo siguiente: la tabla de abajo indica que en cinco intentos se generan aleatoriamente números entre 0 y 9 mil veces en cada intento indicandose la frecuencia con que se generan en cada intento cada uno de ellos.

intento	1	2	3	4	5
número					
0	111	105	102	99	87
1	104	97	104	92	107
2	97	99	111	107	115
3	91	100	102	106	91
4	97	92	106	109	104
5	107	108	99	105	88
6	89	98	96	98	94
7	99	92	82	96	109
8	111	103	113	92	90
9	94	106	85	96	116

- Es importante tomar en cuenta las diferentes posiciones del mensaje en las que se pueden introducir un mismo número de errores lo cual genera diferentes posibilidades de patrones de error. El programa simulador debe cuidar de no generar el mismo patrón de error dos o más veces para un solo valor de PMC ya que esto le resta confiabilidad a la prueba. Surgen aquí las siguientes cuestiones:

a) ¿Cuántos patrones de error se deben generar para cada valor de PMC?

b) ¿Son suficientes los patrones de error generados para cada PMC para poder decir que el desempeño del descodificador, indicado por las estadísticas obtenidas durante la prueba, es confiable?

La respuesta al inciso a) es la siguiente: la longitud del mensaje codificado es de 4 triples (de inicialización del codificador) más 40 triples (del mensaje codificado) más tres triples (de limpieza del codificador), que sumados son 141 *bits* en total. La introducción de un error en 120 posiciones posibles al mensaje codificado implica tener

	C(120,1)	= 120 patrones de error diferentes
2 errores:	C(120,2)	= 7140 " "
3 errores:	C(120,3)	= 280840 " "
4 errores:	C(120,4)	= 8 214 570 " "
5 errores:	C(120,5)	= 190 578 024 " "
18 errores:	C(120,18)	= 1114 millones de billones "

.
.
.

60 errores: C(120,60) = $5.603294919*10^{70}*2^{35}$ " .

De lo anterior se observa que es prácticamente imposible probar todos los patrones de error posibles para cada PMC excepto quizá para los primeros tres valores de PMC a simular, dado que existe una limitación en tiempo y en costo. Así este análisis se limita a generar 120 patrones de error diferentes para PMC=0.01 y PMC=0.99; mientras que para cualquier otro nivel de PMC entre 0.01 y 0.99 se generarán 1000 patrones de error diferentes.

La pregunta del inciso b) queda contestada sabiendo que se pueden obtener conclusiones definitivas acerca del desempeño del algoritmo si se generan todos los patrones de error posibles para todo el rango de PMC lo cual, como ya se dijo, es prácticamente imposible. Sin embargo, los resultados del programa basan su confiabilidad en que cada patrón de error generado es diferente a cada uno de los ya generados anteriormente para un mismo valor de PMC y que cada vez que se ejecute el programa simulador se generarán diferentes grupos de patrones de error.

Un patrón de error se representa en el programa simulador por el arreglo ERGAUS, que sumado al mensaje codificado VE da el mensaje recibido VR a la salida del canal, esto es,

$$VR = ERGAUS \oplus VE.$$

(4) El descodificador es la traducción directa del FORTRAN IV a PASCAL/VS del programa ideado por el M. en I. Fernando Lepe Casillas y simula solamente la combinación del evento cuya probabilidad se indica en la columna P4x6x8 de la tabla de la figura 3.4 que es el de interés en ésta prueba.

Obviamente el objetivo del descodificador es obtener a su salida el mensaje descodificado igual al mensaje original generado por la fuente de

información. En este bloque del programa simulador se agrega el cálculo de estadísticas de desempeño.

Programa Fuente del simulador

El programa fuente se compone de los siguientes módulos:

PROGRAMA PRINCIPAL.- incluye un ciclo cuyo parámetro de control es la probabilidad de error en el canal (PMC) que varía de 0.01 a 0.99, con incrementos de 0.02, dentro de éste se genera el número (LIM) de posibles errores que introduce el patrón de error a generar; en función de LIM se obtiene el número máximo de patrones de error a generar para el valor de PMC del ciclo corriente. Una vez generadas las posiciones erroneas por el procedimiento correspondiente indicadas en el programa principal por el arreglo POINT, éstas se asignan al patrón de error que viene a ser el arreglo ERGAUS. Otras tareas que ejecuta el programa principal son: vigilar que no se genere un patrón de error repetido; llamar a los procedimientos que ejecutan propiamente la codificación, la adición de ruido al mensaje codificado, y la descodificación del mensaje recibido.

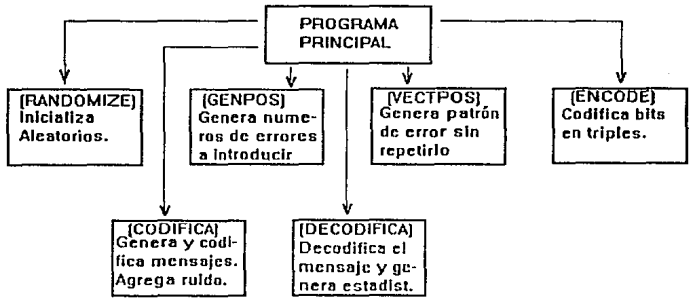


Figura 3.9 Diagrama a bloques del simulador.

RANDOMIZE.- Inicializa la función RANDOM tal como lo requiere la definición de dicha función para el PASCAL/VS.

GENPOS.- este módulo es llamado por el programa principal para calcular LIM en función de PMC.

VECTPOS.- este módulo genera el arreglo POINT que incluye las posiciones erróneas a introducir en el mensaje codificado para lo cual se debe conocer inicialmente el valor de LIM que le pasa el programa principal.

ENCODE.- éste módulo ejecuta la tarea básica de aplicar las ecuaciones 3.3 a 3.5.

CODIFICA.- se ejecutan aquí varias tareas, a saber:

- generar el mensaje de 40 *bits* de longitud representados en el arreglo E;
- llamar al procedimiento ENCODE para codificar el mensaje E, obteniendo el arreglo VE;
- agregar el ruido del canal ERGAUS al mensaje codificado hasta completar los 40 triples correspondientes a los 40 *bits* del mensaje.

DECODIFICA.- se ejecuta la tarea básica de comparar triples recibidos con triples supuestos previamente generados mediante el procedimiento ENCODE, para enseguida tomar las decisiones correspondientes a la corrección de errores de acuerdo al criterio del algoritmo. Adicionalmente éste módulo calcula el número de errores introducidos por ERGAUS sobre los 40 triples del mensaje codificado indicado por el parámetro TOTER que se imprime en el reporte de salida así como el parámetro DIFIN que indica el número de diferencias entre el mensaje original E y el mensaje recibido R; en función de éstos dos parámetros se calcula el porcentaje de errores introducidos al mensaje ECANAL=TOTER/120, y el porcentaje de errores no corregidos ECODEC=DIFIN/40.

A continuación se enlista el programa fuente:

```
PROGRAM ALGOR;
VAR
 REPET,GENERA:BOOLEAN;
 ARCHIVO:TEXT;
 POINT:ARRAY(.1..1000,1..141.) OF INTEGER;
 ANT:ARRAY(.1..120.) OF INTEGER;
 VE,VR,ERGAUS:ARRAY(.1..47,1..3.) OF INTEGER;
 NUE,E,RE:ARRAY(.1..47.) OF INTEGER;
 PMC:REAL;
LIM,NUMCOMB,NUMCOMP,COMBMAX,IDEN,IDEN2,NUMCOL,INDX,I,J,N,O,P
:INTEGER;
{*****************************************************
  PROGRAMA QUE DECODIFICA MENSAJES RECIBIDOS CON UNA
  PROBABILIDAD DE ERROR PMC, GENERA UN REPORTE
  DONDE SE DEPOSITA PMC, VE, ERGAUS, VR, E, R, Y PORCENT. DE
ERROR
 *****************************************************}
PROCEDURE RANDOMIZE;
{*****************************************************
  PROCEDIMIENTO PARA INICIALIZAR LA FUNCION RANDOM
  TOMANDO COMO SUBSTRING UN NUMERO DADO POR EL RELOJ
  DEL SISTEMA SIN QUE SE TOME DICHO SUBSTRING PARA UN
  PROPOSITO PARTICULAR; VARINT ES LA SEMILLA INICIAL Y
  VARREA TOMA EL VALOR ALEATORIO INICIAL
 *****************************************************}
```

```
VAR
  VARREA:REAL;
  FEC,HOR:ALFA;
  HORSTR:STRING(8);
  VARINT:INTEGER;
BEGIN
 DATETIME(FEC,HOR);
 WRITESTR(HORSTR,HOR);
 HORSTR:=SUBSTR(HORSTR,7,2);
 READSTR(HORSTR,VARINT);
 VARREA:=RANDOM(VARINT);
END;
FUNCTION RANDOMPOS:INTEGER;
{**** GENERA UN RANDOM ENTRE 13 Y 132  ****
          LLAMADO RANDOMPOS***********}
VAR
  VARRE2:REAL;
BEGIN
 VARRE2:=RANDOM(0);
 VARRE2:=VARRE2*1000;
 IF VARRE2>132 THEN VARRE2:=VARRE2/7.57;
 IF VARRE2<13 THEN
   VARRE2:=VARRE2+13;
   RANDOMPOS:=ROUND(VARRE2);
END;
PROCEDURE GENPOS(PERR:REAL;VAR CANTALE:INTEGER);
{*****************************************************
  PROCEDIMIENTO PARA CALCULO DE CANTIDAD DE ERRORES
  ALEATORIOS (CANTALE) A INTRODUCIR, EN FUNCION DE PERR
 *****************************************************}
VAR
 PORCENT: ARRAY(.1..120.) OF REAL;
 DELTA1,DELTA2,LIMIT: REAL;
 IND0,CONT: INTEGER;

BEGIN
'***ASIGNA PORCENTAJES DE ERROR DEL MENSAJE ****
  ****CODIFICADO A ELEMENTOS DE PORCENT**********
 FOR IND0:=1 TO 120 DO
  BEGIN
   PORCENT(.IND0.):=IND0/120;
  END;
 CONT:=1;
****COMPARA LA PREOBABILIDAD DE ERROR EN EL *****
**CANAL CON LOS PORCENTAJES DE ERROR  DEL MENSAJE*
 WHILE PORCENT(.CONT.)<PERR DO
  BEGIN
   CONT:=CONT+1;
  END;
 IF PORCENT(.CONT.)=PERR THEN
  BEGIN
```

```
***EL PORCENTAJE DE ERRORES A INTRODUCIR ***********
***ES IGUAL A LA PROBABILIDAD DE ERROR EN EL CANAL**
   LIMIT:=PORCENT(.CONT.)*120;
   CANTALE:=ROUND(LIMIT);
  END
 ELSE
  BEGIN
***EL PORCENTAJE DE ERRORES A INTRODUCIR****
***ES DIFERENTE A LA PROB. DE ERROR EN EL CANAL*****
   DELTA1:=PERR-PORCENT(.CONT-1.);
   DELTA2:=PORCENT(.CONT.)-PERR;
   IF DELTA1>DELTA2 THEN
    BEGIN
***EL PORCENTAJE DE ERRORES A INTRODUCIR********
***SERA IGUAL AL INMEDIATO INFERIOR A LA PROB.***
***DE ERROR EN EL CANAL*****************************
     LIMIT:=PORCENT(.CONT.)*120;
    END
   ELSE
***EL PORCENTAJE DE ERRORES A INTRODUCIR********
***SERA IGUAL AL INMEDIATO SUPERIOR A LA PROB.***
***DE ERROR EN EL CANAL
    BEGIN
     LIMIT:=PORCENT(.CONT-1.)*120;
    END;
    CANTALE:=ROUND(LIMIT);
  END;
END;
PROCEDURE VECTPOS(NUMALE:INTEGER);
{*********************************************************
 PROCEDIMIENTO QUE OBTIENE ERRORES EN POSICIONES ALEATORIAS;
ANT(.IND1.) ES UN ARREGLO QUE GUARDA LAS
POSICIONES (1 A 120) DONDE SE DEBEN INTRODUCIR LOS
 ERRORES EN EL MENSAJE CODIFICADO VE;
 *********************************************************}
VAR
 IND1,SUST,ALEAT,IND2: INTEGER;
BEGIN
{** INICIALIZA POINT *QUE CONTENDRA LOS
*** ERRORES A INTRODUCIR***************}
  FOR IND2:=1 TO 141 DO
   BEGIN
    POINT(.NUMCOMB,IND2.):=0;
   END;
  IND1:=1;
  ALEAT:=RANDOMPOS;
  ANT(.IND1.):=ALEAT;
  POINT(.NUMCOMB,ALEAT.):=1;
  SUST:=IND1;
  ALEAT:=RANDOMPOS;
{*******************************************************}
{** GENERA VECTOR CON POSICIONES ALEATORIAS ****}
```

```
{*****************************************************}
  WHILE IND1<>NUMALE DO
   BEGIN
    IF ALEAT<>ANT(.SUST.) THEN
     BEGIN
     ***** EL NUM. ALEATORIO GENERADO ES DIFERENTE**
     ***** AL NUM. ALEATORIO GENERADO PREVIAMENTE**
      IF SUST=1 THEN
       BEGIN
        IND1:=IND1+1;
        ANT(.IND1.):=ALEAT;
        POINT(.NUMCOMB,ALEAT.):=1;
        SUST:=IND1;
        ALEAT:=RANDOMPOS;
       END
      ELSE
       BEGIN
        SUST:=SUST-1;
       END;
     END
    ELSE
     BEGIN
      SUST:=IND1;
      ALEAT:=RANDOMPOS;
     END;
   END;
END;
PROCEDURE ENCODE(I1,I2,I3,I4:INTEGER;
                 VAR I5,I6,I7:INTEGER);
{*****************************************************
  PROCEDIMIENTO PARA CODIFICAR BITS EN TRIPLES
 *****************************************************}
VAR
IAUX1,IAUX2: INTEGER;
        BEGIN
         I5:=I1;
         I6:=0;
         I7:=0;
         IAUX1:=I1+I2+I3+I4;
         IAUX2:=I1+I3+I4;
         IF(IAUX1=1) OR (IAUX1=3)THEN I6:=1;
         IF(IAUX2=1) OR (IAUX2=3)THEN I7:=1;
        END;
PROCEDURE CODIFICA;
{*****************************************************
        CODIFICACION COMPLETA DE MENSAJES
*****************************************************}
VAR
K,LL,I,J,M,N: INTEGER;
 BEGIN
{*****************************************************}
{*** GENERESE EL MENSAJE DE 40 BITS Y CODIFIQUESE**}
```

```
{*********************************************************}
  FOR I:=1 TO 47 DO
   BEGIN
    E(.I.):=0;
   END;
  WRITELN(ARCHIVO);
  WRITE(ARCHIVO,'VE = ');
  WRITELN(ARCHIVO);
  FOR I:=4 TO 47 DO
   BEGIN
    N:=1;
    O:=2;
    P:=3;
    ENCODE(E(.I.),E(.I-1.),E(.I-2.),E(.I-3.),
           VE(.I,N.),VE(.I,O.),VE(.I,P.));
    WRITE(ARCHIVO,VE(.I,1.):1,VE(.I,2.):1,VE(.I,3.):1,' ');
   END;
  WRITELN(ARCHIVO);
{*********************************************************
** AGREGA RUIDO DEL CANAL AL MENSAJE CODIFICADO**
*********************************************************}
    M:=-5;
    FOR LL:=1 TO 5 DO
        BEGIN
         M:=M+10;
         N:=M+9;
         IF M=45 THEN N:=47;
         FOR K:=M TO N DO
           BEGIN
             FOR J:=1 TO 3 DO
              BEGIN
               VR(.K,J.):=(1-VE(.K,J.))*ERGAUS(.K,J.)+
                          VE(.K,J.)*(1-ERGAUS(.K,J.));
              END;
           END;
        END;
          WRITE(ARCHIVO,'ERGAUS= ');
          WRITELN(ARCHIVO);
          FOR K:=1 TO 47 DO
              BEGIN
              WRITE(ARCHIVO,ERGAUS(.K,1.):1,ERGAUS(.K,2.):1,
                            ERGAUS(.K,3.):1,' ');
              END;
          WRITELN(ARCHIVO);
          WRITE(ARCHIVO,'VR=      ');
          WRITELN(ARCHIVO);
          FOR K:=1 TO 47 DO
              BEGIN
     WRITE(ARCHIVO,VR(.K,1.):1,VR(.K,2.):1,VR(.K,3.):1,' ');
              END;
END;
```

```
PROCEDURE DECODIFICA;
{*******************************************************
 PROCEDIMIENTO QUE DECODIFICA MENSAJE RECIBIDO
 *******************************************************}
VAR
RETEN,ECANAL,ECODEC: REAL;
A,B,I,J,DIF,L,TOTER,ADIFIN, DIFIN,ATOTER,K: INTEGER;
VS: ARRAY(.1..3.) OF INTEGER;
{*******************************************************}
{*** INICIA LA DESCODIFICACION DEL MENSAJE*******}
{*******************************************************}
BEGIN
RETEN:=0;
WHILE RETEN=0 DO
 BEGIN
  A:=0;
  B:=0;
  FOR I:=1 TO 4 DO
   BEGIN
    RE(.I.):=E(.I.)
   END;
  FOR I:=5 TO 47 DO
   BEGIN
    RE(.I.):=VR(.I,1.);
    ENCODE(RE(.I.),RE(.I-1.),RE(.I-2.),RE(.I-3.),
           VS(.1.),VS(.2.),VS(.3.));
    DIF:=0;
    FOR J:=1 TO 3 DO
     BEGIN
      IF VR(.I,J.)<>VS(.J.) THEN DIF:=DIF+1;
     END;
    IF DIF>1 THEN RE(.I.):=1-RE(.I.);
    IF DIF=0 THEN A:=0;
    IF DIF=0 THEN B:=0;
    IF DIF<2 THEN
     BEGIN
     IF I=A+2 THEN A:=0
     END
    ELSE
    BEGIN
     IF A=0 THEN A:=I;
     IF I=A+1 THEN B:=I;
     IF I=A+2 THEN
      BEGIN
       RE(.I.):=1-RE(.I.);
       L:=I-2;
       RE(.L.):=1-RE(.L.);
       A:=0;
      END;
     IF I=B+2 THEN
      BEGIN
       RE(.I.):=1-RE(.I.);
```

```
        L:=I-2;
        RE(.L.):=1-RE(.L.);
        A:=0;
       END;
     END;
   END;
  TOTER:=0;
  FOR I:=5 TO 44 DO
   BEGIN
    FOR J:=1 TO 3 DO
     BEGIN
     TOTER:=TOTER+ERGAUS(.I,J.)
     END;
   END;
  WRITELN(ARCHIVO);
  DIFIN:=0;
  FOR I:=5 TO 44 DO
   BEGIN
    IF E(.I.)<>RE(.I.) THEN DIFIN:=DIFIN+1;
   END;
{***********************************************
**INICIA CALCULO DE PROCENTAJES DE ERROR EN EL**
**PROCESO DE CODIFICACION DESCODIFICACION ******
***********************************************}
  ATOTER:=TOTER;
  ADIFIN:=DIFIN;
  ECANAL:=ATOTER/120;
  ECODEC:=ADIFIN/40;
    WRITE(ARCHIVO,'NUE= ');
    FOR K:=5 TO 47 DO
     BEGIN
      WRITE(ARCHIVO,NUE(.K.):1,' ');
     END;
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'R= ');
    FOR K:=5 TO 44 DO
     BEGIN
      WRITE(ARCHIVO,RE(.K.):1,' ');
     END;
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'E= ');
    FOR K:=5 TO 44 DO
     BEGIN
      WRITE(ARCHIVO,E(.K.):1,' ');
     END;
    WRITELN(ARCHIVO);
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'TOTER= ',TOTER:8,' DIFIN=',DIFIN:8);
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'ECANAL= ',ECANAL:8,' ECODEC=',ECODEC:8);
   IF RETEN=1 THEN
   ELSE
```

```
    BEGIN
     RETEN:=1;
    END;
 END;
END;
{****  FIN DE PROCEDIMIENTO DECODIFICA ***************}
{****   PROGRAMA    PRINCIPAL ************************}
BEGIN
 REWRITE(ARCHIVO);
{*DA EL VALOR INICIAL DE LA PROB DE ERROR EN EL CANAL**}
 PMC:=0.05;
{****INICIALIZA LA FUNCION RANDOM DE PASCAL/VS*********}
 RANDOMIZE;
{****ESTABLECE  EL INTERVALO DE  PMC  PARA SIMULACION***}
 WHILE PMC<0.08 DO
  BEGIN
{*****CALCULA LA CANTIDAD DE ERRORES A INTRODUCIR*******}
{*****EN EL MENSAJE CODIFICADO (LIM) EN FUNCION DE PMC***}
   GENPOS(PMC,LIM);
   WRITELN(ARCHIVO);
   WRITE(ARCHIVO,'PROBABILIDAD DE ERROR EN EL CANAL:
',PMC:5);
   WRITELN('PMC=',PMC);
{*****************************************************}
{****CALACULESE EL NUMERO DE SIMULACIONES *****}
{****A EJECUTAR, COMBMAX, EN FUNCION DE LIM*****}
{*****************************************************}
   IF (LIM=1)OR(LIM=119) THEN
    BEGIN
     COMBMAX:=120
    END
   ELSE
    BEGIN
     COMBMAX:=1000;
    END;
{*  COMBMAX:=100; *****}
   WRITELN('COMBINACIONES=', COMBMAX);
{****INICIALIZA EL NUMERO DE SIMULACIONES NUMCOMB*****}
   NUMCOMB:=1;
{****GENERA LOS ERRORES A INTRODUCIR ********}
   VECTPOS(LIM);
{** GENERA ERGAUS ASIGNANDO LOS ERRORES A *********
{*** INTRODCIR EN LAS POSICIONES INDICADAS POR POINT*}
     INDX:=1;
     FOR I:=1 TO 47 DO
      BEGIN
       INDX:=INDX*I;
       FOR J:=0 TO 2 DO
         BEGIN
          ERGAUS(.I,J+1.):=POINT(.NUMCOMB,J+3*INDX-2.);
         END;
       INDX:=1;
```

```
      END;
{***CALCULESE LA CODIFICACION, TRANSMISION, Y DESCO-*
****DIFICACION DEL MENSAJE ASI LAS ESTADISTICAS DE***
****ERRORES COMETIDOS EN LA DESCODIFICACION**********}
   CODIFICA;
   DECODIFICA;
{**** FIN DEL PROCESO CODIFICACON-DESCODIFICACION ****}
   NUMCOMB:=2;
   NUMCOMP:=1;
{***ESTABLECE EL CICLO DE SIMULACIONES******}
   WHILE NUMCOMB<COMBMAX DO
    BEGIN
{*** GENERA VECTOR ALEATORIO NO REPETIDO ***********}
     REPEAT
      GENERA:=TRUE;
      REPET:=FALSE;
      VECTPOS(LIM);
      WHILE NUMCOMP<>NUMCOMB DO
       BEGIN
        IDEN:=0;
        FOR NUMCOL:=1 TO 141 DO
         BEGIN
         IF POINT(.NUMCOMB,NUMCOL.)=POINT(.NUMCOMP,NUMCOL.)
        THEN
          IDEN:=IDEN+1;
         END;
        IF IDEN=141 THEN
         BEGIN
          IDEN2:=NUMCOMP;
          NUMCOMP:=NUMCOMB;
          REPET:=TRUE;
         END
        ELSE
         BEGIN
          NUMCOMP:=NUMCOMP+1;
         END;
       END;
      IF REPET=FALSE THEN
       BEGIN
       GENERA:=FALSE;
       END
      ELSE
       BEGIN
       WRITELN('VECTOR NUMCOMB:',NUMCOMB:4);
       WRITELN('VECTOR NUMCOMP:',IDEN2:4);
       NUMCOMP:=1;
       END;
     UNTIL GENERA=FALSE;
{*** FIN DE GENERA VECTOR ALEATORIO NO REPETIDO  ****}
{** GENERA ERGAUS ***}
     INDX:=1;
     FOR I:=1 TO 47 DO
```

```
      BEGIN
       INDX:=INDX*I;
       FOR J:=0 TO 2 DO
         BEGIN
          ERGAUS(.I,J+1.):=POINT(.NUMCOMB,J+3*INDX-2.);
         END;
       INDX:=1;
      END;
{**** CALCULA PROCESO DE CODIFICACION-DESCODIFICAC.**}
     CODIFICA;
     DECODIFICA;
{**** FIN DEL PROCESO CODIFICACION-DESCODIFICACION****}
     NUMCOMB:=NUMCOMB+1;
     NUMCOMP:=1;
    END;
   PMC:=PMC+0.02;
  END;
CLOSE(ARCHIVO);
END.
{****    FIN DE PROGRAMA     PRINCIPAL ******************}
```

Simulador del descodificador de Viterbi

Para hacer un buen estudio sobre el descodificador propuesto y por consiguiente obtener buenas conclusiones de los resultados obtenidos es necesario realizar una comparación de éste descodificador con otro que tenga características de funcionamiento muy confiables como es el caso del descodificador de máxima verosimilitud o de Viterbi. Tomando como referencia este descodificador se obtendrán las conclusiones necesarias sobre las características de ambos descodificadores.

Programa Fuente del simulador del descodificador de Viterbi

El programa fuente del simulador del descodificador de Viterbi se compone de los mismos módulos que el programa del simulador del algoritmo propuesto, debido a que se calcularan los mismos factores mediante las mismas variables y se generarán aleatoriamente los vectores de error mediante el procedimiento descrito en el apartado anterior.

Unicamente se cambiará la subrutina o procedimiento que se refiere a la descodificación del mensaje llamada DESCODIFICA por VITERBI. El fin de respetar las variables y los cálculos de los factores es para poder tener una comparación real entre ambos métodos de descodificación. En base a esto únicamente se explicará el funcionamiento del procedimiento o subrutina en donde se aplica el método de descodificación de máxima verosimilitud o de Viterbi.

PROGRAMA PRINCIPAL.- incluye un ciclo cuyo parámetro de control es la probabilidad de error en el canal (PMC) , dentro de éste se genera el número (LIM) de posibles errores. Una vez generadas las posiciones erróneas se asignan al patrón de error ERGAUS. Otras tareas que ejecuta el programa principal son: llamar a los procedimientos que ejecutan propiamente la codificación, la adición de ruido al mensaje codificado, y la descodificación del mensaje recibido.

VITERBI.- Este procedimiento se encarga de generar todas las ramas que conforman el enrejado, etapa por etapa, y calcula la distancia de Hamming de cada una con respecto al correspondiente triple recibido de VR. Estas distancias se guardan en cada etapa en una matriz de distancias MATDIS.

Al mismo tiempo se va obteniendo el mensaje perteneciente a cada triple recibido en otra matriz llamada MATMEN. Este proceso se lleva a cabo en los procedimientos ETA1, ETA2, ETA3, MAT1, MAT2 y MAT3. Después de obtener estas matrices hasta la etapa 3 se calculan las distancias de cada trayectoria posible mediante el procedimiento DISTRAY.

De la etapa 4 en adelante se tiene que discriminar de par en par 8 de 16 trayectorias supuestas con distancia mayor . De las trayectorias restantes también llamadas sobrevivientes se obtiene su distancia y el bit de mensaje y se colocan dentro de las matrices MATDIS y MATMEN respectivamente. Todo este proceso se lleva a cabo en ETA4.

Ya procesadas las 40 etapas o 40 triples enviados, se decide en base a cúal de las ocho trayectorias sobrevivientes es la distancia mínima, tomandose el mensaje R asociado a esa trayectoria, como el mensaje enviado E. Este proceso se lleva a cabo dentro del procedimiento MENSAJE.

Adicionalmente este programa, al igual que el simulador anterior, calcula el número de errores introducidos por ERGAUS sobre los 40 triples del mensaje codificado indicado por el parámetro TOTER que se imprime en el reporte de salida así como el parámetro DIFIN que indica el número de diferencias entre el mensaje original E y el mensaje recibido R; en función de estos dos parámetros se calcula el porcentaje de errores introducidos al mensaje ECANAL=TOTER/120, y el porcentaje de errores no corregidos ECODEC=DIFIN/40.

En la siguiente página se enlista el programa fuente:

```
PROGRAM ALGOR;
VAR
 REPET,GENERA:BOOLEAN;
 ARCHIVO:TEXT;
 POINT:ARRAY(.1..1000,1..141.) OF INTEGER;
 ANT:ARRAY(.1..120.) OF INTEGER;
 VE,VR,ERGAUS:ARRAY(.1..47,1..3.) OF INTEGER;
 NUE,EN,RE:ARRAY(.1..47.) OF INTEGER;
 PMC:REAL;
LIM,NUMCOMB,NUMCOMP,COMBMAX:INTEGER;
IDEN,IDEN2,NUMCOL,INDX,I,J,N,O,P:INTEGER;
DRAMA:INTEGER;
ETAPA:INTEGER;
{***REN'S:INDICES DE ASIGNACION DE DISTANCIAS A LAS****
****              RAMAS DEL ENREJADO                 *****}
REP,RENI,RENJ,RENK,RENL,RENM,RENN,RENO,RENP:INTEGER;

{***DXY'S:DISTANCIA DE RAMA QUE VA DEL NODO X AL NODO Y ***
****DEL ENREJADO, DONDE XY=AA,AB,BC,BD,CE,CF,DG,DH      ****}
    DAA,DAB,DBC,DBD,DCE,DCF,DDG,DDH,VECTOR: INTEGER;
    S,T: INTEGER;

{***DT: GUARDA LA DIST. TOTAL DE LAS 8 SOBREVIVIENTES***}
    DT:ARRAY(.1..8.) OF INTEGER;

{***RAMA:ARREGLO QUE GUARDA LOS VALORES DE CADA UNO DE***}
{***LOS OCHO ESTADOS DEL CODIFICADOR*****}
    RAMA: ARRAY(.1..8,1..3.) OF INTEGER;

{***MATDIS:ARREGLO QUE GUARDA LAS DISTANCIAS DE CADA UNA***}
{***DE LAS RAMAS QUE FORMAN LAS TRAYECTORIAS SOBREVIV.*****}
    MATDIS: ARRAY(.1..8,1..43.) OF INTEGER;

{***MATMEN:ARREGLO QUE GUARDA LOS MENSAJESASOCIADOS A C/U**}
{***DE LAS RAMAS QUE FORMAN LAS TRAYECTORIAS SOBREVIV.*****}
    MATMEN: ARRAY(.1..8,1..43.) OF INTEGER;
{***VT Y VM GUARDAN TEMPORALMENTE LOS MENSAJES DE ***
****LAS TRAYECTORIAS SOBREVIVIENTES Y SUS CORRES- ***
****PONDIENTES DISTANCIAS                          ***}
    VT: ARRAY(.1..8,1..43.) OF INTEGER;
    VM: ARRAY(.1..8,1..43.) OF INTEGER;

{***LAS CONSTANTES A, B, C,...,H DEFINEN LOS ESTADOS***
****POSIBLES DEL CODIFICADOR                      *********}
CONST A=1;
      B=2;
      C=3;
      D=4;
      E=5;
      F=6;
      G=7;
      H=8;
```

```
PROCEDURE RANDOMIZE;
{*********************************************************
  PROCEDIMIENTO PARA INICIALIZAR LA FUNCION RANDOM
 *********************************************************}
VAR
  VARREA:REAL;
  FEC,HOR:ALFA;
  HORSTR:STRING(8);
  VARINT:INTEGER;
BEGIN
 DATETIME(FEC,HOR);
 WRITESTR(HORSTR,HOR);
 HORSTR:=SUBSTR(HORSTR,7,2);
 READSTR(HORSTR,VARINT);
 VARREA:=RANDOM(VARINT);
END;
FUNCTION RANDOMPOS:INTEGER;
{**** GENERA UN RANDOM ENTRE 13 Y 132  ****************}
VAR
  VARRE2:REAL;
BEGIN
 VARRE2:=RANDOM(0);
 VARRE2:=VARRE2*1000;
 IF VARRE2>132 THEN VARRE2:=VARRE2/7.57;
 IF VARRE2<13 THEN
   VARRE2:=VARRE2+13;
   RANDOMPOS:=ROUND(VARRE2);
END;
PROCEDURE GENPOS(PERR:REAL;VAR CANTALE:INTEGER);
{*********************************************************
  PROCEDIMIENTO PARA CALCULO DE CANTIDAD DE ERRORES
  A INTRODUCIR, EN FUNCION DE PERR
 *********************************************************}
VAR
 PORCENT: ARRAY(.1..120.) OF REAL;
 DELTA1,DELTA2,LIMIT: REAL;
 IND0,CONT: INTEGER;
BEGIN
 FOR IND0:=1 TO 120 DO
  BEGIN
   PORCENT(.IND0.):=IND0/120;
  END;
 CONT:=1;
 WHILE PORCENT(.CONT.)<PERR DO
  BEGIN
   CONT:=CONT+1;
  END;
 IF PORCENT(.CONT.)=PERR THEN
  BEGIN
   LIMIT:=PORCENT(.CONT.)*120;
   CANTALE:=ROUND(LIMIT);
  END
```

```
 ELSE
  BEGIN
   DELTA1:=PERR-PORCENT(.CONT-1.);
   DELTA2:=PORCENT(.CONT.)-PERR;
   IF DELTA1>DELTA2 THEN
    BEGIN
     LIMIT:=PORCENT(.CONT.)*120;
    END
   ELSE
    BEGIN
     LIMIT:=PORCENT(.CONT-1.)*120;
    END;
    CANTALE:=ROUND(LIMIT);
  END;
END;
PROCEDURE VECTPOS(NUMALE:INTEGER);
{*********************************************************
 PROCEDIMIENTO QUE OBTIENE ERRORES EN POSICIONES ALEATORIAS
 *********************************************************}
VAR
 IND1,SUST,ALEAT,IND2: INTEGER;
BEGIN
{** INICIALIZA POINT ********************************}
  FOR IND2:=1 TO 141 DO
   BEGIN
    POINT(.NUMCOMB,IND2.):=0;
   END;
  IND1:=1;
  ALEAT:=RANDOMPOS;
  ANT(.IND1.):=ALEAT;
  POINT(.NUMCOMB,ALEAT.):=1;
  SUST:=IND1;
  ALEAT:=RANDOMPOS;
{*****************************************************}
{** GENERA VECTOR CON POSICIONES ALEATORIAS ***}
{*****************************************************}
  WHILE IND1<>NUMALE DO
   BEGIN
    IF ALEAT<>ANT(.SUST.) THEN
     BEGIN
      IF SUST=1 THEN
       BEGIN
        IND1:=IND1+1;
        ANT(.IND1.):=ALEAT;
        POINT(.NUMCOMB,ALEAT.):=1;
        SUST:=IND1;
        ALEAT:=RANDOMPOS;
       END
      ELSE
       BEGIN
        SUST:=SUST-1;
       END;
```

```
     END
    ELSE
     BEGIN
      SUST:=IND1;
      ALEAT:=RANDOMPOS;
     END;
   END;
END;
PROCEDURE ENCODE(I1,I2,I3,I4:INTEGER;
                 VAR I5,I6,I7:INTEGER);
{*****************************************************
  PROCEDIMIENTO PARA CODIFICAR BITS EN TRIPLES
 *****************************************************}
VAR
IAUX1,IAUX2: INTEGER;
        BEGIN
         I5:=I1;
         I6:=0;
         I7:=0;
         IAUX1:=I1+I2+I3+I4;
         IAUX2:=I1+I3+I4;
         IF(IAUX1=1) OR (IAUX1=3)THEN I6:=1;
         IF(IAUX2=1) OR (IAUX2=3)THEN I7:=1;
        END;
PROCEDURE CODIFICA;
{*****************************************************
        CODIFICACION COMPLETA DE MENSAJES
 *****************************************************}
VAR
K,LL,I,J,M,N: INTEGER;
 BEGIN
{*****************************************************}
{*** GENERESE MENSAJE DE 40 BITS Y CODIFIQUESE ***}
{*****************************************************}
  FOR I:=1 TO 47 DO
   BEGIN
    EN(.I.):=0;
   END;
  WRITELN(ARCHIVO);
{*****************************************************
 ****   AGREGA RUIDO DEL CANAL AL MENSAJE CODIFICADO****
 *****************************************************}
    M:=-5;
    FOR LL:=1 TO 5 DO
        BEGIN
         M:=M+10;
         N:=M+9;
         IF M=45 THEN N:=47;
         FOR K:=M TO N DO
           BEGIN
             FOR J:=1 TO 3 DO
              BEGIN
```

```
                VR(.K,J.):=(1-VE(.K,J.))*ERGAUS(.K,J.)+
                           VE(.K,J.)*(1-ERGAUS(.K,J.));
              END;
            END;
          END;
 END;

PROCEDURE MENSAJE;
{******INICIA PROCEDIMIENTO MENSAJE *********}
{***PROCEDIMIENTO QUE ESTIMA CUAL ES LA******
****TRAYECTORIA ENVIADA UNA VEZ GENERADAS ***
*********      LAS SOBREVIVIENTES       *******}
VAR
   K,TOTER,DIFIN,ADIFIN,ATOTER,EDO,R,J,I:INTEGER;
   ECANAL,ECODEC:REAL;
BEGIN
{***INDICA CON LA VARIABLE "S" LA MINIMA     **************
****DISTANCIA HAMMING DE LAS TRAYECOTIAS SOBREVIVIENTES***}
   S:=1;
   FOR J:=2 TO 8 DO
   BEGIN
    T:=J;
    IF DT(.T.)<DT(.S.) THEN
    BEGIN
     S:=T
    END
    ELSE
    BEGIN
    IF DT(.S.)=DT(.T.)THEN
      BEGIN
{***DADO QUE HAY DISTANCIAS IGUALES, ELIGE UNA ***
*************      ALEATORIAMENTE          ********}
       RANDOMIZE;
       IF RANDOMPOS<73 THEN
       S:=T;
      END;
    END;
   END;
  TOTER:=0;
  FOR I:=5 TO 44 DO
   BEGIN
    FOR J:=1 TO 3 DO
     BEGIN
     TOTER:=TOTER+ERGAUS(.I,J.)
     END;
   END;
  DIFIN:=0;
  FOR I:=5 TO 44 DO
   BEGIN
    IF EN(.I.)<>MATMEN(.S,I-4.) THEN DIFIN:=DIFIN+1;
   END;
  ATOTER:=TOTER;
```

```
  ADIFIN:=DIFIN;
  ECANAL:=ATOTER/120;
  ECODEC:=ADIFIN/40;
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'R= ');
    FOR K:=5 TO 44 DO
     BEGIN
      WRITE(ARCHIVO,MATMEN(.S,K-4.):1,' ');
     END;
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'E= ');
    FOR K:=5 TO 44 DO
     BEGIN
      WRITE(ARCHIVO,EN(.K.):1,' ');
     END;
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'TOTER= ',TOTER,' DIFIN=',DIFIN);
    WRITELN(ARCHIVO);
    WRITE(ARCHIVO,'ECANAL= ',ECANAL,' ECODEC=',ECODEC);
END;
{******TERMINA PROCEDIMIENTO MENSAJE *********}

PROCEDURE EXPO(XPONEN:INTEGER;VAR POTENCIA:INTEGER);
{******INICIA PROCEDIMIENTO EXPO ************
***CALCULA 2 ELEVADO A LA "XPONEN" POTENCIA***}
VAR
 ACUM,BASE: INTEGER;
 BEGIN
  ACUM:=0;
  BASE:=2;
  POTENCIA:=1;
  REPEAT;
   BEGIN
    POTENCIA:=POTENCIA*BASE;
    ACUM:=ACUM+1;
   END;
  UNTIL ACUM=XPONEN;
 END;
{******FIN DE PROCEDIMIENTO EXPO *********}

PROCEDURE DISRAM;
{******INICIA PROCEDIMIENTO DISRAM *********}
{***CALCULA LA DISTANCIA HAMMING ENTRE LA****
****RAMA RECIBIDA Y LA RAMA SUPUESTA      ****}
VAR I: INTEGER;
 BEGIN
  DRAMA:=0;
  FOR I:=1 TO 3 DO
   BEGIN
    IF RAMA(.VECTOR,I.)<>VR(.ETAPA+4,I.) THEN
     DRAMA:=DRAMA+1;
   END;
```

```
 END;
{******FIN DE PROCEDIMIENTO DISRAM *********}

PROCEDURE ETA1;
{******INICIA PROCEDIMIENTO ETA1 *********}
{***CALCULA LAS DISTANCIAS DE RAMA DE LA ETAPA 1***}
 BEGIN
    VECTOR:=A;
    DISRAM;
    DAA:=DRAMA;
    VECTOR:=B;
    DISRAM;
    DAB:=DRAMA;
 END;
{******FIN DE PROCEDIMIENTO ETA1 *********}

PROCEDURE ETA2;
{******INICIA PROCEDIMIENTO ETA2 *********}
{***CALCULA LAS DISTANCIAS DE RAMA DE LA ETAPA 2***}
 BEGIN
    VECTOR:=C;
    DISRAM;
    DBC:=DRAMA;
    VECTOR:=D;
    DISRAM;
    DBD:=DRAMA;
 END;
{******FIN DE PROCEDIMIENTO ETA2 *********}

PROCEDURE ETA3;
{******INICIA PROCEDIMIENTO ETA3 *********}
{***CALCULA LAS DISTANCIAS DE RAMA DE LA ETAPA 3***}
 BEGIN
    VECTOR:=E;
    DISRAM;
    DCE:=DRAMA;
    VECTOR:=F;
    DISRAM;
    DCF:=DRAMA;
    VECTOR:=G;
    DISRAM;
    DDG:=DRAMA;
    VECTOR:=H;
    DISRAM;
    DDH:=DRAMA;
 END;
{******FIN DE PROCEDIMIENTO ETA3 *********}

PROCEDURE MAT1;
{******INICIA PROCEDIMIENTO MAT1 *********}
{***ASIGNA DISTANCIAS HAMMING A LAS RAMAS DE LA ETAPA***
****1 DEL ENREJADO EN FUNCION DEL NUMERO DE ETAPA     ***}
```

```
VAR
  X,Y,I: INTEGER;
 BEGIN
     X:=ETAPA;
    EXPO(X,Y);
    REP:=8 DIV Y;
    FOR I:=1 TO REP DO
     BEGIN
      RENI:=I;
      MATDIS(.RENI,ETAPA.):=DAA;
      MATMEN(.RENI,ETAPA.):=RAMA(.A,1.);
      RENJ:=RENI+REP;
      MATDIS(.RENJ,ETAPA.):=DAB;
      MATMEN(.RENJ,ETAPA.):=RAMA(.B,1.);
     END;
 END;
{******FIN DE PROCEDIMIENTO MAT1 *********}

PROCEDURE MAT2;
{******INICIA PROCEDIMIENTO MAT2 *********}
{***ASIGNA DISTANCIAS HAMMING A LAS RAMAS DE LA ETAPA***
****2 DEL ENREJADO EN FUNCION DEL NUMERO DE ETAPA    ***}
VAR I: INTEGER;
 BEGIN
      I:=RENI;
      FOR I:=1 TO REP DO
       BEGIN
        RENK:=RENJ+I;
        MATDIS(.RENK,ETAPA.):=DBC;
        MATMEN(.RENK,ETAPA.):=RAMA(.C,1.);
        RENL:=RENK+REP;
        MATDIS(.RENL,ETAPA.):=DBD;
        MATMEN(.RENL,ETAPA.):=RAMA(.D,1.);
       END;
 END;
{******FIN DE PROCEDIMIENTO MAT2 *********}

PROCEDURE MAT3;
{******INICIA PROCEDIMIENTO MAT3 *********}
{***ASIGNA DISTANCIAS HAMMING A LAS RAMAS DE LA ETAPA***
****3 DEL ENREJADO EN FUNCION DEL NUMERO DE ETAPA    ***}
VAR I: INTEGER;
 BEGIN
  I:=RENI;
  FOR I:=1 TO REP DO
   BEGIN
    RENM:=RENL+I;
    MATDIS(.RENM,ETAPA.):=DCE;
    MATMEN(.RENM,ETAPA.):=RAMA(.E,1.);
    RENN:=RENM+REP;
    MATDIS(.RENN,ETAPA.):=DCF;
    MATMEN(.RENN,ETAPA.):=RAMA(.F,1.);
```

```
    RENO:=RENN+REP;
    MATDIS(.RENO,ETAPA.):=DDG;
    MATMEN(.RENO,ETAPA.):=RAMA(.G,1.);
    RENP:=RENO+REP;
    MATDIS(.RENP,ETAPA.):=DDH;
    MATMEN(.RENP,ETAPA.):=RAMA(.H,1.);
   END;
 END;
{******FIN DE PROCEDIMIENTO MAT3 *********}

PROCEDURE ETA4;
{******INICIA PROCEDIMIENTO ETA4 **************
***CALCULO DE DISTANCIAS DE TRAYECTORIAS*******
***SOBREVIVIENTES Y DISCRIMINACION DE LAS******
***NUEVAS TRAYECTORIAS CON MAYOR DISTANCIA*****}
VAR
 I,DEA,DTAA,DTEA,DEB,DTAB,DTEB,DFC,DTFC,DTBC: INTEGER;
 DFD,DTFD,DTBD,DGE,DTGE,DTCE,DGF,DTGF,DTCF: INTEGER;
 DHG,DTHG,DTDG,DHH,DTHH,DTDH: INTEGER;
 BEGIN
   VECTOR:=E;
{***CALCULA LA DISTANCIA HAMMING "DEA" DE LA NUEVA*****
****RAMA GENERADA                                    *****}
   DISRAM;
   DEA:=DRAMA;
{***CALCULA LAS DISTANCIAS DE TRECTORIA QUE LLEGAN******
****AL NODO "A" DE LA SIGUIENTE ETAPA:DTAA Y DTEA ******}
   DTAA:=DAA+DT(.A.);
   DTEA:=DEA+DT(.E.);
   IF DTAA>DTEA THEN
    BEGIN
{***DTEA RESULTO SER LA MENOR DISTANCIA; LA NUEVA ******
****TRAYECTORIA SOBREVIVIENTE SALE EN ESTA ETAPA POR****
****EL NODO E Y SE DIRIGE AL NODO A*}
     FOR I:=1 TO ETAPA-1 DO
      BEGIN
      MATDIS(.1,I.):=VT(.5,I.);
      MATMEN(.1,I.):=VM(.5,I.);
      END;
{***SE AGREGA LA MENOR DISTANCIA "DEA" A LA MATRIZ******
****DE DISTANCIAS, Y EL BIT MENSAJE DEL ULTIMO TRIPLE***
****DE LA NUEVA SOBREVIVIENTE A LA MATRIZ DE MENSAJES***}
     MATDIS(.1,ETAPA.):=DEA;
     MATMEN(.1,ETAPA.):=RAMA(.E,1.);
    END
   ELSE
    BEGIN
{***DTAA RESULTO SER LA MENOR DISTANCIA; LA NUEVA ******
****TRAYECTORIA SOBREVIVIENTE SALE EN ESTA ETAPA POR****
****EL NODO A Y SE DIRIGE AL NODO A*}
     FOR I:=1 TO ETAPA-1 DO
      BEGIN
```

```
      MATDIS(.1,I.):=VT(.1,I.);
      MATMEN(.1,I.):=VM(.1,I.);
      END;
{***SE AGREGA LA MENOR DISTANCIA "DAA" A LA MATRIZ******
****DE DISTANCIAS, Y EL BIT MENSAJE DEL ULTIMO TRIPLE***
****DE LA NUEVA SOBREVIVIENTE A LA MATRIZ DE MENSAJES***}
     MATDIS(.1,ETAPA.):=DAA;
     MATMEN(.1,ETAPA.):=RAMA(.A,1.);
    END;
   VECTOR:=F;
   DISRAM;
   DEB:=DRAMA;
   DTAB:=DAB+DT(.A.);
   DTEB:=DEB+DT(.E.);
   IF DTAB>DTEB THEN
    BEGIN
     FOR I:=1 TO ETAPA-1 DO
      BEGIN
      MATDIS(.2,I.):=VT(.5,I.);
      MATMEN(.2,I.):=VM(.5,I.);
      END;
     MATDIS(.2,ETAPA.):=DEB;
     MATMEN(.2,ETAPA.):=RAMA(.F,1.);
    END
   ELSE
    BEGIN
     FOR I:=1 TO ETAPA-1 DO
     BEGIN
      MATDIS(.2,I.):=VT(.1,I.);
      MATMEN(.2,I.):=VM(.1,I.);
     END;
     MATDIS(.2,ETAPA.):=DAB;
     MATMEN(.2,ETAPA.):=RAMA(.B,1.);
    END;
   VECTOR:=G;
   DISRAM;
   DFC:=DRAMA;
   DTBC:=DBC+DT(.B.);
   DTFC:=DFC+DT(.F.);
   IF DTBC>DTFC THEN
    BEGIN
     FOR I:=1 TO ETAPA-1 DO
     BEGIN
      MATDIS(.3,I.):=VT(.6,I.);
      MATMEN(.3,I.):=VM(.6,I.);
     END;
     MATDIS(.3,ETAPA.):=DFC;
     MATMEN(.3,ETAPA.):=RAMA(.G,1.);
    END
   ELSE
    BEGIN
    FOR I:=1 TO ETAPA-1 DO
```

```
 BEGIN
  MATDIS(.3,I.):=VT(.2,I.);
  MATMEN(.3,I.):=VM(.2,I.);
 END;
  MATDIS(.3,ETAPA.):=DBC;
  MATMEN(.3,ETAPA.):=RAMA(.C,1.);
 END;
VECTOR:=H;
DISRAM;
DFD:=DRAMA;
DTBD:=DBD+DT(.B.);
DTFD:=DFD+DT(.F.);
IF DTBD>DTFD THEN
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.4,I.):=VT(.6,I.);
   MATMEN(.4,I.):=VM(.6,I.);
  END;
  MATDIS(.4,ETAPA.):=DFD;
  MATMEN(.4,ETAPA.):=RAMA(.H,1.);
 END
ELSE
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.4,I.):=VT(.2,I.);
   MATMEN(.4,I.):=VM(.2,I.);
  END;
  MATDIS(.4,ETAPA.):=DBD;
  MATMEN(.4,ETAPA.):=RAMA(.D,1.);
 END;
VECTOR:=A;
DISRAM;
DGE:=DRAMA;
DTCE:=DCE+DT(.C.);
DTGE:=DGE+DT(.G.);
IF DTCE>DTGE THEN
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.5,I.):=VT(.7,I.);
   MATMEN(.5,I.):=VM(.7,I.);
  END;
  MATDIS(.5,ETAPA.):=DGE;
  MATMEN(.5,ETAPA.):=RAMA(.A,1.);
 END
ELSE
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.5,I.):=VT(.3,I.);
```

```
    MATMEN(.5,I.):=VM(.3,I.);
   END;
   MATDIS(.5,ETAPA.):=DCE;
   MATMEN(.5,ETAPA.):=RAMA(.E,1.);
  END;
VECTOR:=B;
DISRAM;
DGF:=DRAMA;
DTCF:=DCF+DT(.C.);
DTGF:=DGF+DT(.G.);
IF DTCF>DTGF THEN
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.6,I.):=VT(.7,I.);
   MATMEN(.6,I.):=VM(.7,I.);
  END;
  MATDIS(.6,ETAPA.):=DGF;
  MATMEN(.6,ETAPA.):=RAMA(.B,1.);
 END
ELSE
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.6,I.):=VT(.3,I.);
   MATMEN(.6,I.):=VM(.3,I.);
  END;
  MATDIS(.6,ETAPA.):=DCF;
  MATMEN(.6,ETAPA.):=RAMA(.F,1.);
 END;
VECTOR:=C;
DISRAM;
DHG:=DRAMA;
DTDG:=DDG+DT(.D.);
DTHG:=DHG+DT(.H.);
IF DTDG>DTHG THEN
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.7,I.):=VT(.8,I.);
   MATMEN(.7,I.):=VM(.8,I.);
  END;
  MATDIS(.7,ETAPA.):=DHG;
  MATMEN(.7,ETAPA.):=RAMA(.C,1.);
 END
ELSE
 BEGIN
  FOR I:=1 TO ETAPA-1 DO
  BEGIN
   MATDIS(.7,I.):=VT(.4,I.);
   MATMEN(.7,I.):=VM(.4,I.);
  END;
```

```
      MATDIS(.7,ETAPA.):=DDG;
      MATMEN(.7,ETAPA.):=RAMA(.G,1.);
     END;
    VECTOR:=D;
    DISRAM;
    DHH:=DRAMA;
    DTDH:=DDH+DT(.D.);
    DTHH:=DHH+DT(.H.);
    IF DTDH>DTHH THEN
     BEGIN
      FOR I:=1 TO ETAPA-1 DO
      BEGIN
       MATDIS(.8,I.):=VT(.8,I.);
       MATMEN(.8,I.):=VM(.8,I.);
      END;
      MATDIS(.8,ETAPA.):=DHH;
      MATMEN(.8,ETAPA.):=RAMA(.D,1.);
     END
    ELSE
     BEGIN
      FOR I:=1 TO ETAPA-1 DO
      BEGIN
       MATDIS(.8,I.):=VT(.4,I.);
       MATMEN(.8,I.):=VM(.4,I.);
      END;
      MATDIS(.8,ETAPA.):=DDH;
      MATMEN(.8,ETAPA.):=RAMA(.H,1.);
     END;
 END;
{******FIN DE PROCEDIMIENTO ETA4 *********}

PROCEDURE DISTRAY;
{***INICIA PROCEDIMIENTO DISTRAY *****
****PARA CALCULAR LA DISTANCIA  DE CADA TRAYECTORIA***
****GENERADA HASTA ESTA ETAPA Y EL MENSAJE CORRES-****
****PONDIENTE A CADA SOBREVIVIENTE HASTA LA ETAPA ****
****CORRIENTE                                      ****}
VAR RENG,COL,SUM:INTEGER;
 BEGIN
  FOR RENG:=1 TO 8 DO
   BEGIN
    SUM:=0;
    FOR COL:=1 TO ETAPA DO
     BEGIN
     SUM:=SUM+MATDIS(.RENG,COL.);
     VT(.RENG,COL.):=MATDIS(.RENG,COL.);
     VM(.RENG,COL.):=MATMEN(.RENG,COL.);
     END;
     DT(.RENG.):=SUM;
   END;
 END;
{***** FIN DE PROCEDIMIENTO DISTRAY ******}
```

```
PROCEDURE VITERBI;
  {************INICIA PROCEDIMIENTO VITERBI***************
   ***EJECUTA LA DESCODIFICACIONDE MAXIMA VEROSIMILITUD****
   ***SOBRE CUARENTA TRIPLES DEL MENSAJE RECIBIDO      ****}
VAR ET,INIX,INIY: INTEGER;
BEGIN
{***INICIALIZA CON LOS VALORES DEL TRIPLE ASOCIADO*****
****A CADA RAMA DEL ENREJADO EN LAS ETAPAS 1, 2, 3******}
RAMA(.A,1.):=0;
RAMA(.A,2.):=0;
RAMA(.A,3.):=0;
RAMA(.B,1.):=1;
RAMA(.B,2.):=1;
RAMA(.B,3.):=1;
RAMA(.C,1.):=0;
RAMA(.C,2.):=1;
RAMA(.C,3.):=0;
RAMA(.D,1.):=1;
RAMA(.D,2.):=0;
RAMA(.D,3.):=1;
RAMA(.E,1.):=0;
RAMA(.E,2.):=1;
RAMA(.E,3.):=1;
RAMA(.F,1.):=1;
RAMA(.F,2.):=0;
RAMA(.F,3.):=0;
RAMA(.G,1.):=0;
RAMA(.G,2.):=0;
RAMA(.G,3.):=1;
RAMA(.H,1.):=1;
RAMA(.H,2.):=1;
RAMA(.H,3.):=0;
{***INICIALIZA LAS MATRICES DE DISTANCIAS DE TRAYEC-*****
****TORIA, MENSAJES SUPUESTOS Y SUS CORRESPONDIENTES*****
****SOPORTES VT Y VM                                *****}
FOR INIX:=1 TO 8 DO
 BEGIN
  FOR INIY:=1 TO 43 DO
   BEGIN
   MATDIS(.INIX,INIY.):=0;
   MATMEN(.INIX,INIY.):=9;
   VT(.INIX,INIY.):=0;
   VM(.INIX,INIY.):=9;
   END;
 END;
FOR ET:=1 TO 43 DO
 BEGIN
  ETAPA:=ET;
  CASE ETAPA OF
   1: BEGIN
       ETA1;
       MAT1;
      END;
```

```
   2: BEGIN
      ETA1;
      ETA2;
      MAT1;
      MAT2;
      END;
   3: BEGIN
      ETA1;
      ETA2;
      ETA3;
      MAT1;
      MAT2;
      MAT3;
      DISTRAY;
      END;
 4..43: BEGIN
         ETA1;
         ETA2;
         ETA3;
         ETA4;
         DISTRAY;
        END;
   OTHERWISE WRITELN('ERROR EN PROGRAMA')
   END;
 END;
 MENSAJE;
END;
{****   PROGRAMA    PRINCIPAL ************************}
BEGIN
 REWRITE(ARCHIVO);
 PMC:=0.35;
 RANDOMIZE;
 WHILE PMC<0.40 DO
  BEGIN
   GENPOS(PMC,LIM);
   WRITELN(ARCHIVO);
   WRITE(ARCHIVO,'PROBABILIDAD DE ERROR EN EL CANAL: ',PMC);
   WRITELN('PMC=',PMC);
   IF (LIM=1)OR(LIM=119) THEN
    BEGIN
     COMBMAX:=120
    END
   ELSE
    BEGIN
     COMBMAX:=1000;
    END;
   WRITELN('COMBINACIONES=', COMBMAX);
   NUMCOMB:=1;
   VECTPOS(LIM);
{** GENERESE ERGAUS**********************}
     INDX:=1;
     FOR I:=1 TO 47 DO
      BEGIN
```

```
        INDX:=INDX*I;
        FOR J:=0 TO 2 DO
          BEGIN
           ERGAUS(.I,J+1.):=POINT(.NUMCOMB,J+3*INDX-2.);
          END;
        INDX:=1;
       END;
{**** CALCULESE  CODIFICACION-DESCODIF. DEL MENSAJE*}
   CODIFICA;
   VITERBI;
{**** FIN DE DESCODIFICACION CON VITERBI ****}
   NUMCOMB:=2;
   NUMCOMP:=1;
   WHILE NUMCOMB<COMBMAX DO
    BEGIN
{*** GENERA VECTOR ALEATORIO NO REPETIDO **}
     REPEAT
      GENERA:=TRUE;
      REPET:=FALSE;
      VECTPOS(LIM);
      WHILE NUMCOMP<>NUMCOMB DO
       BEGIN
        IDEN:=0;
        FOR NUMCOL:=1 TO 141 DO
         BEGIN
      IF POINT(.NUMCOMB,NUMCOL.)=POINT(.NUMCOMP,NUMCOL.)THEN
           IDEN:=IDEN+1;
         END;
        IF IDEN=141 THEN
         BEGIN
          IDEN2:=NUMCOMP;
          NUMCOMP:=NUMCOMB;
          REPET:=TRUE;
         END
        ELSE
         BEGIN
          NUMCOMP:=NUMCOMP+1;
         END;
       END;
      IF REPET=FALSE THEN
       BEGIN
       GENERA:=FALSE;
       END
      ELSE
       BEGIN
       WRITELN('VECTOR NUMCOMB:',NUMCOMB:4);
       WRITELN('VECTOR NUMCOMP:',IDEN2:4);
       NUMCOMP:=1;
       END;
     UNTIL GENERA=FALSE;
{*** FIN DE GENERA VECTOR ALEATORIO NO REPETIDO  ****}
{** GENERA ERGAUS **********************}
     INDX:=1;
```

```
      FOR I:=1 TO 47 DO
       BEGIN
        INDX:=INDX*I;
        FOR J:=0 TO 2 DO
          BEGIN
           ERGAUS(.I,J+1.):=POINT(.NUMCOMB,J+3*INDX-2.);
          END;
        INDX:=1;
       END;
{**** CALCULESE CODIFICACION Y DESCODIFICACION ****}
      CODIFICA;
      VITERBI;
{**** FIN DE CODIFICACION-DESCODIFICACION DEL MENS.**}
      NUMCOMB:=NUMCOMB+1;
      NUMCOMP:=1;
     END;
   PMC:=PMC+0.02;
  END;
CLOSE(ARCHIVO);
END.
{****    FIN DE PROGRAMA     PRINCIPAL ******************}
```

Resultados de la simulación de ambos descodificadores

Parámetros utilizados

El programa de simulación deposita en un archivo de salida los valores de los vectores que a continuación se explican:

E : Vector de 47 bits que representa el mensaje generado por la fuente de información que por simplicidad siempre tendrá un valor igual a cero. De los 47 bits , los 4 primeros son de inicializacion del codificador , y los siguientes 40 son del mensaje a transmitir y los ultimos 3 son para limpiar el codificador.

VE: Vector organizado en 47 triples que es la version codificada del vector E.

ERGAUS: Vector organizado en 47 triples que representa el patron de errores introducido al mensaje codificado VE en el canal digital, cada elemento de este vector cuyo valor sea 1, sera un error introducido en VE en la misma posicion que ocupa este 1 en ERGAUS.

VR: Vector de 47 triples que representa el mensaje codificado con el ruido introducido en el canal y que recibe el decodificador.

VS: Vector de 47 triples que contiene los triples supuestos generados a partir del primer bit de cada triple de VR, el vector VS es comparado con VR para obtener el patrón de diferencias en las posiciones pares e impares para decidir sobre el valor estimado del mensaje original.

R: Vector de 47 bits que es el mensaje decodificado a partir del vector VR, es en sí una estimación del mensaje original E.

De esta información generada por el programa se generan los siguientes parámetros que nos son útiles para medir el desempeño del descodificador:

-**DIFIN**.- Es el número de bits diferentes entre el vector E y el vector R.

-**TOTER**.- Es el numero de errores (1's) introducidos al vector mensaje VE por el canal digital.

-**ECODEC**.- Es el porcentaje de errores no corregidos por el algoritmo y se calcula como sigue:

$$ECODEC = (DIFIN / 40) \times 100$$

De esta fórmula se deduce que si ECODEC vale 0% significa que el mensaje se ha decodificado sin errores, si ECODEC vale 100% el algoritmo corrige cero errores para cuando PMC = 1, es decir, si el valor de ECODEC crece, menor es la capacidad de corrección del algoritmo.

-**ECANAL**.-Es el porcentaje de errores introducidos en el mensaje codificado VR por el canal digital o sea:

$$ECANAL = (TOTER / 120) \times 100$$

Nótese que ECODEC se evalúa respecto a 40 bits del mensaje y no respecto a los 47 bits de los vectores E o R, y que ECANAL se evalúa sobre los 120 bits que forman los 40 triples del mensaje codificado y no los 141 bits que constituyen al vector VE.

La figura 3.10 muestra un ejemplo de la información depositada por el programa en el archivo de salida para PMC=0.01 mostrando el valor de los vectores y parámetros anteriormente descritos.

```
PROBABILIDAD DE ERROR EN EL CANAL:  1.000000000000E-02
VE =
[illegible]
ERDAUS=
[illegible]
VR=
[illegible]

R= [illegible]
E= [illegible]

TOTER=            1 DIFIN=          [illegible]
ECANAL=  0.3333333333333E-03 ECODEC= 0.0000000000000E+00
VE =
[illegible]
ERDAUS=
[illegible]
VR=
[illegible]

R= [illegible]
E= [illegible]

TOTER=            1 DIFIN=          [illegible]
ECANAL=  0.3333333333333E-03 ECODEC= 0.0000000000000E+00
VE =
[illegible]
ERDAUS=
[illegible]
VR=
[illegible]

R= [illegible]
E= [illegible]

TOTER=            1 DIFIN=          [illegible]
ECANAL=  0.3333333333333E-03 ECODEC= 0.0000000000000E+00
VE =
[illegible]
ERDAUS=
[illegible]
VR=
[illegible]

R= [illegible]
E= [illegible]

TOTER=            1 DIFIN=          [illegible]
ECANAL=  0.3333333333333E-03 ECODEC= 0.0000000000000E+00
VE =
[illegible]
ERDAUS=
[illegible]
VR=
[illegible]

R= [illegible]
E= [illegible]

TOTER=            1 DIFIN=          [illegible]
ECANAL=  0.3333333333333E-03 ECODEC= 0.0000000000000E+00
VE =
[illegible]
ERDAUS=
[illegible]
VR=
[illegible]
```

Figura 3.10 Muestra del archivo generado por el simulador.

El programa se ejecutó para el rango PMC de 0.01 a 0.57 simultáneamente con el programa que decodifica en base a el algoritmo de Viterbi para hacer una compararción de ambos algoritmos (de Lepe y de Viterbi).

Los resultados fueron vaciados en la tabla de la fig. 3.11.

En la primer columna se tiene el valor de PMC (en %).

En la segunda columna tenemos el valor de ECODEC.

En la tercera columna, la frecuencia con que se desempeñó el algoritmo para el correspondiente valor de ECODEC respecto al total de veces que descodificó un mensaje con el nivel de error de PMC indicado (se descodificó 119 veces para PMC=0.01 y 999 veces para PMC>0.01).

La cuarta columna indica lo que se define como razón de descodificación (DECRAZO) y es la razón de errores sin corregir (ECODEC) a los errores introducidos en el mensaje codificado (TOTER), este parámetro nos permite observar si el algoritmo disminuyó el porcentaje de errores introducidos en el mensaje descodificado R respecto del mensaje recibido VR. Por ejemplo, en la figura 3.11 se observa que para PMC=13% y ECODEC=5% la razón de descodificación es de 2 errores introducidos en el mensaje descodificado (de 40 bits) respecto a 16 errores introducidos en el mensaje recibido (de 120 bits).

Algo importante de mencionar es que ni el algoritmo propuesto ni el algoritmo de Viterbi llegan a tener una DECRAZO>1.

Tabla comparativa de ambos descodificadores

En la figura 3.11 se observa a primera vista que el algoritmo propuesto disminuye su capacidad de correccion en cuanto aumenta el nivel de ruido en el canal (PMC) dado que la frecuencia para valores bajos de ECODEC para el algoritmo propuesto disminuye mas rapido que la frecuencia para el algoritmo de Viterbi.

Por ejemplo:

* Para PMC=11% y ECODEC=0% (mensaje descodificado sin error) la frecuencia para el algoritmo propuesto es 249 y para Viterbi la frecuencia es de 959.

* Para PMC=13% y el mismo valor de ECODEC la frecuencia para el algoritmo propuesto es de 114 y para Viterbi la frecuencia es de 884.

Esto es, la capacidad de corrección de Viterbi es casi 8 veces mayor al algoritmo propuesto.

Descodificador Propuesto.				Descodificador deViterbi.			
%PMC	ECODEC(%)	FREC	DECRAZO	%PMC	ECODEC(%)	FREC	DECRAZO
1.00	0.0	119/119	0/1	1.0	0.0	119/119	0/1
3.00	0.0	923/999	0/4	3.00	0.0	999/999	0/4
3.00	2.5	65	1/4				
3.00	5.0	8	2/4				
3.00	7.5	2	3/4				
3.00	10.0	1	4/4				
5.00	0.0	812/999	0/6	5.00	0.0	998/999	0/6
5.00	2.5	126	1/6	5.00	5.0	1	2/6
5.00	5.0	31	2/6				
5.00	7.5	15	3/6				
5.00	10.0	11	4/6				
5.00	12.5	1	5/6				
5.00	15.0	3	6/6				
7.00	0.0	646/999	0/8	7.00	0.0	998/999	0/8
7.00	2.5	182	1/8	7.00	5.0	1	2/8
7.00	5.0	90	2/8				
7.00	7.5	48	3/8				
7.00	10.0	16	4/8				
7.00	12.5	14	5/8				
7.00	15.0	1	6/8				
7.00	17.5	1	7/8				
7.00	20.0	1	8/8				
9.00	0.0	397/999	0/11	9.00	0.0	987/999	0/11
9.00	2.5	184	1/11	9.00	2.5	6	1/11
9.00	5.0	166	2/11	9.00	5.0	5	2/11
9.00	7.5	116	3/11	9.00	7.5	1	3/11
9.00	10.0	69	4/11				
9.00	12.5	39	5/11				
9.00	15.0	15	6/11				
9.00	17.5	6	7/11				
9.00	20.0	2	8/11				
9.00	22.5	3	9/11				
9.00	25.0	1	10/11				
11.0	0.0	249/999	0/13	11.0	0.0	959/999	0/13
11.0	2.5	187	1/13	11.0	2.5	12	1/13
11.0	5.0	141	2/13	11.0	5.0	16	2/13
11.0	7.5	142	3/13	11.0	7.5	7	3/13
11.0	10.0	118	4/13	11.0	10.0	3	4/13
11.0	12.5	67	5/13				
11.0	15.0	40	6/13				
11.0	17.5	25	7/13				
11.0	20.0	15	8/13				
11.0	22.5	10	9/13				
11.0	25.0	1	10/13				
11.0	27.5	2	11/13				
11.0	30.0	0	12/13				
11.0	32.5	2	13/13				
13.0	0.0	114/999	0/16	13.0	0.0	884/999	0/16
13.0	2.5	106	1/16	13.0	2.5	36	1/16
13.0	5.0	117	2/16	13.0	5.0	39	2/16
13.0	7.5	152	3/16	13.0	7.5	23	3/16
13.0	10.0	139	4/16	13.0	10.0	11	4/16
13.0	12.5	108	5/16	13.0	12.5	5	5/16
13.0	15.0	86	6/16	13.0	15.0	1	6/16
13.0	17.5	61	7/16				
13.0	20.0	51	8/16				
13.0	22.5	23	9/16				
13.0	25.0	14	10/16				
13.0	27.5	9	11/16				
13.0	30.0	7	12/16				
13.0	32.5	2	13/16				
13.0	35.0	1	14/16				
13.0	37.5	0	15/16				
13.0	40.0	1	16/16				

%PMC	ECODEC(%)	FREC	DECRAZO	%PMC	ECODEC(%)	FREC	DECRAZO
15.0	0.0	47/999	0/18	15.0	0.0	784/999	0/18
15.0	2.5	63	1/18	15.0	2.5	60	1/18
15.0	5.0	96	2/18	15.0	5.0	67	2/18
15.0	7.5	133	3/18	15.0	7.5	45	3/18
15.0	10.0	133	4/18	15.0	10.0	23	4/18
15.0	12.5	122	5/18	15.0	12.5	11	5/18
15.0	15.0	120	6/18	15.0	15.0	5	6/18
15.0	17.5	83	7/18	15.0	17.5	1	7/18
15.0	20.0	71	8/18	15.0	20.0	3	8/18
15.0	22.5	47					
15.0	25.0	35					
15.0	27.5	25					
15.0	30.0	11					
15.0	32.5	6					
15.0	35.0	4					
15.0	37.5	2					
15.0	40.0	1					
17.0	0.0	20/999	0/20	17.0	0.0	631/999	0/20
17.0	2.5	39	1/20	17.0	2.5	78	1/20
17.0	5.0	72	2/20	17.0	5.0	118	2/20
17.0	7.5	93	3/20	17.0	7.5	66	3/20
17.0	10.0	106	4/20	17.0	10.0	42	4/20
17.0	12.5	112	5/20	17.0	12.5	26	5/20
17.0	15.0	114	6/20	17.0	15.0	16	6/20
17.0	17.5	112	7/20	17.0	17.5	11	7/20
17.0	20.0	99	8/20	17.0	20.0	5	8/20
17.0	22.5	75	9/20	17.0	22.5	3	9/20
17.0	25.0	48	10/20	17.0	25.0	3	10/20
17.0	27.5	37	11/20				
17.0	30.0	34	12/20				
17.0	32.5	15	13/20				
17.0	35.0	7	14/20				
17.0	37.5	9	15/20				
17.0	40.0	5	16/20				
17.0	42.5	2	17/20				
19.0	0.0	2/999	0/23	19.0	0.0	315/999	0/23
19.0	2.5	12	1/23	19.0	2.5	131	1/23
19.0	5.0	21	2/23	19.0	5.0	144	2/23
19.0	7.5	46	3/23	19.0	7.5	146	3/23
19.0	10.0	71	4/23	19.0	10.0	93	4/23
19.0	12.5	75	5/23	19.0	12.5	68	5/23
19.0	15.0	105	6/23	19.0	15.0	33	6/23
19.0	17.5	129	7/23	19.0	17.5	30	7/23
19.0	20.0	110	8/23	19.0	20.0	21	8/23
19.0	22.5	115	9/23	19.0	22.5	11	9/23
19.0	25.0	75	10/23	19.0	25.0	6	10/23
19.0	27.5	74	11/23	19.0	27.5	5	11/23
19.0	30.0	63	12/23	19.0	30.0	1	12/23
19.0	32.5	39	13/23				
19.0	35.0	29	14/23				
19.0	37.5	11	15/23				
19.0	40.0	9	16/23				
19.0	42.5	9	17/23				
19.0	45.0	1	18/23				
19.0	47.5	2	19/23				
19.0	50.0	1	20/23				
21.0	0.0	0/999	0/25	21.0	0.0	165/999	0/25
21.0	2.5	1	1/25	21.0	2.5	87	1/25
21.0	5.0	18	2/25	21.0	5.0	161	2/25
21.0	7.5	18	3/25	21.0	7.5	140	3/25
21.0	10.0	45	4/25	21.0	10.0	136	4/25
21.0	12.5	65	5/25	21.0	12.5	84	5/25
21.0	15.0	87	6/25	21.0	15.0	78	6/25
21.0	17.5	91	7/25	21.0	17.5	57	7/25
21.0	20.0	103	8/25	21.0	20.0	40	8/25
21.0	22.5	99	9/25	21.0	22.5	22	9/25
21.0	25.0	116	10/25	21.0	25.0	15	10/25
21.0	27.5	99	11/25	21.0	27.5	10	11/25
21.0	30.0	94	12/25	21.0	30.0	0	12/25

%PMC	ECODEC(%)	FREC	DECRAZO
21.0	32.5	64	13/25
21.0	35.0	50	14/25
21.0	37.5	22	15/25
21.0	40.0	11	16/25
21.0	42.5	8	17/25
21.0	45.0	0	18/25
21.0	47.5	1	19/25

%PMC	ECODEC(%)	FREC	DECRAZO
21.0	32.5	1	13/25
21.0	35.0	2	14/25
21.0	37.5	1	15/25

23.0	0.0	0/999	0/28
23.0	2.5	0	1/28
23.0	5.0	0	2/28
23.0	7.5	8	3/28
23.0	10.0	13	4/28
23.0	12.5	37	5/28
23.0	15.0	55	6/28
23.0	17.5	93	7/28
23.0	20.0	106	8/28
23.0	22.5	116	9/28
23.0	25.0	113	10/28
23.0	27.5	106	11/28
23.0	30.0	95	12/28
23.0	32.5	92	13/28
23.0	35.0	59	14/28
23.0	37.5	50	15/28
23.0	40.0	19	16/28
23.0	42.5	20	
23.0	45.0	10	
23.0	47.5	5	
23.0	50.0	2	

23.0	0.0	22/999	0/28
23.0	2.5	58	1/28
23.0	5.0	110	2/28
23.0	7.5	112	3/28
23.0	10.0	121	4/28
23.0	12.5	134	5/28
23.0	15.0	131	6/28
23.0	17.5	96	7/28
23.0	20.0	68	8/28
23.0	22.5	46	9/28
23.0	25.0	25	10/28
23.0	27.5	34	11/28
23.0	30.0	14	12/28
23.0	32.5	14	13/28
23.0	35.0	10	14/28
23.0	37.5	1	15/28
23.0	40.0	2	16/28

25.0	0.0	0/999	0/30
25.0	2.5	0	1/30
25.0	5.0	3	2/30
25.0	7.5	2	3/30
25.0	10.0	8	4/30
25.0	12.5	17	5/30
25.0	15.0	31	6/30
25.0	17.5	49	7/30
25.0	20.0	74	8/30
25.0	22.5	69	9/30
25.0	25.0	127	10/30
25.0	27.5	146	11/30
25.0	30.0	115	12/30
25.0	32.5	109	13/30
25.0	35.0	78	14/30
25.0	37.5	58	15/30
25.0	40.0	51	16/30
25.0	42.5	37	17/30
25.0	45.0	15	18/30
25.0	47.5	6	19/30
25.0	50.0	2	20/30
25.0	52.2	2	21/30

25.0	0.0	5/999	0/30
25.0	2.5	32	1/30
25.0	5.0	55	2/30
25.0	7.5	94	3/30
25.0	10.0	101	4/30
25.0	12.5	126	5/30
25.0	15.0	119	6/30
25.0	17.5	114	7/30
25.0	20.0	101	8/30
25.0	22.5	62	9/30
25.0	25.0	61	10/30
25.0	27.5	48	11/30
25.0	30.0	29	12/30
25.0	32.5	20	13/30
25.0	35.0	14	14/30
25.0	37.5	5	15/30
25.0	40.0	5	16/30
25.0	42.5	4	17/30
25.0	45.0	3	18/30
25.0	47.5	1	19/30

27.0	0.0	0/999	0/32
27.0	2.5	0	1/32
27.0	5.0	0	2/32
27.0	7.5	1	3/32
27.0	10.0	7	4/32
27.0	12.5	9	5/32
27.0	15.0	26	6/32
27.0	17.5	40	7/32
27.0	20.0	62	8/32
27.0	22.5	96	9/32
27.0	25.0	94	10/32
27.0	27.5	112	11/32
27.0	30.0	128	12/32
27.0	32.5	111	13/32
27.0	35.0	95	14/32
27.0	37.5	79	15/32
27.0	40.0	57	16/32
27.0	42.5	38	17/32
27.0	45.0	28	18/32

27.0	0.0	0/999	0/32
27.0	2.5	10	1/32
27.0	5.0	22	2/32
27.0	7.5	41	3/32
27.0	10.0	73	4/32
27.0	12.5	99	5/32
27.0	15.0	126	6/32
27.0	17.5	115	7/32
27.0	20.0	120	8/32
27.0	22.5	95	9/32
27.0	25.0	88	10/32
27.0	27.5	73	11/32
27.0	30.0	59	12/32
27.0	32.5	28	13/32
27.0	35.0	19	14/32
27.0	37.5	16	15/32
27.0	40.0	11	16/32
27.0	42.5	2	17/32
27.0	45.0	0	18/32

%PMC	ECODEC(%)	FREC	DECRAZO
27.0	47.5	8	19/32
27.0	50.0	6	20/32
27.0	52.5	2	21/32
29.0	0.0	0/999	0/35
29.0	2.5	0	1/35
29.0	5.0	0	2/35
29.0	7.5	0	3/35
29.0	10.0	2	4/35
29.0	12.5	3	5/35
29.0	15.0	8	6/35
29.0	17.5	14	7/35
29.0	20.0	32	8/35
29.0	22.5	54	9/35
29.0	25.0	86	10/35
29.0	27.5	109	11/35
29.0	30.0	118	12/35
29.0	32.5	114	13/35
29.0	35.0	126	14/35
29.0	37.5	111	15/35
29.0	40.0	79	16/35
29.0	42.5	54	17/35
29.0	45.0	49	18/35
29.0	47.5	27	19/35
29.0	50.0	5	20/35
29.0	52.5	4	21/35
29.0	55.0	3	22/35
29.0	57.5	1	23/35

%PMC	ECODEC(%)	FREC	DECRAZO
27.0	47.5	2	19/32
29.0	0.0	0/999	0/35
29.0	2.5	0	1/35
29.0	5.0	2	2/35
29.0	7.5	16	3/35
29.0	10.0	28	4/35
29.0	12.5	52	5/35
29.0	15.0	71	6/35
29.0	17.5	96	7/35
29.0	20.0	120	8/35
29.0	22.5	122	9/35
29.0	25.0	111	10/35
29.0	27.5	109	11/35
29.0	30.0	78	12/35
29.0	32.5	59	13/35
29.0	35.0	42	14/35
29.0	37.5	43	15/35
29.0	40.0	19	16/35
29.0	42.5	14	17/35
29.0	45.0	5	18/35
29.0	47.5	4	19/35
29.0	50.0	3	20/35
29.0	52.5	2	21/35

Figura 3.11 Tabla comparativa de Frecuencias.

Gráficas de los datos obtenidos con la simulación.

A partir de los datos que se obtuvieron con la simulación de ambos descodificadores es posible ahora obtener las gráficas en donde se comparan las principales características de funcionalidad de ambos descodificadores.

Las tablas mostradas en la figura 3.11 se han sintetizado aún más tomando la suma de los valores de frecuencia cuyo valor de ECODEC es menor que el valor de PMC, como la parte corregida (en %) del total de mensajes descodificados, así para PMC=0.13, la suma de las frecuencias cuyo ECODEC<0.13 es 736 para el algoritmo propuesto; por tanto el porcentaje de mensajes corregidos es:

$$(736/999)x100 = 73.67\%$$

Tomando este valor como la capacidad de corrección del algoritmo se genero la gráfica de la figura 3.12.

En ésta gràfica se indican los valores de PMC en el rango en que se corrió el programa de simulación contra el % de mensajes corregidos. En esta figura se hace evidente que la capacidad de corrección del algoritmo de Viterbi tiende a permanecer constante cuando se incrementa el ruido del canal en tanto que el algoritmo propuesto disminuye notablemente su capacidad de corrección conforme aumenta el ruido del canal, permaneciendo además siempre por debajo de la capacidad del algoritmo de Viterbi, lo cual comprueba su calidad de algoritmo subóptimo.

Otra comparación gráfica se indica en la figuras 3.13(a), 3.13(b),, para el algoritmo propuesto o de Lepe y 3.13(c) y 3.13(d)para el algoritmo de Viterbi.

Aquí se indica en un esquema tridimensional los valores de PMC, ECODEC(%) y frecuencia (normalizada) reportados por el programa de simulación hasta un valor de PMC=0.39. En estas figuras se observa como el algoritmo de Lepe distribuye mas las frecuencias sobre los valores de ECODEC hacia la derecha conforme aumenta el ruido del canal (PMC) que el algoritmo de Viterbi, de hecho el algoritmo de Viterbi comienza a distribuir las frecuencias hacia la derecha hasta que PMC=11%.

CAPACIDAD DE CORRECCION DE ERRORES

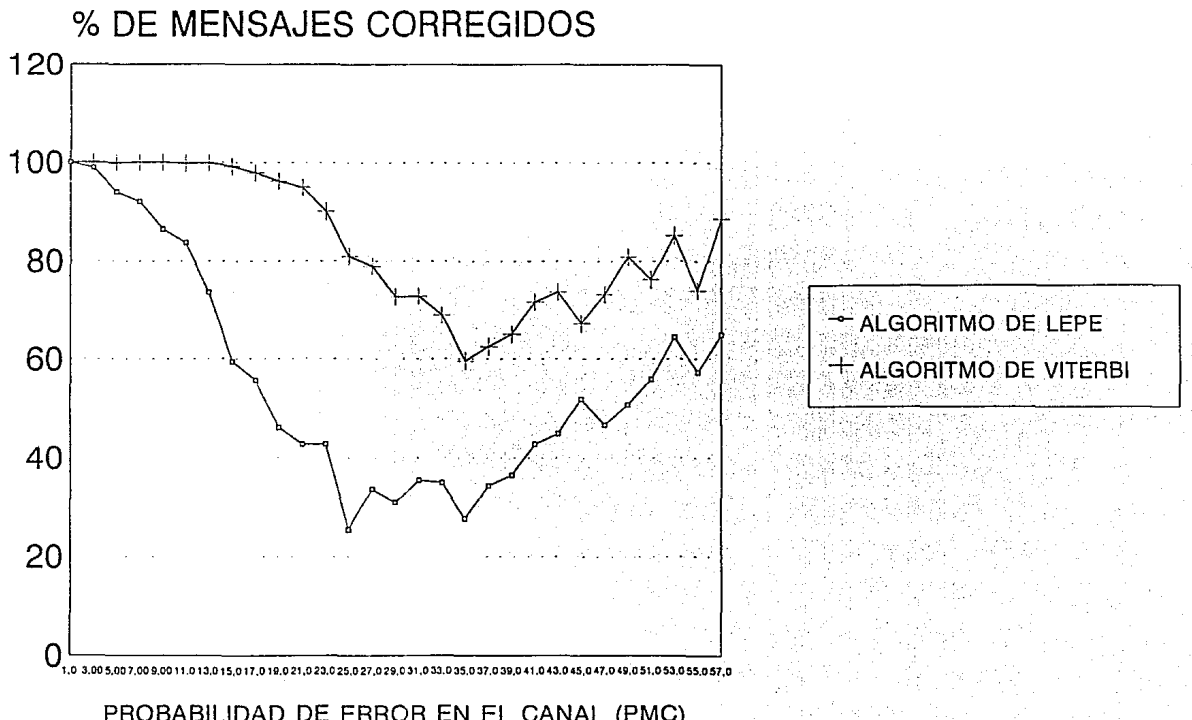


Figura 3.12

FRECUENCIA DE MENSAJES DECODIFICADOS

ALGORITMO DE LEPE

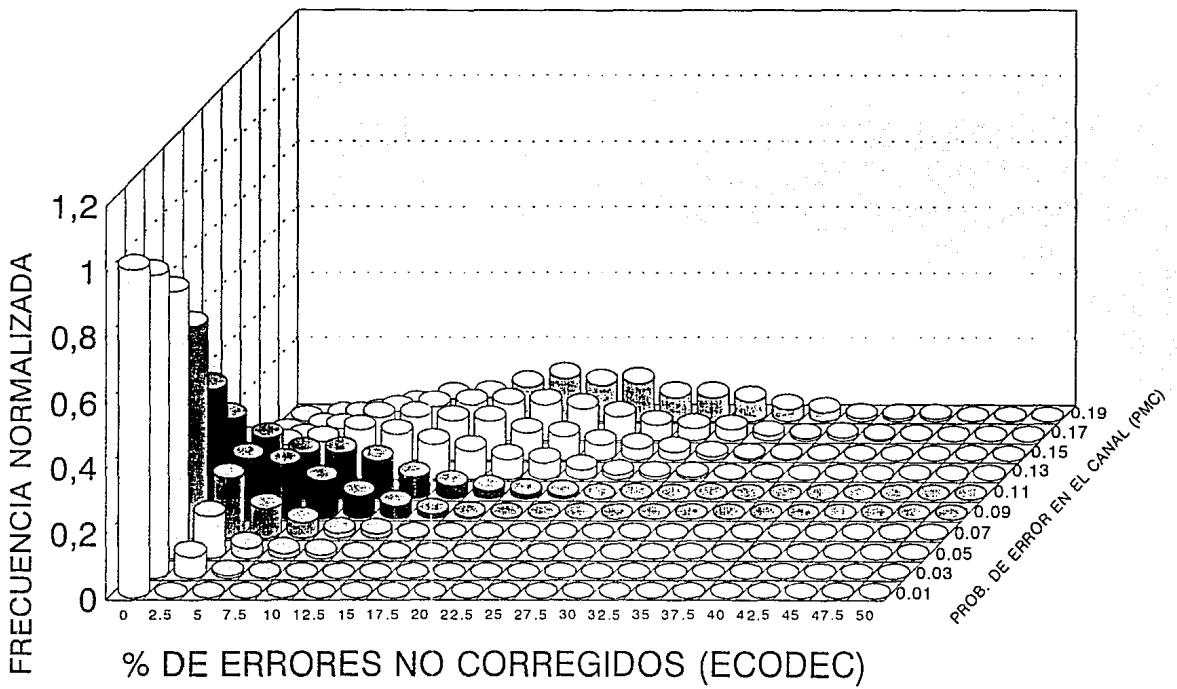


Figura 3.13(a)

FRECUENCIA DE MENSAJES DESCODIFICADOS

ALGORITMO DE LEPE

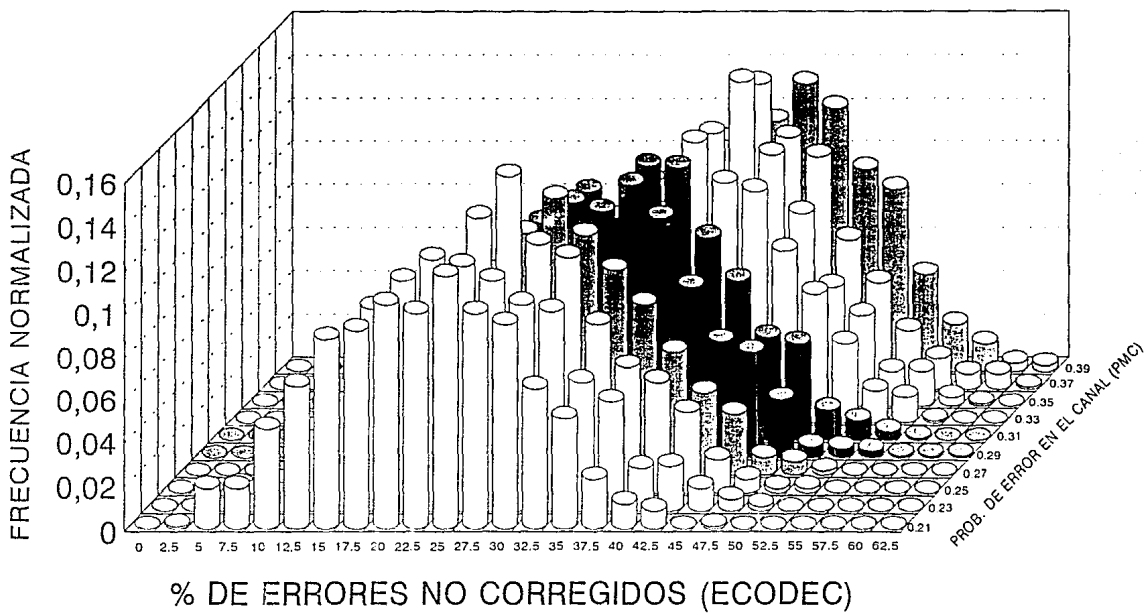


Figura 3.13(b)

FRECUENCIA DE MENSAJES DECODIFICADOS

ALGORITMO DE VITERBI

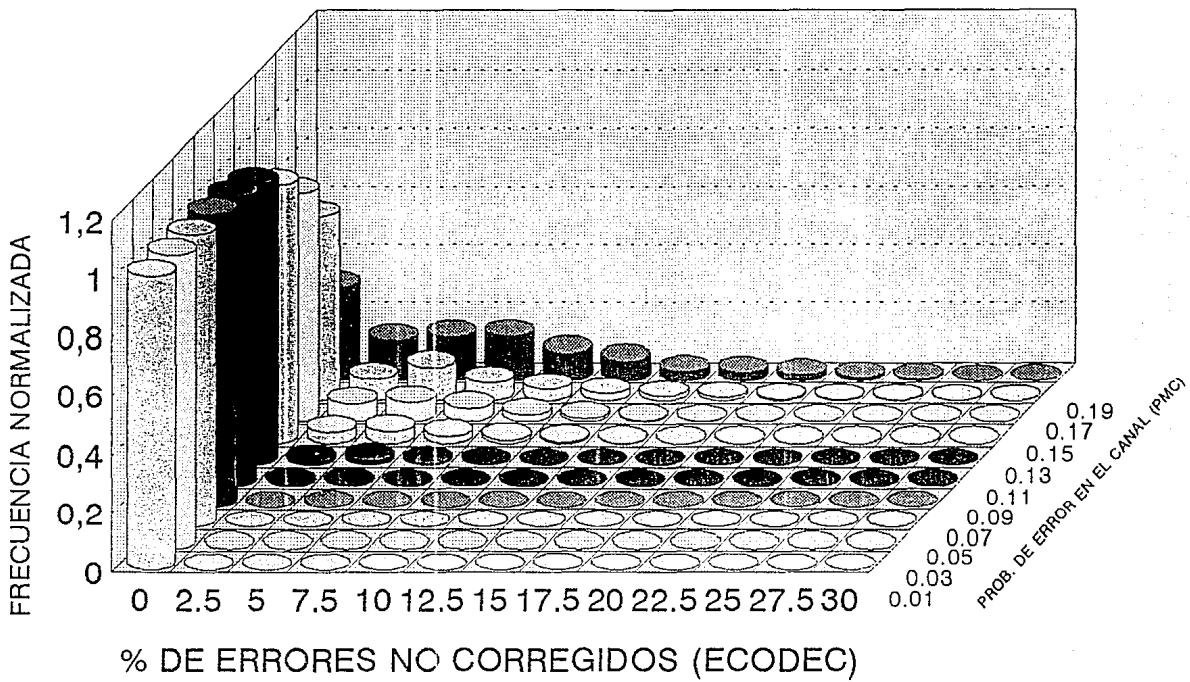


Figura 3.13(c)

FRECUENCIA DE MENSAJES DECODIFICADOS

ALGORITMO DE VITERBI

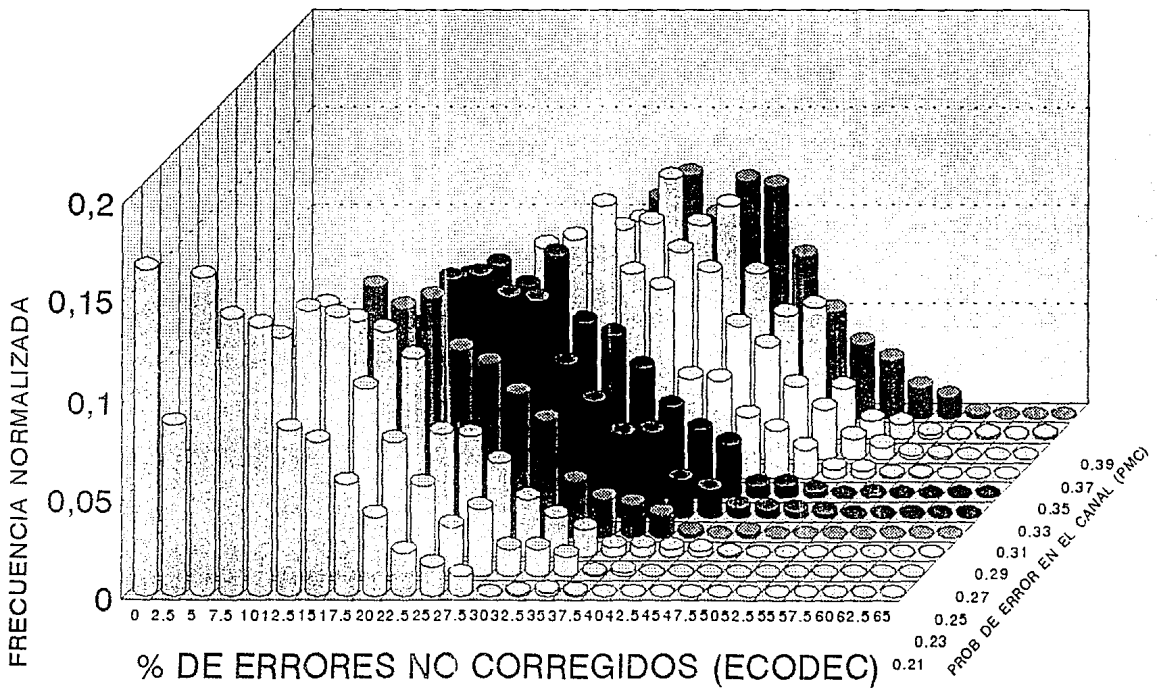


Figura 3.13(d)

Retardo de descodificación

Para medir el retardo de decodificación de ambos algoritmos se cuantificó el tiempo de ejecución de la rutina de descodificación de ambos algoritmos en el programa de simulación con la funcion interna de PASCAL/VS TIME, la cual proporciona el tiempo de ejecucion del programa en microsegundos. El resultado es alentador para el algoritmo propuesto dado que el tiempo mas alto de ejecución para este fué de dos milisegundos aproximadamente en tanto que el tiempo mínimo de ejecución para el algoritmo de Viterbi fué de aproximadamente catorce milisegundos, esto es, siete veces mas lento que el algoritmo propuesto. Esto para PMC>0.15 porque para valores menores de PMC el mayor valor del tiempo de ejecución fué de 1 milisegundo aproximadamente.

Cabe mencionar que los tiempos de ejecución dados como máximos en este párrafo ocurren con una frecuencia aproximada de 1 en 1000 en tanto que los valores minimos dados para viterbi son mucho más frecuente (aproximadamente 700 en 1000).

Por último se menciona que un valor típico para el algoritmo propuesto es de 300 microsegundos. Lo anterior se ilustra gráficamente en la figura 3.14.

TIEMPO DE EJECUCION
ALGORITMO PROPUESTO

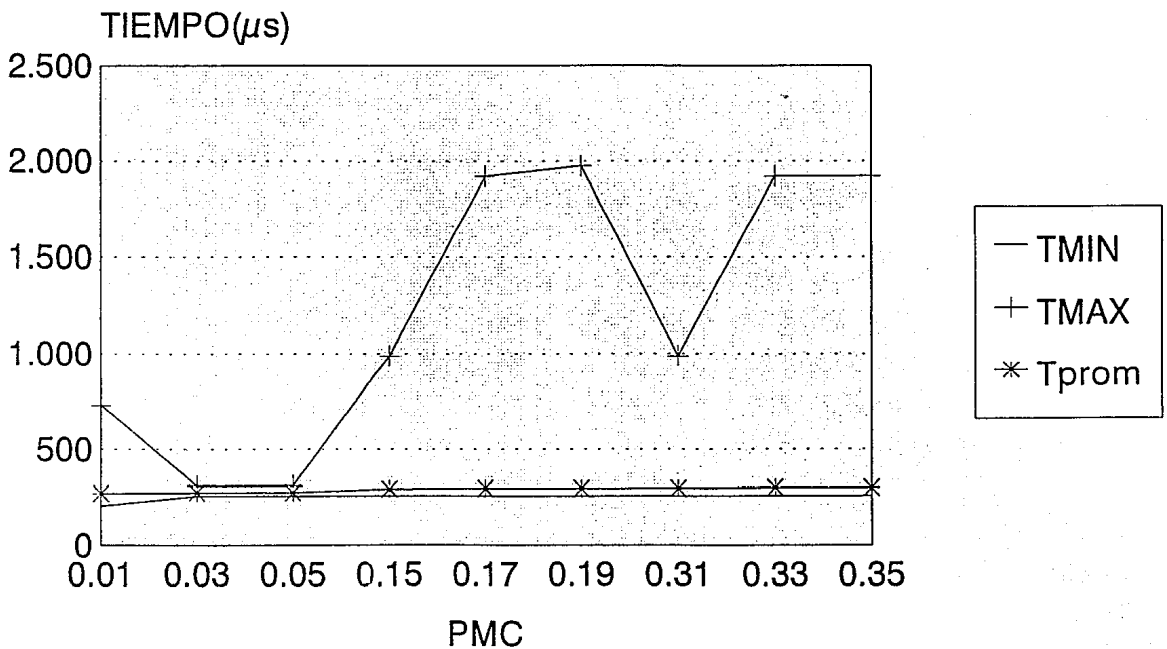


Figura 3.14(a)

TIEMPO DE EJECUCION

ALGORITMO DE VITERBI

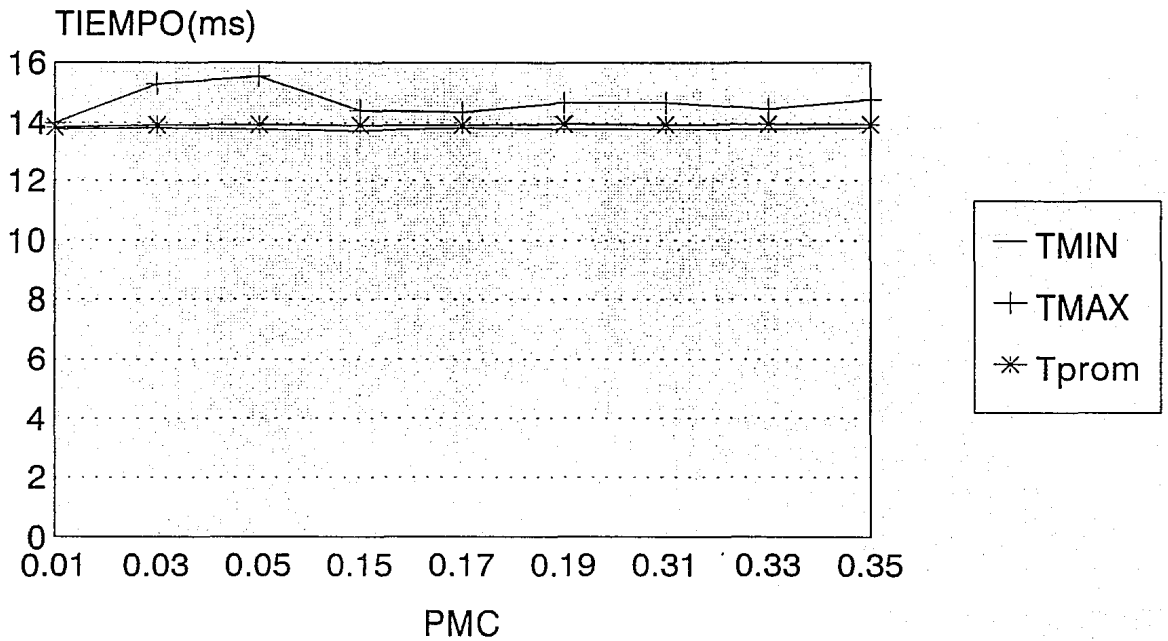


Figura 3.14(b)

REFERENCIAS

[1] - Taub Herbert; Schilling Donald
Principles of Communication Systems
Ed. Mc Graw Hill, 1986

[2] - Clark George C.; Cain J. Bibb
Error Correction Coding for Digital Communications.
Aplication of comm. Theory Series Editor: R. W.Lucky;
Bell Laboratories, 1981

[3] - Haykin Simon
Digital Communications
Ed.John Wiley & sons, 1988

[4] - Wiggert Djimitri
Error-Control Coding and Applications
Artech House, INC. Dedham, Massachusets

[5] - Borras García Hugo E.; Iriarte Balderrama Rafael,
Frontera de la Cruz Bernardo
Apuntes de Probabilidad y Estadística Fac. de
Ingeniería 1985

[6] - Walpole R.H., Myers R. H.
Probabilidad y Estadistica p/Ing.
Ed. Interamericana 2a Ed., México 1986

[7] - Heller A. Jerold; Jacobs Irwin M.
Viterbi Decoding for Satellite and Space Commm.
Revista: IEEE Trans. on Communication Technology
Vol. 19 no.5, oct 1971.

[8] - Lepe-Casillas, Fernando.
Dos Algoritmos para decodificar códigos convolucionales
Sometido a la revista catalana Questió. No publicado.

CAPITULO 4

DESCRIPCION DEL ALGORITMO DE DESCODIFICACION EMPLEADO EN EL SISTEMA DE TELEFONIA DIGITAL 1240

4.1 Introducción

En este capítulo se describe inicialmente el codificador utilizado dentro del Sistema 12, en sus módulos denominados Elementos de Control, y cuya función principal es dar confiabilidad a la información que maneja el microprocesador de cada elemento de control y que se almacena en la memoria asociada al mismo. Es importante proveer de la máxima confiabilidad a la información que se maneja en las centrales telefónicas digitales ya que sus funciones, el control y la calidad del servicio, dependen en mucho de este aspecto. La confiabilidad se vuelve mas crítica dado el hecho de que la operación de estos sistemas se basa en un control distribuido, lo que significa que sus módulos tienen una capacidad de autosuficiencia tal que si llegara a ocurrir algún problema dentro del mismo, éste se aislará y el elemento de control podrá tomar decisiones por sí mismo y determinar que acción a tomar es la más adecuada para dar continuidad a sus respectivas funciones. Se propone como alternativa el poder utilizar el descodificador que se presentó en el capítulo 3, en el sistema 12, aprovechando las características de memoria que éste presenta, como es capacidad, velocidad en lectura y escritura, y los diferentes mecanismos que provee el sistema en cuanto a refresco de memoria y protección contra escritura en ciertos segmentos de la misma. El codificador al que se hace referencia en este capítulo es una solución ya existente para asegurar la confiabilidad en el manejo de información dentro del sistema 12. Por otro lado, se considera factible la implantación del codificador descrito al final de este capítulo, al analizar también las ventajas y desventajas de éste último dentro de los requerimientos de una central digital.

4.2 Arquitectura de la tarjeta del Elemento de Control

La arquitectura básica de la central digital 1240 de ITT se compone de una red simple compuesta de una red digital de conmutación que conecta una gran variedad de módulos, ver fig. 4.1.

Todos estos módulos contienen un equipo físico en común: El elemento de control (CE), que consiste de los siguientes circuitos:

- Interfaz terminal (TERI)
- Microprocesador (TCPB)
- Memoria asociada al microprocesador

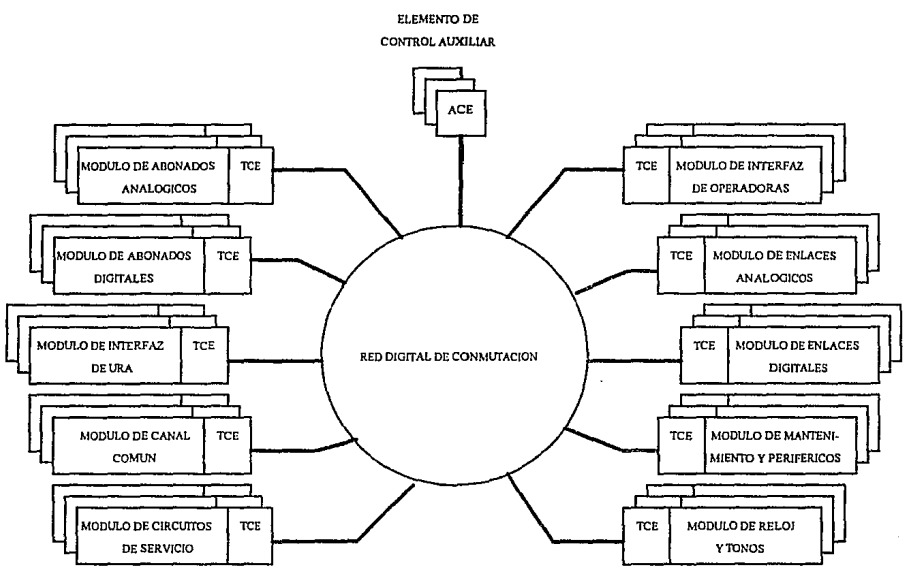


Fig. 4.1 Arquitectura básica de la central 1240 de ITT.

En los módulos o elementos de control terminal (TCE), el TCPB está dedicado a ejecutar los procesos terminales del sistema. Esto significa que el TCPB supervisa los enlaces entre las terminales que ejecutan varias funciones y la red digital de conmutación.

En los elementos de control auxiliar (ACE), el TCPB ejecuta procesos que no tienen un efecto directo sobre los elementos terminales de la central. Los elementos de control auxiliar ACE tienen una jerarquía de control más alta que la de los elementos de control terminal TCE.

Existen varias trayectorias de comunicación posibles entre el elemento de control y su correspondiente terminal, algunos utilizan el bus del TCPB. Por ejemplo: para los módulos de troncales digitales y los módulos de abonados, la terminal se conecta al elemento de control a través del TERI mediante un enlace PCM; para el módulo de servicios, la terminal se conecta a él mediante el TERI, con un enlace PCM y directamente por un bus de alta velocidad (HSB) del microprocesador. El TCPB incluye un microprocesador 8086 (en versiones recientes el microprocesador utilizado es el V30, el cual es completamente compatible con los diseños basados en el 8086, con la ventaja de que es un 40% más rápido que el 8086). Dicho microprocesador tiene acceso a funciones propias de su tarjeta y a funciones externas a la tarjeta.

El diagrama de bloques de la figura 4.2 muestra los bloques internos ligados a trayectorias de datos y funciones internas del TCPB. Los elementos que componen a la tarjeta, además del procesador, son:

- Memoria RAM (con capacidad de 1 Mbyte)
- Memoria ROM (tipo EEPROM de 16 Kbytes)
- Puertos de entrada/salida
- Interfaz serie, etc.

Otras trayectorias de datos y direcciones dan acceso a entradas/salidas, como lo es la interfaz terminal y aplicaciones especiales al puerto dual de la memoria RAM.

El acceso a los diferentes buses depende de las señales seleccionadas por el Descodificador de Direcciones. El Descodificador de Direcciones de la Memoria da acceso siempre que algún dispositivo haga referencia a la memoria. El bus interno (local) multiplexado en tiempo, parte del procesador de 20 bits . Este bus interno se constituye de un bus de direcciones local y un bus de datos interno.

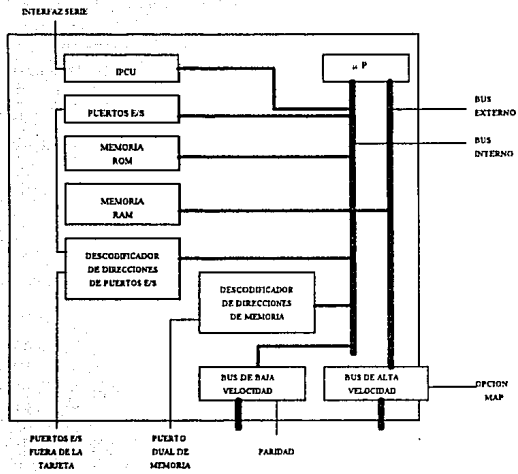


Fig.4.2 Diagrama de bloques del TCPB.[11]

El circuito IPCU (Integrated Processor Control Unit) que se observa como bloque en la figura 4.2, está diseñado para simplificar y optimizar el diseño de un procesador basado en el circuito 70116 (o V30) que incluye a la memoria en la misma tarjeta. Así el IPCU descodifica por sí mismo el bus del procesador multiplexado en tiempo. Este bus del procesador se conecta al IPCU vía el bus interno de datos/direcciones.

Así, de la misma fig. 4.2 se ve que este bus tiene una doble función:

-Proporcionar una conexión de datos hacia la RAM, ROM y entradas/salidas;
-Además, proporcionar una conexión de datos y direcciones al IPCU.

El diagrama físico de la tarjeta TCPB se muestra en la figura 4.3.

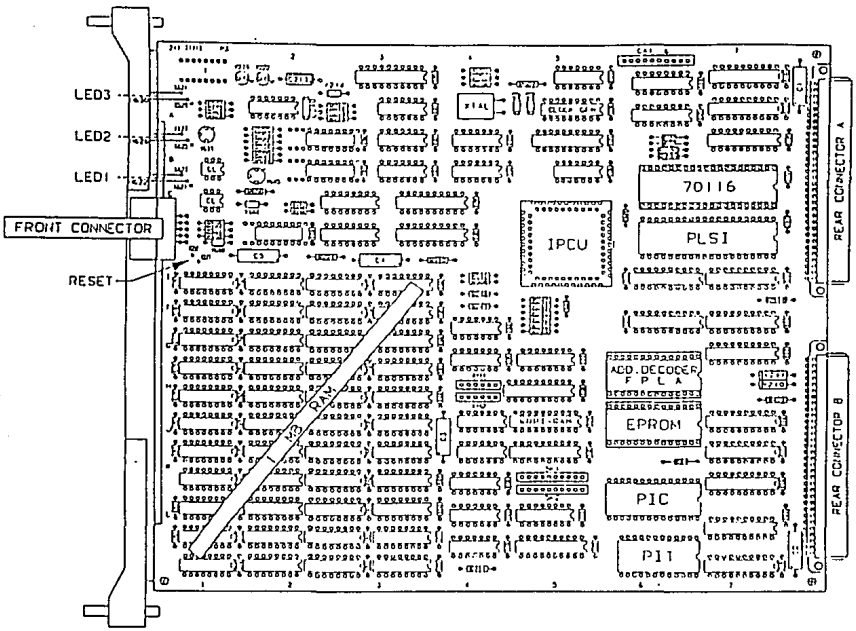


Fig.4.3 Diagrama físico del la tarjeta TCPB.

4.3 Descripción de la memoria asociada al microprocesador

Memoria RAM

Se encuentra alojada en la tarjeta del TCPB. Esta memoria tiene una capacidad de 1 Mbyte, arreglada en dos bancos de 256 mil palabras de 16 bits cada una más el espacio que ocupa el código de corrección de errores ECC para cada palabra de 16 bits. Cada uno de estos bancos están organizados en una palabra de 16 bits de datos más 6 bits de verificación. Físicamente, los circuitos RAM tienen una dimensión de 256000 renglones de un bit cada uno. Entonces, para

tienen una dimensión de 256000 renglones de un bit cada uno. Entonces, para cubrir el rango de direccionamiento de 1 Mbyte se necesitan dos grupos de 22 circuitos RAM, un grupo para cada banco de memoria, lo cual hace un total de 44 circuitos RAM.

Los 20 bits de direccionamiento provistos por el procesador los multiplexa el circuito IPCU hacia los pines de entrada de direcciones de los circuitos RAM.

Básicamente, una palabra se conforma de dos bytes que pueden direccionarse juntos o por separado, esto es, por palabra (dos bytes) o por byte. El bit de direcciones A0 (bit menos significativo) junto con la señal BHE (Byte High Enable) determinan el modo de acceso según la siguiente tabla:

BHE	A0	Tipo de datos a transferir
1	0	PALABRA (sólo direcciones pares)
0	1	BYTE (direcciones impares)
0	0	BYTE (direcciones pares)
0	1	NO VALIDO

El bit A19 del bus de direcciones selecciona el banco de memoria 0 ó 1. Para tener una densidad de empaquetamiento alta, los circuitos de RAM sólo tienen 9 terminales de direcciones y por lo tanto los 18 bits de direcciones restantes (A1 a A18) los multiplexa el IPCU en dos grupos de 9 bits.

En un primer tiempo las direcciones de renglón habilitan a los dispositivos de memoria con la señal RAS (Row Address Strobe), en un segundo paso las direcciones de columna habilitan a los dispositivos de memoria con la señal CAS (Column Address Select).

Se cuenta también con un contador interno de direcciones que cada vez que se inicia el ciclo de memoria con la solicitud de la señal de vitalización, el contador de direcciones de vitalización se aplica a los dispositivos de memoria y al final del ciclo el contador se incrementa en uno. Se requiere una vitalización de memoria cada 15 microsegundos para garantizar que se realizarán 512 ciclos de vitalización cada 8 ms.

La memoria RAM se accesa por su procesador asociado o bien por un procesador externo. Cuando se accesa por el procesador externo, el procesador local es deshabilitado y la tarjeta llega a ser una memoria externa para el otro procesador.

Modo bus local

En este modo, sólo el procesador local tiene acceso a su memoria RAM asociada (memoria que se considera local). Por lo tanto, éste inicia un cambio de estado que es descodificado por el controlador de bus integrado en el circuito IPCU. Como resultado, la dirección del procesador se almacena en el descodificador de direcciones. Si ésta indica la selección de memoria RAM (bits ADS0,1= 0,0), la lógica de control de memoria del circuito IPCU inicia un ciclo de memoria.

Modo bus multimaestro

En este modo, tanto el procesador local V30 como el procesador externo 8086 tienen acceso a la memoria local RAM. EL circuito IPCU controla el acceso a memoria, al informarse mediante la señal MAP (fig. 4.2) (Mode Access Processor), manejando un arbitro de bus, el cual decide y otorga el acceso ya sea al procesador remoto o al procesador local. Para el caso en que al procesador externo se le otorgue el acceso, cuando éste lo haya solicitado previamente, mediante la señal BREQ (fig. 4.5) (Bus Request), el procesador externo sólo proporciona la dirección y el estado para que el circuito IPCU los guarde y los descodifique. Si el procesador local gana el acceso a la memoria, se le informará al procesador externo por la señal de condición ocupada a través del árbitro de bus.

Arbitro de acceso a memoria

Un árbitro decide si el ciclo que se comenzó es para el procesador o para una revitalización de memoria. Este árbitro lo controla también el circuito IPCU.

Lógica de corrección y detección de errores (EDC)

El circuito IPCU contiene la lógica necesaria para generar los bits de verificación de una palabra dato de 16 bits según el código de Hamming modificado que será visto en el apartado 4.5, y para corregir la palabra dato

leída de memoria cuando los bits de verificación son generados al operar el IPCU sobre dicha palabra dato. La lógica EDC va a corregir un error simple y detectar errores múltiples. Para esto se utilizan 6 bits de verificación por cada palabra dato. La lógica EDC se habilitar cuando el bit 0 (HINH= HAMMING LOGIC INHIBIT), del registro PCR del circuito IPCU es cero, o si se hace un requerimiento mediante la entrada BREQ, momento a partir del cual el Hamming se activará.

La bandera SERC (Simple Error) del circuito IPCU se activa únicamente por la ocurrencia de un error simple. La bandera MERR (Multiple Error) del circuito IPCU se activa si ocurren dos o más errores.

También se incluye un circuito cerrojo de ocho bits para guardar el síndrome del código de Hamming utilizado aquí, y puede accesarse a través del puerto de entrada/salida con dirección 32H (H=hexadecimal). Los seis bits menos significativos del cerrojo contienen la información del síndrome, los otros dos restantes se leerán como 0's. Esta información es el resultado de la suma módulo 2 de los bits verificadores generados a partir de los datos de entrada y los bits verificadores generados a partir de los datos que se leen de la memoria.

Manejo de datos

El diagrama de bloques de la figura 4.4 indica la estructura básica de la lógica del manejo de datos.

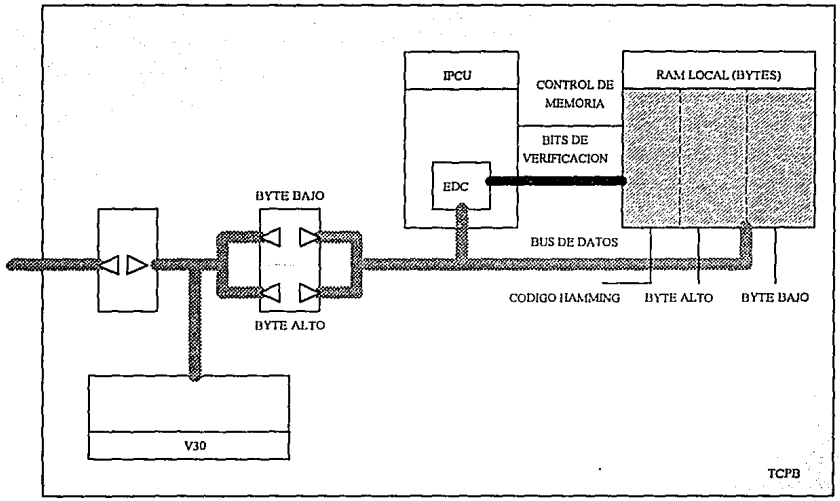


Figura 4.4

Capacidad RAM local: datos: 16 bits x 512 kbytes
Código Hamming: 6 bits x 512 kbytes

Función del IPCU: señales de control de la RAM + lógica EDC

Bus de Datos: bidireccional de 16 bits divididos en byte alto y bajo, recibidos y transmitidos al procesador.

Durante los ciclos de escritura se generan bits de verificación según el código Hamming sobre la palabra de datos. Estos bits se almacenan en la memoria junto con la palabra de datos.

Durante el ciclo de lectura se utiliza el código Hamming para generar bits de verificación a partir de la palabra dato leída. Estos bits se utilizarán para hacer una comparación con los bits de verificación generados en el ciclo de escritura y almacenados en la memoria junto con la palabra dato, con el objeto de corregir errores simples dentro de la palabra dato.

La corrección de un solo bit es transparente; sin embargo, el circuito IPCU indica mediante la señal SERC para un acceso a la lectura si debe realizarse o no una corrección. Si se detecta un error doble, el circuito IPCU cambia la señal MERR al estado bajo, para indicar que no se ejecutará la corrección del error.

Para permitir la ejecución de la rutina de prueba a la memoria, se debe activar la señal HINH (HAMMING INHIBIT) para inhibir la lógica de corrección de errores y que los ciclos de memoria operen como sigue:

En ciclos de escritura: el código de verificación liberado por la palabra direccionada permanece inalterada al escribir la palabra de datos; en ciclos de lectura: no hay corrección de errores y por lo tanto la señales SERC y MERR se inhiben aún cuando se detecte un error.

Se tienen diferentes ciclos de operación a la memoria:

a) Lectura de una palabra

Durante los ciclos de lectura, se leen 22 bits de la localidad direccionada de memoria. Estos 22 bits se aplican al generador del síndrome en la lógica de corrección de errores EDC que está integrada al IPCU. Las salidas corresponden al síndrome que permite la detección de error: si todos los bits del síndrome son cero, no hay error; si uno o más bits están en estado alto, la palabra tendrá error; si el síndrome tiene peso par, existe un error doble en la palabra. En caso contrario habrá sólo un error.

Sí se detecta un sólo error, el generador del código de corrección define la posición del bit erróneo, entonces éste es corregido mediante el corrector de datos interno.

b) Escritura de una palabra

Durante el ciclo de escritura se genera la palabra de verificación de Hamming a partir de los 16 bits de datos que provienen de la interfaz del procesador.

En este ciclo, el generador del síndrome funciona como generador del código, aterrizando las seis entradas correspondientes a la palabra de verificación. La salida del generador del síndrome corresponde a la palabra de verificación que es escrita en la RAM simultáneamente con los 16 bits de datos.

c) Lectura de un byte

Internamente, se realiza de manera similar a la lectura de una palabra, dado que este sistema de corrección se basa en un código de corrección generado para una palabra. De tal manera que sólo se solicita un byte, el menos significativo (LSB) o el más significativo (MSB) que va a transmitirse a la interfaz del procesador.

d) Escritura de un byte

Cuando se solicita la escritura de un byte, primero se genera una nueva palabra que es elaborada a partir de dos bytes, uno recibido de la interfaz del procesador y el otro leído (y posiblemente corregido) de la localidad direccionada de memoria de acuerdo a la siguiente tabla:

BHE	ADDR00	BYTE BAJO	BYTE ALTO
0	0	INTERFAZ	MEMORIA
1	1	MEMORIA	INTERFAZ

Primer ciclo: lectura

Una corrección de error leída del contenido original de la palabra direccionada para asegurar la corrección de datos originales

Segundo ciclo: escritura

Una nueva palabra de datos se genera a partir del byte que se va a escribir y los contenidos ya corregidos para el otro byte, y un nuevo código de verificación Hamming se genera para esta nueva palabra. La nueva palabra completa es escrita en la memoria. Las señales MEMEC y MEMDER indican si hay o no corrección o detección de errores previos.

4.4 Descripción Funcional de la Unidad de Control de Proceso Integrado IPCU (IPCU: Integrated Processor Control Unit)

La unidad IPCU es un circuito digital monolítico LSI de tecnología C-MOS tipo comercial. Soporta varias funciones de control necesarias en una tarjeta construida con base en el microprocesador 8086. Además, incorpora una interfaz de comunicación serie capaz de manejar una comunicación bidireccional simultánea asíncrona, un controlador de RAM dinámica, una lógica EDC de 16 bits, y una lógica de temporización. El circuito actúa como interfaz directamente en el bus multiplexado de datos/direcciones.

Para la tecnología C-MOS utilizada en la construcción del circuito se requieren 12 V de alimentación; el circuito está empaquetado en una envoltura que debe ser montada sobre un soporte de 84 entradas para terminales. La figura 4.4 muestra el aspecto físico del circuito.

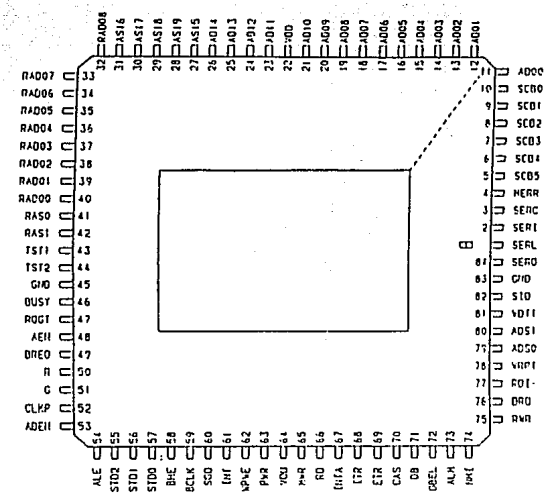


Figura 4.5 Disposición física del IPCU

Bloques funcionales del IPCU

- Registro de Control del Procesador (PCR) y Registro de Control Extendido del Procesador (EPCR)

El PCR agrupa las tareas de control y estado, que permiten el control por software de los estados de la memoria y del procesador. Consiste de un registro de 16 bits organizado como registro de sólo lectura (byte alto del PCR) y registro de escritura/lectura. Puede leerse o escribirse en él con dos modalidades: modo byte y modo palabra.

El EPCR se constituye de un registro con 8 bits de sólo lectura, byte menos significativo (LOW byte), y 8 bits de lectura/escritura, byte más significativo (HIGH byte). El EPCR también se puede accesar en modo byte y modo palabra.

Dado que el circuito IPCU no ofrece terminales disponibles para accesar estos registros, se utilizan las terminales SERI (SERIAL INPUT) y SERO (SERIAL OUTPUT). Las disposiciones de los bits de cada registro se muestran en la figura 4.6.a y 4.6.b

- Operación de la interfaz en serie SI

La interfaz en serie o SI (SI: Serial Interface) permite al procesador comunicarse con un dispositivo externo a una velocidad de 1200 Bauds, que puede cambiarse a 9600 Bauds cambiando a nivel alto el bit 12 HBDRT del registro EPCR.

Esta interfaz es tipo asíncrona bidireccional alternada. Proporciona un transmisor y un receptor ambos independientes y con memoria intermedia doble. Tanto receptor como transmisor operan a la misma velocidad. No se prevee la detección de errores excepto cuando hay un sobreflujo a la entrada de la memoria intermedia que se indica con el bit 5 en el registro EPCR.

- Bus Arbitro

El bus árbitro en el IPCU actúa en conjunto con el dispositivo externo compatible 8289.

Tiene la capacidad de manejar la señal de protocolo RQ/GT (Request/Grant) del microprocesador 8086. También están provistas las señales BUSY, para protocolo de señales; BCLK (Bus Clock), para la sincronización de reloj; y AEN (Arbiter Enable), para habilitación de señal, para el control de las memorias intermedias de la tarjeta del procesador donde se aloja el IPCU. El bus árbitro se controla con la señal de solicitud de bus BREQ (Bus Request).

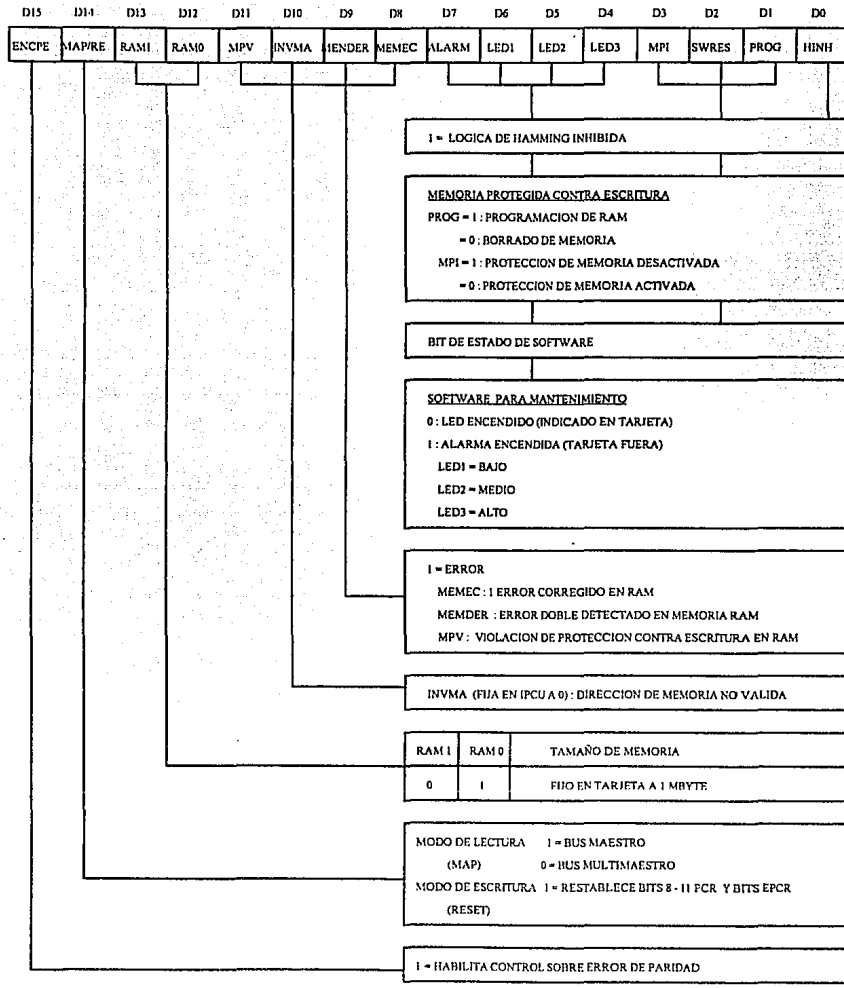


Fig. 4.6.a Asignación de bits en el registro de control PCR del TCPB

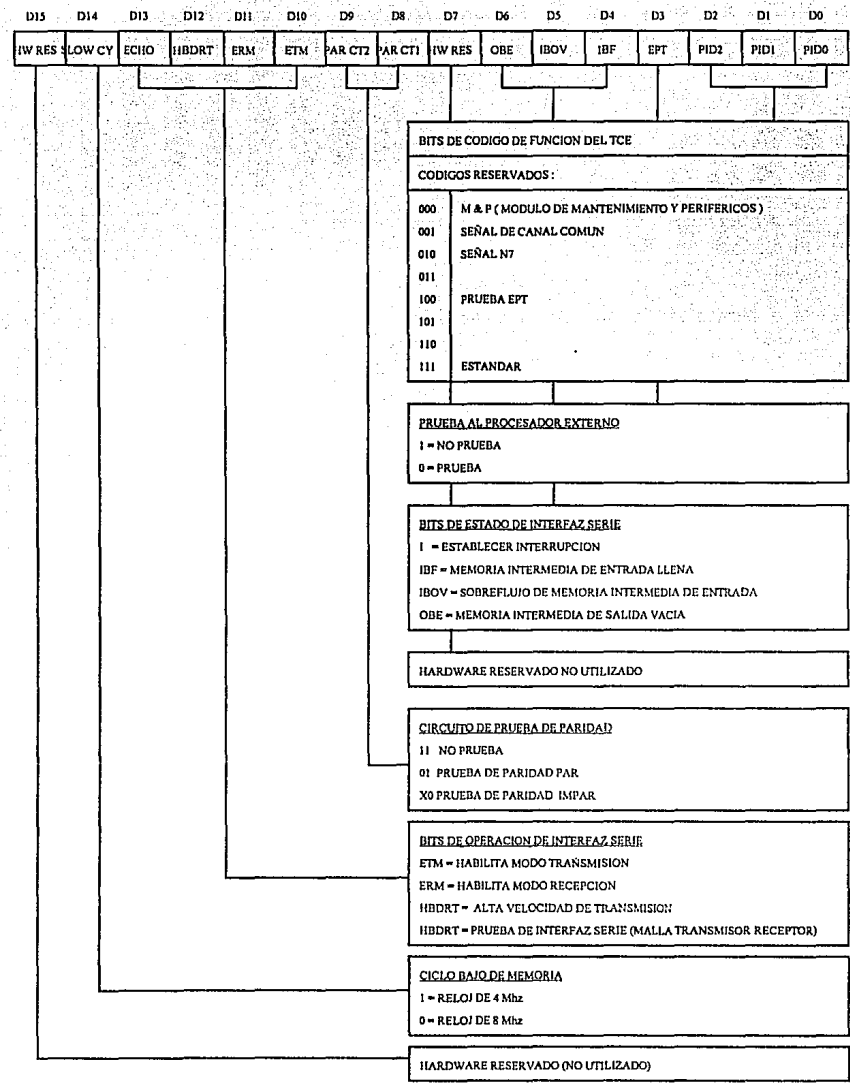


Fig. 4.6.b Asignación de bits en el registro de control extendido EPCR del TCPB

Arbitro de acceso a memoria

Se incluye un árbitro de acceso a memoria para evitar conflictos entre una solicitud de reescritura y una solicitud de acceso a memoria. La reescritura de memoria se solicita cada 15 μs. Esto asegura la ejecución de 512 ciclos de refresco cada 8 ms, con las direcciones de reescritura a memoria presentes en las salidas RAD0-X, X=1, 2, 3..., 8.

Lógica EDC

La lógica EDC que se ha implantado opera sobre 16 bits de datos durante el ciclo de memoria. Proporciona corrección y detección de un bit erróneo y detección de un error doble.

4.5 Descripción del código de Hamming aplicado a la central telefónica digital 1240 de ITT

Fundamentos de Corrección y Detección de errores

Se tomarán como fundamentos los siguientes aspectos:

- Un sistema de memoria es una componente del sistema en la cual la confiabilidad es importante. También es una componente del sistema que puede modificarse ampliamente para enriquecer su confiabilidad.

Para incrementar la confiabilidad del sistema más allá de la confiabilidad del dispositivo, se han desarrollado técnicas de codificación con redundancia para la detección y corrección de errores.

Las primeras formas de detección de errores en las computadoras se centraban en un simple esquema de verificación de paridad o en la verificación de una suma pregenerada.

- Con la técnica de la paridad, un bit agregado a cada palabra forman la palabra codificada. La paridad es par o impar, dependiendo del esquema elegido. Cuando el sistema recupera la palabra de la memoria, ésta regenera la paridad de la palabra y la compara con la paridad del bit agregado.

Aunque con la paridad se detecta el error, no es posible su corrección, debido a que cada uno de los errores que pueden ocurrir tiene un efecto idéntico y por tanto se hace imposible la reconstrucción de la palabra original.

Si ocurre un error doble, éste no se detecta porque son dos los bits que han cambiado de estado, provocando que el peso de la palabra sea el mismo (peso es la suma de todos los unos de palabra).

- Otro método de detección de errores, que es popular porque no requiere de Hardware externo, utiliza sumas módulo 2 de verificación para un bloque de datos entero. Una suma de verificación es una suma aritmética de cada palabra de datos en el bloque y es almacenada en la memoria al final del bloque.

Al leer el bloque se genera una nueva suma de verificación y se compará con la original.

Como en los sistemas de paridad, no es posible corregir un error cuando este ocurre. Entonces el sistema debe intentar releer o reescribir los datos.

Corrección de Errores

Los dos sistemas mencionados anteriormente introducen una cierta confiabilidad en el sistema, si ocurre un error éste se detecta. En la actualidad, sin embargo, se prefieren sistemas más sofisticados y más poderosos, los cuales, en algunas ocasiones hacen crítico su empleo debido a la cantidad de veces que accesan la memoria y a la velocidad de operación requerida.

Como consecuencia de un artículo publicado por R. W. Hamming (Abril de 1950) sobre códigos de corrección de errores, el tipo de código más ampliamente usado es el Código Hamming.

Los códigos Hamming agregan bits a las palabras de datos para dar una verificación válida a la palabra completa. Esos bits adicionales que se incluyen, haya o no ocurrido un error, se llaman bits de codificación o bits de verificación. Para ilustrar el mecanismo del Código Hamming, se ha utilizado el esquema básico de la figura 4.7. Además de la memoria los Circuitos de Corrección de Errores (ECC: Error Correction Circuits) tienen cinco componentes, a saber:

- Generador del bit de verificación de escritura.

- Generador del bit de verificación de lectura.

- Generador del síndrome.

- Corrección del bit

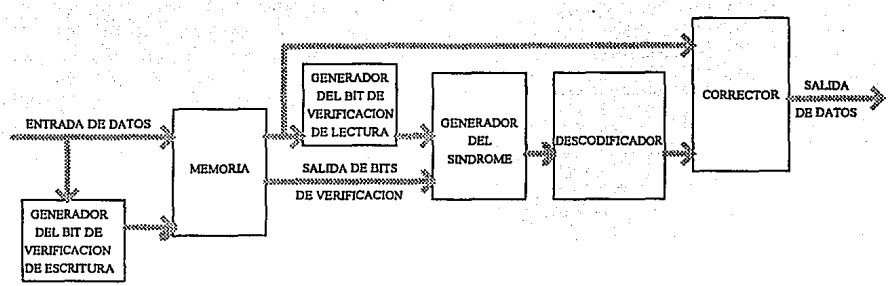
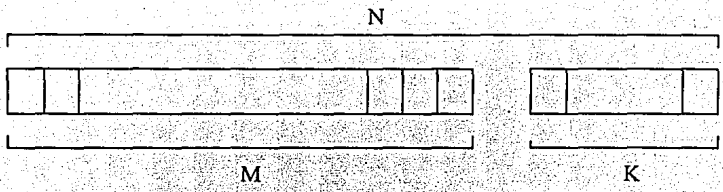


Figura 4.7 Diagrama a bloques del sistema EDC (Error Detection & Correction) basado en el código Hamming

- Generador del bit Verificador de Escritura y Generador del bit Verificador de Lectura

Los bits verificadores de escritura se generan cuando se escriben los datos en la memoria, mientras que los bits verificadores de lectura se generan cuando se leen los datos de la memoria.

La palabra codificada en la memoria contiene los bits adicionales, según se muestra en la siguiente figura:



N = número de bits en la palabra codificada
M = número de bits de datos
K = número de bits de codificación

En esta figura la distribución de los M y los K bits no guardan estrictamente el orden indicado, sino que su orden obedece a una distribución particular.

El número de K bits de codificación necesarios para corregir un error que contiene una palabra de datos de longitud M se lista en la siguiente tabla [2]

K BITS DE VERIFICACION	M BITS DE DATOS	
	CORRECCION SIMPLE/ DETECCION SIMPLE	CORRECCION SIMPLE/ DETECCION DOBLE
4	$5 \leq M \leq 11$	$1 \leq M \leq 3$
5	$12 \leq M \leq 26$	$4 \leq M \leq 10$
6	$27 \leq M \leq 57$	$11 \leq M \leq 25$
7	$58 \leq M \leq 120$	$26 \leq M \leq 56$
8	$121 \leq M \leq 247$	$57 \leq M \leq 119$

Así, para detectar y corregir un bit erróneo en una palabra de datos de 16 bits, se deben utilizar 5 bits de codificación. Como resultado, el número de bits en total de la palabra codificada es 21. Se puede generar una tabla llamada carta de Hamming donde se organizan los N = M+K bits de la palabra indicando su posición decimal, ver figura 4.8

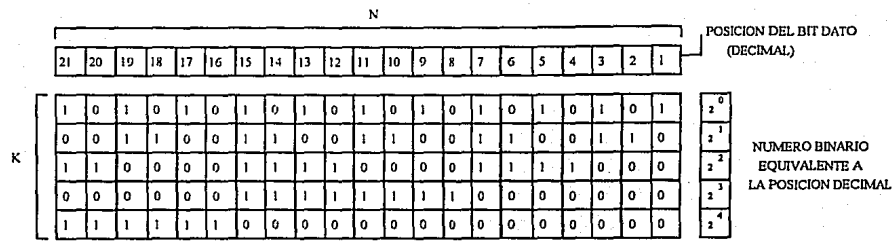


Figura 4.8 Carta de Hamming

Nótese que en la figura aún no se especifica donde se localizan los M y los K bits. Los valores de la posición decimal de cada bit de la palabra codificada se convierten a su equivalente binario representados en la tabla como las columnas que se encuentran debajo del número de posición (decimal) del bit. De la tabla anterior la longitud de los equivalentes binarios es K = 5 bits.

El peso de la palabra de verificación se determina solamente por los 1's (unos) en las posiciones de los bits indicados en la figura 4.8, siendo así estos serán reemplazados por una x y los 0's se eliminarán.

El resultado se muestra en la figura 4.9.

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
X		X		X		X		X		X		X		X		X		X		X	C1
		X	X			X	X			X	X			X	X			X	X		C2
X	X					X	X	X	X					X	X	X	X				C4
						X	X	X	X	X	X	X	X		.						C8
X	X	X	X	X	X																C16

DONDE C1, C2, C4, C8, Y C16 SON LOS BITS DE VERIFICACION

Figura 4.9 Conversión de la Carta de Hamming

Las cinco posiciones, columnas 1, 2, 4, 8 y 16 tienen solamente una 'x' en sus columnas como se aprecia en la figura 4.9. Para efectos de cálculo de la palabra código, los bits de verificación se localizarán en esas posiciones, y los bits de datos se localizarán en las 16 posiciones restantes que están entre los bits de verificación. Quedando así definida la distribución de los M bits de datos y los K bits de verificación que componen la palabra código.

La figura 4.10 muestra las posiciones de (M) bits de datos y (K) bits de verificación de la palabra código para M=16.

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	22	1	POSICION
16	15	14	13	12		11	10	9	8	7	6	5		4	3	2		1			BITS DE DATOS
					C16								C8				C4		C2	C1	BITS DE VERIFICACION

Figura 4.10 Distribución de los bits de datos y los bits de verificación en la palabra código.

Los bits 1, 2, 4, 8, y 16 se obtienen a partir de la circuitería de generación y son almacenados en memoria según la figura 4.10.

La parte de los bits de datos M1, M2,..., M16 de la figura 4.10 se utiliza para generar los bits de verificación. Para calcular un bit de verificación se efectúa una suma módulo 2 entre los bits de datos cuyas posiciones indicadas en la fig. 4.10 coinciden con una 'x' al sobreponerlos en el renglón correspondiente, de la figura 4.9, al bit de verificación por calcular como lo indica la figura 4.11.

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
M16		M14		M12		M11		M9		M7		M5		M4		M2		M1			C1
		M14	M13			M11	M10			M7	M6			M4	M3						C2
M16	M15					M11	M10	M9	M8					M4	M3	M2					C4
						M11	M10	M9	M8	M7	M6	M5									C8
M16	M15	M14	M13	M12																	C16

DONDE C1, C2, C4, C8, Y C16 SON LOS BITS DE VERIFICACION

Figura 4.11 Posición en que se ordenan los bits de datos de la palabra código respecto a la 'carta de Hamming' para obtener los bits de verificación.

Así los bits de verificación se definen por las siguientes ecuaciones:

$$C1 = M1 \oplus M2 \oplus M4 \oplus M5 \oplus M7 \oplus M9 \oplus M11 \oplus M12 \oplus M14 \oplus M16$$

$$C2 = M1 \oplus M3 \oplus M4 \oplus M6 \oplus M7 \oplus M10 \oplus M11 \oplus M13 \oplus M14$$

$$C4 = M2 \oplus M3 \oplus M4 \oplus M8 \oplus M9 \oplus M10 \oplus M11 \oplus M15 \oplus M16$$

$$C8 = M5 \oplus M6 \oplus M7 \oplus M8 \oplus M9 \oplus M10 \oplus M11$$

$$C16 = M12 \oplus M13 \oplus M14 \oplus M15 \oplus M16$$

- Generador del Síndrome, Descodificador y Corrección de Bits

El síndrome del código se genera haciendo una suma módulo dos entre los bits de verificación obtenidos a partir de los bits de datos que se leen de memoria y los bits de verificación regenerados.

Cualquier bit de verificación nuevo que difiera de los bits de verificación originales será puesto igual a uno en el síndrome. La identificación del bit erróneo hecha por el síndrome será dada por el valor binario de la posición de dicho bit. Por ejemplo, para identificar si el bit 3 es erróneo, el síndrome correspondiente será 00011. Como los bits de verificación tienen sólo una x en su columna (figura 4.9), estos son independientes de los demás.

Si un bit de verificación falla, el síndrome, que es el resultado de la suma módulo 2 entre los bits de verificación de lectura y los bits de verificación de escritura, contendrá un sólo '1'. Una falla en los bits de datos se podrá identificar con dos o más 1's en el síndrome. Así, el proceso no solamente revela el error sino que con ayuda del descodificador, puede decir exactamente cual es el bit erróneo. Luego entonces el corrector puede decidir sin equivocación que bit de datos va a corregir.

Ejemplo del Código "Directo" de Hamming

Para ilustrar plenamente la aplicación y el funcionamiento del código de Hamming para corregir errores dentro del sistema 1240 de ITT, se da un ejemplo de ello.

- Se supondrá que la palabra de datos es de 16 bits:

5039H, o bien 0101 0000 0011 1001 (binario).

Ahora esta palabra se escribe a la memoria junto con los bits de verificación generados en este paso, figura 4.12.

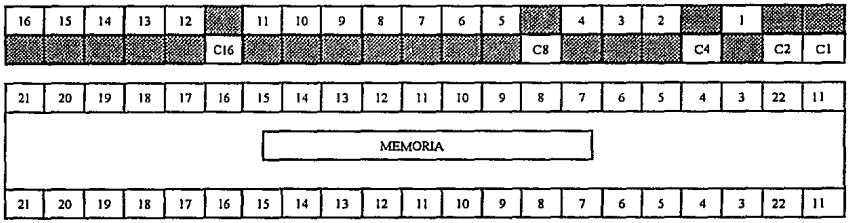


Figura 4.12 Escritura de la palabra código en la memoria

- Los bits de verificación se generan al pasar la palabra de datos por la Carta de Hamming de la fig. 4.8 y ejecutando el cálculo de la suma módulo dos de los bits marcados con 'x'. Los bits de verificación generados son para este ejemplo 11110 (binario), ver figura 4.13.

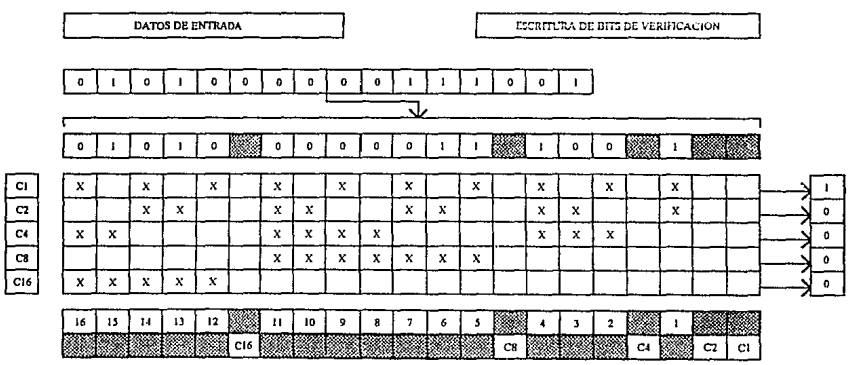


Figura 4.13 Generación de los bits de verificación a partir de los bits de datos.

- Estos bits de verificación se incorporan a los datos, formando así la palabra codificada que se almacenará en memoria.

- Ahora se forza a que el error en los datos leídos de memoria sea en la posición decimal 7 del bit, que corresponde al bit de datos 4, de esta manera se genera un nuevo conjunto de bits de verificación cuando la palabra codificada que contiene el error ya indicado se lee de memoria, los nuevos bits de verificación son 11001 (binario), ver figura 4.14.

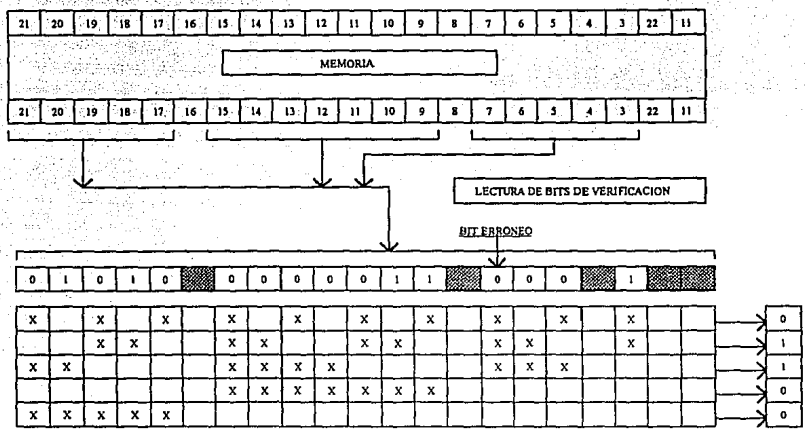


Figura 4.14 Lectura de la palabra de datos y generación de los bits de verificación.

- Ahora se obtiene el síndrome aplicando la suma módulo 2 a los bits de verificación nuevos 11001 con los bits de verificación originales 11110 (leídos de memoria).

El síndrome resultante es 00111 (binario).

Este resultado indica que la posición 7, correspondiente al bit de datos 4, es erróneo, lo cual se ilustra en la figura 4.15.

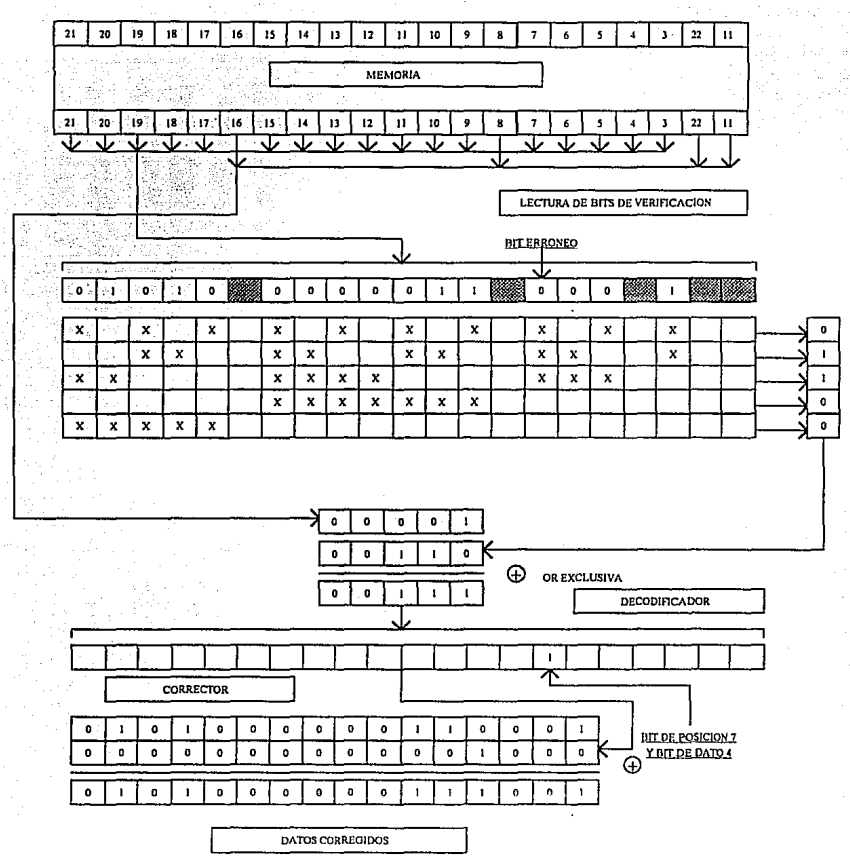


Figura 4.15 Etapa generadora del síndrome

Conclusiones acerca de la aplicación del código Hamming (directo)

Aunque el Código de Hamming directo es simple, su implantación física en el sistema 12 (en hardware) representa algunos problemas:

Primero, el número de bits a los que se les aplica la suma exclusiva genera un retardo en la trayectoria, lo que hace lento al sistema. El retardo de propagación de una compuerta lógica que efectúa suma exclusiva de 10 entradas es mucho mayor que el retardo de la misma compuerta pero con cinco entradas.

Por otro lado, dos bits erróneos pueden causar que un bit correcto sea indicado como erróneo. Por ejemplo, si los bits de verificación C1 y C2 fallan, el bit dato 1 sería marcado como error.

Debido a estas dificultades, el Código de Corrección de Errores (ECC) más comúnmente utilizado es un Código de Hamming 'modificado', que detecta errores dobles y corregirá un sólo error.

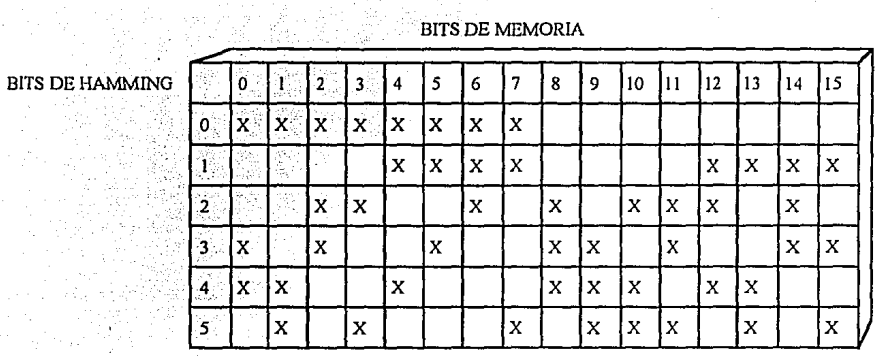
Código de Hamming modificado (Detección Doble, Corrección Simple)

Desarrollo del Código

Con métodos de Algebra moderna se prueba que se requiere una distancia mínima de 4 entre dos palabras codificadas para detectar dos errores o corregir un solo error (ver ecuación (2.1)).

Como resultado se tienen tres bits de verificación como mínimo para proteger cada bit de datos. Si no hay errores, el peso del síndrome es cero, el cual puede considerarse que es par. Los síndromes con peso par no pueden utilizarse. Si dos bits de verificación fallan, el síndrome tendrá dos unos o peso par, el peso par se detecta como un error doble al hacer la verificación de la paridad sobre el síndrome. Si fallan dos bits de datos, el síndrome tendrá nuevamente un peso par y un error detectable. Al agregar un bit de verificación más a los bits de verificación se tiene la capacidad de detectar errores dobles.

Los síndromes para el código SBC/DBD (Corrección de un Bit Erróneo/ Detección de Dos Bits Erróneos) se desarrollan de la misma manera que en el código de Hamming directo, excepto que, los síndromes no mapean directamente las posiciones de los bits, esto es, no se precisan posiciones como en el ejemplo anterior. En Hamming modificado, el síndrome tiene un peso impar y no se incrementa como en el ejemplo del código Hamming 'directo'. Las palabras síndrome pueden mapearse para cualquier posición del bit, garantizando códigos idénticos generados al momento de guardar y recuperar información de memoria. La figura 4.16 muestra el formato para la generación del código como lo utiliza la memoria del TCPB.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	X	X	X	X								
1					X	X	X	X					X	X	X	X
2			X	X			X		X		X	X	X		X	
3	X		X			X			X	X		X			X	X
4	X	X			X				X	X	X		X	X		
5		X		X				X		X	X	X		X		X

Figura 4.16 Formato de la carta de Hamming modificada [3]

De la figura anterior se observa que todos los renglones tienen 8 "x" y que cada columna tiene 3 "x". Todas las columnas son diferentes y cada una de ellas representa el síndrome para el bit de memoria correspondiente en error.

Ejemplo del código de Hamming modificado

En este ejemplo la implantación hardware de este código es similar a la del código Hamming directo.

- Se asume que la palabra dato con longitud 16 bits = 5039H.

- Los bits de verificación se generan al pasar la palabra dato por la Carta de Hamming y ejecutando el cálculo de paridad par sobre los bits cuyas posiciones coinciden con las marcas 'x' de la carta, ver figura 4.17. De la figura 4.17 también se aprecia que las posiciones de los bits de verificación están aterrizados, por lo que sus valores iniciales son cero.

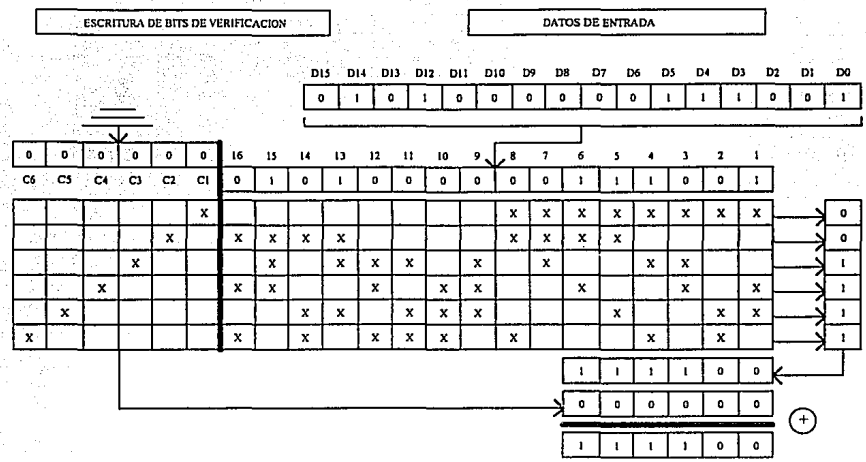


Figura 4.17 Generación de la palabra código con la carta de Hamming modificada

Los bits de verificación, resultado de hacer la suma módulo dos entre los valores iniciales de la palabra de verificación y los bits de paridad, en la escritura son: 111100 (binario). Estos bits de verificación se incorporan a los datos, formando así la palabra codificada que se escribe en la memoria, ver figura 4.18.

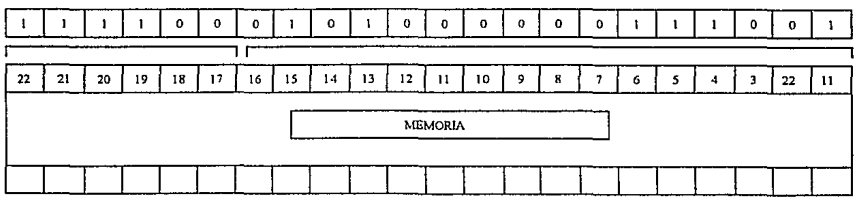


Fig. 4.18 Incorporación a la memoria de bits de datos y bits de verificación.

- Ahora se fuerza a que el error, generado dentro de la memoria sobre la palabra dato, sea el bit de posición 4, bit dato 3, debido a esto se generar un nuevo conjunto de bits de verificación verificación cuando la palabra codificada se lee de la memoria.

Los nuevos *bits* de verificación son = **011001** (binario)

La generación de estos nuevos *bits* de verificación se ilustra en la figura 4.19.

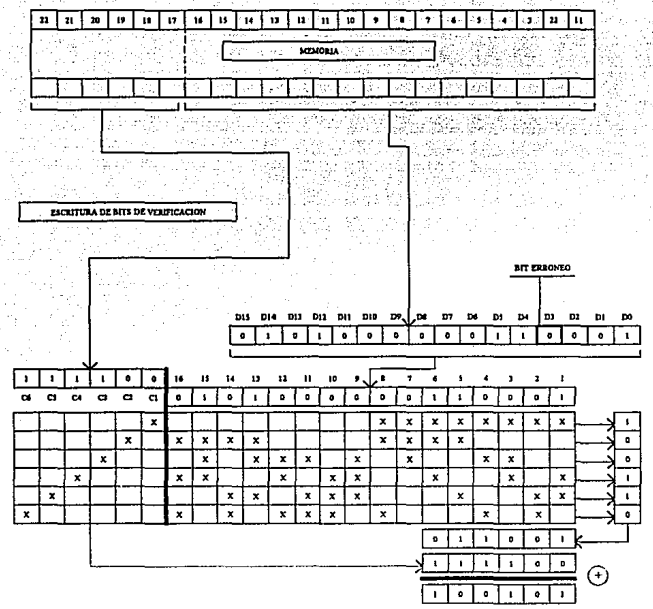


Figura 4.19 Generación de los *bits* de verificación al leer información de memoria

- Cuando se hace la suma modulo dos de los *bits* de verificación generados al leer la palabra dato de la memoria con los *bits* de verificación generados al escribir la palabra dato en la memoria, se obtiene el síndrome que en este caso resulta :

100101 (binario)

- Ahora se realiza una verificación para ver si existe un sólo error o un error doble, recordando que:

Paridad del síndrome impar = > un sólo error.

- El síndrome resultante puede guardarse en la tabla de descodificación mostrada en la figura 4.20 para encontrar la posición del *bit* erróneo, en caso de que así suceda; en este caso el valor del síndrome corresponde en la tabla al *bit* 4, el descodificador toma entonces la palabra correctora correspondiente y aplica una suma exclusiva entre ésta y la palabra dato previamente leída de memoria regenerando así la palabra de salida correcta, como se aprecia también en la figura 4.20.

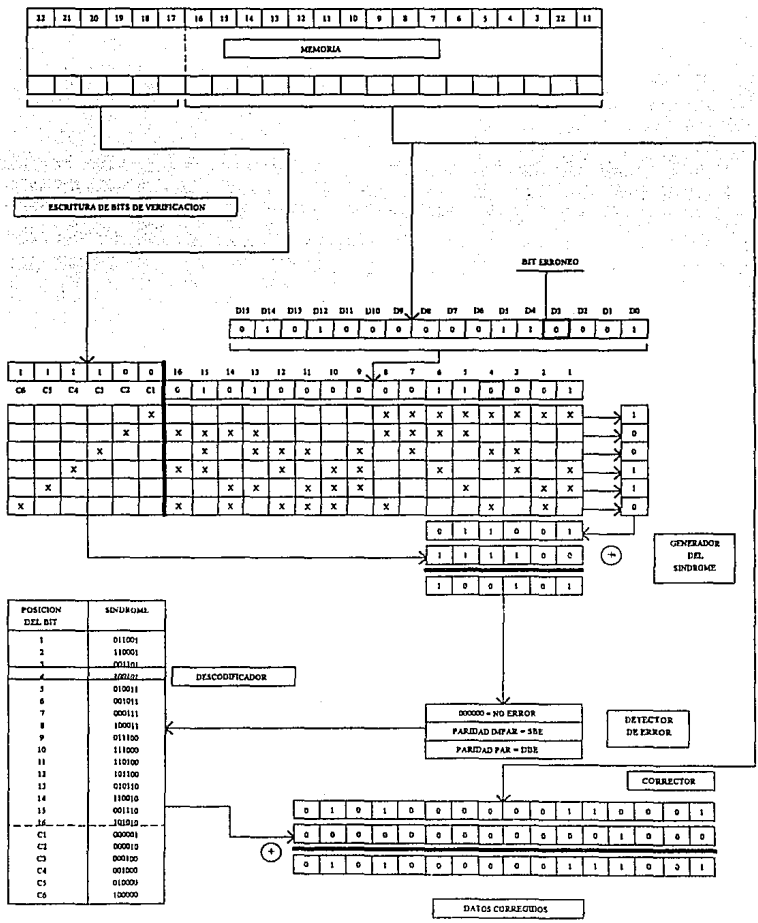


Figura 4.20 Tabla de descodificación y corrector de la palabra dato.

4.6 Descripción de la aplicación del algoritmo de descodificación desarrollado con aplicación práctica al sistema de telefonía en cuestión

Se da a continuación un ejemplo de aplicación siguiendo la secuencia de descodificación que se propone en el capítulo 3 para descodificar una secuencia de datos que serán almacenados en el sistema de memoria, contenido en el **TCPB** de cada elemento de control de la central telefónica, y posteriormente recuperados de ella.

Antes de almacenar los datos se supondrá que se codificarán y se almacenarán secuencialmente organizados en triples. Al recuperarse de la memoria en forma secuencial, serán leídos los mismos triples, pero con cierta probabilidad de que alguno de ellos no sea la que originalmente se almacenó; por lo cual los triples leídos serán descodificadas y se decidirá de acuerdo al criterio del algoritmo sobre cual de los *bits* descodificados se harán correcciones.

La información se almacena en un sistema de memoria que incluye la lógica de corrección de errores con el fin de dar confiabilidad al mismo sistema, y el espacio en memoria necesario para almacenar toda la información ya codificada y organizada en triples de tal manera que al leer la información se pueda reconocer ésta como triples generados por el codificador y así ser descodificados por el algoritmo propuesto.

El sistema se concibe, en una primera aproximación, como un conjunto de bloques de los que la memoria es uno de ellos, ver figura 4.21; los bloques restantes, seis en total, son los siguientes:

- Codificador propuesto
- Generador de triples supuestos
- Comparador de triples
- Acumulador de diferencias
- Descodificador
- Corrector

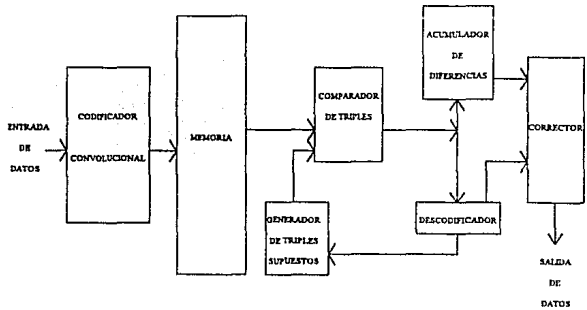


Figura 4.21 Sistema de detección y corrección de errores de la memoria del TCPB, basada en el algoritmo propuesto.

Se da ahora una breve explicación funcional de cada bloque:

Codificador

De la figura 4.21 se observa en primer instancia que la información a escribir en memoria será codificada totalmente, esto es, todos los *bits* de entrada han de pasar por el codificador y cada *bit* que entre generará tres *bits* a la salida del codificador. Los triples generadas a la salida del codificador se relacionan con los *bits* de entrada mediante la ecuaciones (3.3), (3.4) y (3.5). Por similitud con el sistema de detección y corrección de errores que está implantado actualmente en el sistema 1240 de ITT se supondrá que se almacenará la información por triples.

Memoria

Los datos almacenados en la memoria serán los triples codificados organizados en grupos de 16. Los triples también serán leídos en grupos de 16 para posteriormente poder hacer una comparación del desempeño del código con el código ya implantado en el sistema 1240.

El espacio de memoria que ocuparán los triples será tres veces mayor que el espacio que se ocupa originalmente utilizando el esquema de detección y corrección de errores (*EDC*) implantado actualmente en el sistema 1240, lo cual representa una evidente desventaja para este descodificador.

Comparador de Triples

Mediante una suma exclusiva entre un triple supuesto y un triple recuperado de memoria se obtiene un triple cuyos unos indicarán el número de diferencias entre los primeros dos triples.

Descodificador

Mediante el número de diferencias entre el triple supuesto y el triple recibido se descodificará $R(I)=\underline{VR}(I,1)$ si hay una o ninguna diferencia, en caso contrario se descodificará $R(I)=\overline{VR}(I,1)$.

Generador de Triples Supuestos

Este consistirá del codificador convolucional de 3 etapas ya descrito en el capítulo 3 y tomará como entradas los *bits* descodificados.

Acumulador de Diferencias

El sistema incluirá un acumulador de diferencias entre el triple supuesto y el leído de memoria, los sumará hasta llegar al valor de $DIF=3$ momento en el cual habilitará al corrector. El acumulador restablecerá su valor a cero cuando el descodificador realice 3 descodificaciones.

Corrector

El corrector actuará al momento de ser habilitado por el acumulador de diferencias (ADIF=2 y CDEC=3) aplicando un vector corrector sobre los bits descodificados, este vector corrector tiene el valor **10100** binario que corrige las posiciones L y L+2 de acuerdo al criterio del algoritmo. La figura 4.22(a) muestra el caso en el que la secuencia de datos almacenados se recupera de memoria sin error, la figura 4.22(b) muestra la situación en la que la información se recupera de memoria con un error; en la tercera circunstancia, figura 4.22(c), se observa que la información se recupera de memoria con dos errores.

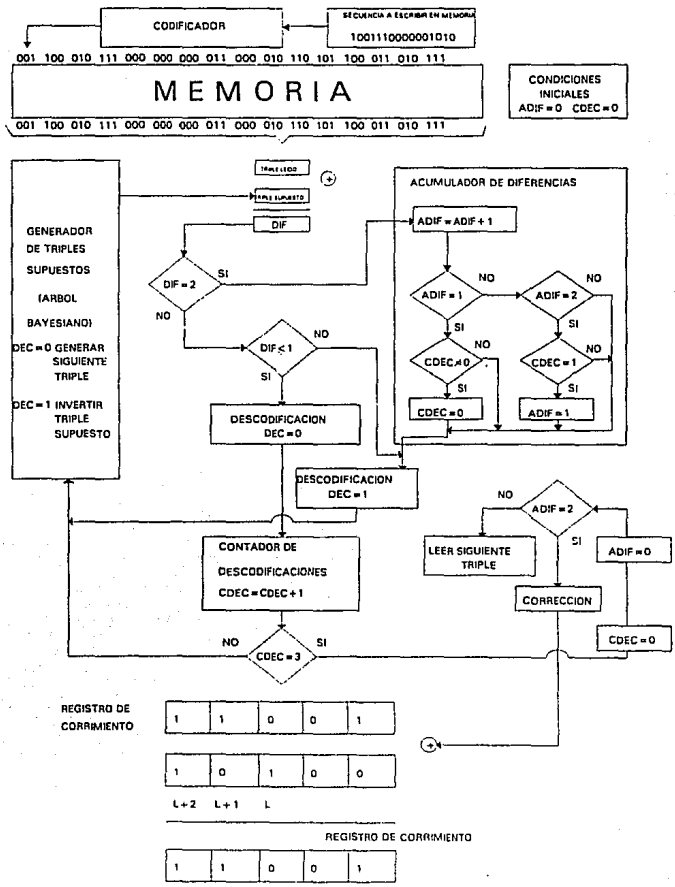


Figura 4.22(a) Recuperación de datos de la memoria sin error

Para los casos en que se tiene un error, o ningún error al recuperar la información de memoria, no se acumularán diferencias al comparar los triples, lo que implica que el corrector no habilitará el vector corrector para actuar sobre los dígitos descodificados y por tanto estos serán los mismos que los dígitos que originalmente se escribieron al sistema de memoria. Lo anterior equivale a decir que para este descodificador un error en la secuencia a descodificar es transparente y por tanto será corregido.

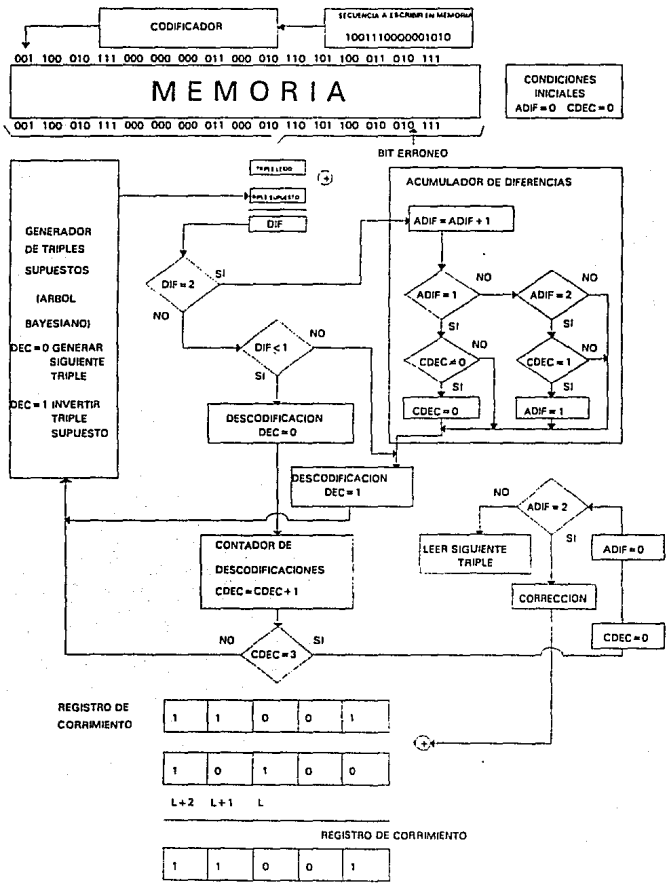


Figura 4.22(b) Recuperación de datos de la memoria con un error

Para la figura 4.22(a) y 4.22(b) se observa que la secuencia descodificada es la misma: **...11001**, siendo que en el caso de la figura 4.22(b) en el tercer triple leído (de derecha a izquierda a la salida de la memoria) contiene un error simple, de donde se deduce que el descodificador puede corregir un error.

En el caso de un error doble en un triple de los recuperados de memoria, se generan triples supuestos que seguirán una trayectoria errónea sobre el árbol Bayesiano y que al ser comparados generan diferencias que al acumularse habilitarán al corrector para que actúe sobre las posiciones descodificadas indicadas por el algoritmo de descodificación.

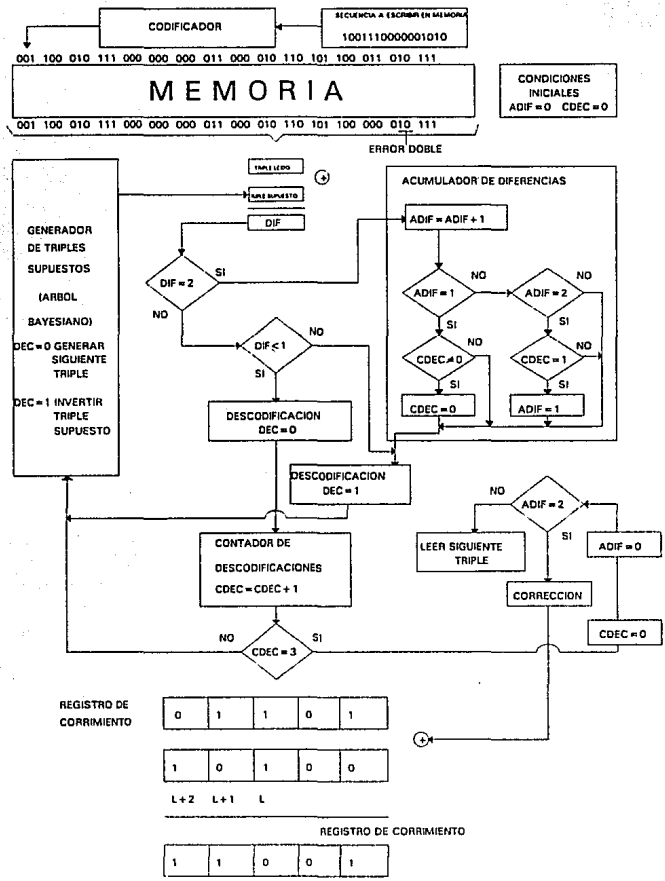


Figura 4.22(c) Recuperación de datos de la memoria con error doble

La figura 4.22(c) muestra la situación en que aparece un error doble en el tercer triple leído de memoria (de derecha a izquierda en la figura), que de acuerdo a la secuencia de triples a descodificar y siguiendo cuidadosamente las etapas de generación de triples supuestos, comparación de triples, y descodificación del sistema de detección y corrección de errores propuesto, se obtendrá la secuencia de los primeros cinco *bits* descodificados siguiente : **...01101**, que comparando con las secuencias descodificadas de las figuras 4.22(a) y 4.22(b) se observa que existen dos errores alternados (posiciones L, L+2) en ésta descodificación, sin embargo, de acuerdo al criterio del algoritmo se ha de aplicar el vector corrector al momento de presentarse esta situación y que se habilita por el acumulador de diferencias que ya para entonces tendrá un valor ADIF=2. El registro de corrimiento de salida tendrá la secuencia corregida **...11001** que es la misma que la descodificada en las figuras 4.22(a) y 4.22(b).

En la figura 4.22 (a) se observa que cada vez que son tomadas tres decisiones sin detección de errores, la variable CDEC se inicializa siempre a cero para iniciar un nuevo conteo de descodificaciones. En el caso de leerse un triple con un solo error, se generará el siguiente triple (generador de triples supuestos) continuando con una nueva decisión; en el caso de detectar un triple con dos diferencias respecto al supuesto, ADIF toma el valor de uno y CDEC se inicializa a cero con el fin de marcar la decisión L (en caso de que CDEC sea diferente de cero), si se ha detectado otro triple con dos diferencias ADIF tomará el valor de dos y se verificará que este evento se haya generado en la decisión L+2 (CDEC=1), ADIF tomará el valor 1, lo que implica ignorar error doble en L+1 dedicando atención sólo a las decisiones L y L+2. Al tomarse cada 3 decisiones, habiéndose, o no, detectado errores, ADIF y CDEC siempre serán inicializados a cero, asegurándose, de esta forma, que el proceso de corrección de errores se realice en forma continua.

El ejemplo anterior hace ver que este algoritmo es capaz no sólo de detectar sino también de corregir errores dobles dentro de un triple, ventaja muy atractiva de un algoritmo para ser utilizada en un sistema de comunicaciones o de memoria, en este caso la confiabilidad al leer información de la memoria del **TCPB** se incrementaría al doble.

La confiabilidad a largo plazo de la tarjeta TCPB queda establecida de acuerdo a las especificaciones de la ITT, en 12000 FITS (fallas en un billón de horas) sin corrección de error, lo cual implica que una vez rebasada esta tasa de error se aplicará el algoritmo de Hamming modificado.

La manera como se vería afectada la velocidad de transmisión de datos de la memoria RAM de la tarjeta TCPB al incrementar dos bits de redundancia por cada bit dato, se deduce a partir de los siguientes valores[41]:

CICLO	DURACION DEL CICLO(μs)	
	frec. de reloj = 4 MHz	frec. de reloj = 8 MHz
MEMORIA EN TARJETA		
- ESCRITURA DE PALABRA	1	0.5
- ESCRITURA DE BYTE	1.5	0.75
- LECTURA	1.25	0.625

Tomando en cuenta que los tiempos proporcionados en la tabla para cada ciclo de lectura y escritura incluyen a los bits de datos y a los de verificación, sumando un total de 22, en el caso en que se agregan dos bits de redundancia por cada bit de dato, el total de bits que se deben de accesar en cada ciclo es de 48 lo cual implica aproximadamente 2.25 veces más la duración de cada ciclo lo cual significa una relativa desventaja

En esta primer aproximación de lo que es el descodificador propuesto no se profundiza en los aspectos funcionales del *Hardware*, dado que llegar a implantar físicamente el descodificador prototipo en la tarjeta del **TCPB** implica costos muy elevados, además tiempo de realización que resultó estar fuera de nuestro alcance; sin embargo, no es difícil establecer que la lógica de descodificación que incluye *hardware* y *software* debe integrarse en el circuito IPCU (descrito en este capitulo) como lo está actualmente la lógica de descodificación de Hamming que utiliza el sistema 1240 de ITT.

Referencias:

[1] Terminal Control Processor-B, Handout, Bell Education Centre

[2] Terminal Control Processor-B, Handout, Bell Education Centre

[3] Terminal Control Processor-B, Handout, Bell Education Centre

[4] Design Specification, PBA TCPB, Documentación Alcatel.

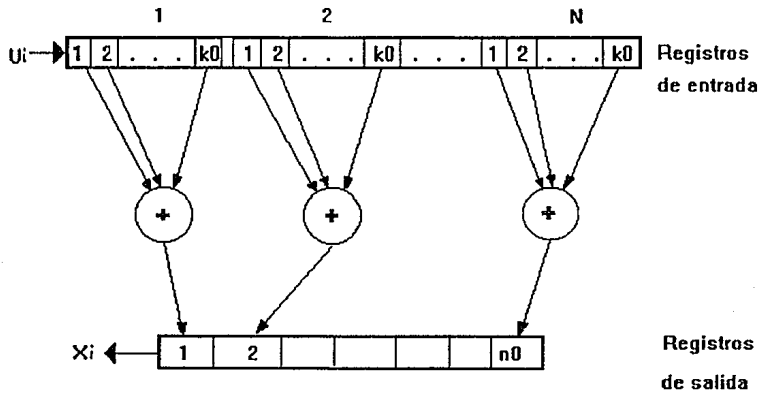
Textos y dibujos elaborados en paquete sw Winword de Microsoft.

APÉNDICE:
LAS CONJETURAS DE VALLEJO SOBRE LA GEOMETRÍA DE LA PROPAGACIÓN DE ERRORES.

PRINCIPIOS DE CÓDIGOS COVOLUCIONALES

Un decodificador convolucional es un sistema de memoria que genera n0 dígitos binarios para todo mensaje de k0 bits que llega a su entrada.

El índice de este código se define como Rc= k0/n0 :



k0 - número de dígitos del me^Al nsaje
N - longitud límite
n0 - dígitos generados (depende de k0 y de los (N-1)k dígitos).

Fig.A1

Se llama un código convolucional de (n0,k0) de una longitud limitada N. Para un bloque de k0 dígitos que se introducen en los registros de entrada , sus salidas se alimentan a n0 sumadores módulo 2; posteriormente se recorre a la derecha el bloque y se alimenta un nuevo bloque de bits de entrada en los registros N.

Se define u como un vector mensaje seminfinito y x el vector codificado correspondiente.

Se puede describir un decodificador convolucional empleando N submatrices G1,G2,G3,...,GN con k0 renglones y n0 columnas.

La submatriz Gi describe las conexiones entre los registros de entrada y el resultado de salida.

Un elemento 1 de Gi significa una conexión y un 0 significa no conexión.

Un código convolucional puede ser sistemático, si en cada segmento de n0 dígitos generados, los primeros k0 dígitos son idénticos a los correspondientes dígitos mensaje.

El arreglo matricial queda de la siguiente forma:

$$G1 = \begin{array}{c} \overset{k0}{\overbrace{}} \quad \overset{n0-k0}{\overbrace{}} \\ \left[\begin{array}{ccccccc|c} 1 & 0 & 0 & . & . & . & 0 & \\ 0 & 1 & 0 & . & . & . & 0 & \\ . & . & . & & & & . & \\ . & . & . & & & & . & P1 \\ . & . & . & & & & . & \\ 0 & 0 & 0 & . & . & . & 1 & \end{array}\right] \end{array}$$

$$Gi = \begin{array}{c} \overset{k0}{\overbrace{}} \quad \overset{n0-k0}{\overbrace{}} \\ \left[\begin{array}{ccccccc|c} 0 & 0 & 0 & . & . & . & 0 & \\ 0 & 0 & 0 & . & . & . & 0 & \\ . & . & . & & & & . & \\ . & . & . & & & & . & Pi \\ . & . & . & & & & . & \\ 0 & 0 & 0 & . & . & . & 0 & \end{array}\right] \end{array}$$

$$i = 2, 3, \ldots, N$$

Fig.A2

Para un código (n0,1) se generan n0 dígitos de la secuencia xi para cada ui dígitos según lo siguiente:

Se tiene una convolucion discreta de los vectores G1,G2,G3,...,GN, ya que las submatrices Gi son en realidad vectores de dimensión (1 x n0), y los N dígitos de la secuencia de entrada Ui,Ui-1,...,Ui-N+1 .

Es más conveniente describir un código convolucional, cuando se tiene dimensión (n0,1), por sus vectores generadores gi, ya que estos generadores describen las conexiones entre los sumadores y los N estados del registro de entrada.

Donde gi se define como:

$$gi = (gi1, gi2, \ldots, giN)$$

$i = 1,2,\ldots,n0$
$k0 = 1$
N = longitud límite

Para representar el valor octal de un vector generador se utilizará la siguiente convención.

$$(gi) = (gi1, gi2, \ldots, giN)_8$$

Por ejemplo:

sea: $gi = (1\ 0\ 1)$
el equivalente octal es: $(gi) = 5$

REPRESENTACION MEDIANTE DIAGRAMA DE ESTADOS DE LOS CODIGOS CONVOLUCIONALES (n0,1) .

Esta representación gráfica describe las secuencias de salida del sistema de memoria finita observandose la dependencia entre la secuencia de entrada, la secuencia generada y el estado anterior del dispositivo.

Se define la memoria empleada por el codificador como:

$$L = N - 1$$

Suponiendo N=2, se tiene el estado Sl del codificador en el tiempo l como el contenido de la memoria en ese mismo instante.

$$\&l = (ul-1, ul-2)$$

Se podrá definir 2^L estados posibles.

El diagrama de estados queda definido por:

- Los estados por ciclo y su etiqueta

- Trayectoria continua si se envió un 0

- Trayectoria discontinua si se envió un 1

- Etiquetas de trayectoria representando los dígitos codificados de salida para cada transición.

Para el código con vectores generadores g1 = (100) = 4, g2 = (101) = 6, g3 = (111) = 7 se tiene el siguiente diagrama de estados:

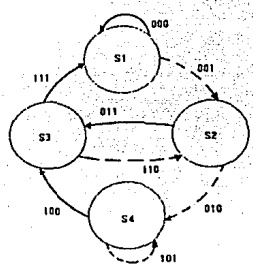


Fig.A3

La complejidad del manejo de este diagrama se incrementa con el crecimiento de L, normalmente se utiliza para L≤ 10.

Se puede aplicar estos conceptos para códigos (n0,k0) y longitud límite N.

Donde: la memoria requerida será, L = N - 1

el número de estados será, $S= 2^{k0L}$
y el número de trayectorias será,en el diagrrama de estados : ty = 2S

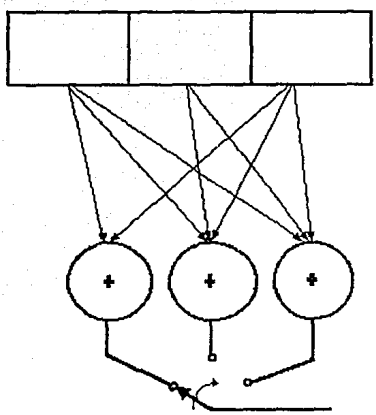
Donde , para cada estado se tienen 2 trayectorias de salida.

Las etiquetas de las trayectorias son secuencias de n0 dígitos.

Con lo anterior se realiza un ejemplo con los siguientes valores:

Para N=3, n0=3 y k0=1

Es un código (3,1) , con Rc= 1/3



g1 = (g11 g12 g13)

g2 = (g21 g22 g23)

g3 = (g31 g32 g33)

Fig.A4

Se tiene su correspondiente equivalente octal siguiendo con el mismo ejemplo pero con los vectores generadores

$$(g1) = (1\ 0\ 1)_8 = 5$$

$$(g2) = (1\ 1\ 1)_8 = 7$$

$$(g3) = (1\ 1\ 1)_8 = 7$$

se obtiene el siguiente diagrama de estados:

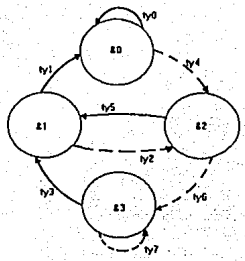


Fig.A5

Si N=3 la memoria requerida será:

$$L = N-1 = 3-2 = 2$$

El número de estados:

$$S = 2^2 = 4$$

y el número de trayectorias:

$$ty = 2S = 8$$

Realizando una tabla de transición de estados y sus respectivas salidas, agrupando primero para entrada cero, se tiene:

ENTRADA	ESTADO ANTERIOR	TRANSICION	ESTADO ACTUAL	SALIDA
	&0	ty0	&0	000
	&1	ty1	&0	111
0	&2	ty2	&1	011
	&3	ty3	&1	100
	&0	ty4	&2	111
	&1	ty5	&2	000
1	&2	ty6	&3	100
	&3	ty7	&3	011

Tabla A1

De donde se observa que los valores correspondientes a sus salidas asociadas son complementarias, ésto es :

$$sty0=\overline{sty4},\ sty1=\overline{sty5},\ sty2=\overline{sty6},\ sty3=\overline{sty7}$$

ESTRATEGIA DE CORRECCION DE ERRORES

Se propone la siguiente estrategia para la corrección de dos errores, que se introducen por efectos de ruido en el canal en los triples recibidas.

Definición del proceso de decodificación y corrección de error.

Se desarrollará el proceso de decodificación y corrección para un código de longitud límite con N=4, el registro de salida con no=3, a éste vector se le denominará triple y se supondrá que si se detecta una diferencia es que hay error debido a ruido en el canal.

Se tiene un dispositivo como receptor en el cual llega la información en forma serial y se acumulan los bits en el sistema de memoria de longitud N como se ve en la figura 1.

Se realiza una comparación entre los triples recibidas en contra de unas triples que se generan localmente llamadas triples supuestas.

Supóngase que se está realizando el proceso de decodificación hasta la j-ésima terna en el que no han ocurrido errores. En la terna j-ésima ocurre un error doble ó triple, lo que ocasiona que al comparar la terna recibida con la terna supuesta, se detecte una diferencia, o ninguna, pero las siguientes triples supuestas corresponden a una trayectoria del árbol equivocada, lo cual hará que , en los triples j+1,j+2,j+3,...j+N se detecten nuevamente diferencias entre las triples supuestas y las recibidas.

Se supone que en los triples recibidas después del j ya no ocurren errores en canal. Algunas triples recibidas en la que se detecte una diferencia con la terna supuesta, se decodificará erróneamente, a menos que se le sume un bit corrector. Un total de N bits correctores forma el vector corrector, el valor de estos bits son el resultado de las comparaciones entre los triples recibidas y los triples supuestas, cuando existen dos diferencias entre los triples recibidas y los triples supuestas, el valor del bit corrector es 1, de otra manera este valor es 0.

El resultado de sumar el vector corrector con los bits decodificados en las posiciones j+1,j+2,j+3,...,j+N, es el de trasladarse y recuperar la trayectoria del árbol adecuada.

Posteriormente se sigue realizando el proceso de decodificación normalmente.

LEMA 1

Existe un sólo vector corrector para cada código y se compone de un conjunto de bits

de longitud N.

Al intentar aplicar la anterior estrategia de decodificación se obtuvieron las siguientes conclusiones.

Para códigos que tienen por lo menos un vector generador que cumpla con la siguiente desigualdad, no es aplicable la estrategia de corrección. Debido a que para estos códigos, existen tres diferentes combinaciones para introducir dos errores en una terna, lo que implica tener dos trayectorias con igual distancia Hamming, y / o una trayectoria con distancia nula.

$$(gi) < S/2$$

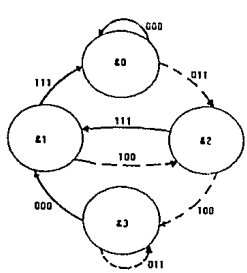
Debido a estas condiciones no se puede determinar un vector corrector único, ya que no existe sólo una estrategia o patrón de corrección.

Sea el siguiente ejemplo :

Un código, que tiene los siguientes vectores generadores g1=(0,1,0), g2=(1,1,1) y g3=(1,1,1)

Para este código, S=8 y (g1)=3 , (g2)=7 , (g3)=7

El diagrama de estados es el siguiente:



Si se envía la siguiente secuencia:

1, 0, 1, 1

Los triples se codifican como:

011 111 100 100

Fig.A6

Si se introduce un doble error en cualquier terna recibida y se intenta decodificar, comparando con los triples supuestas se obtiene las siguientes conclusiones.

Intruduciendo un doble error en la primera terna

011 111 100 100

101 111 100 100

La primera terna deberá compararse con las salidas del primer estado o sea los triples supuestas 000 ó 011, en la que observamos que se detecta dos diferencias y no se tiene puede decidir por ninguna de ellas.

Una de las otras dos combinaciones de doble error para la primer terna provoca la misma situación y en la última no se detecta el error en la terna recibida.

las tres combinaciones de doble error son las siguientes:

Para		0 1 1		
triples con doble error		0 0 0	1 0 1	1 1 0
Triples supuestas	000	- - -	X - X	X X -
	011	- X X	X X -	X - X
conclusión		decisión errónea	sin decisión	sin decisión

A1

X bit marcado con error
- bit sin error

Tabla A2

LEMA 2

Es indispensable tener entonces vectores generadores con:

$$(gi) \geq S/2 \text{ , para todo i}$$

a partir de este valor de los vectores generadores se generan los llamados códigos sistemáticos, se tienen códigos que desarrollan sólo 2 trayectorias con diferentes distancias de Hamming, lo que permite tomar decisiones y por lo tanto tener un sólo vector corrector para un código en particular.

El trabajo se concreta entonces a determinar el valor del vector corrector para cada código.
El rango de valores que el vector corrector Vc podrá tomar es:

$$S \leq (Vc) \leq (2S - 1)$$

(Vc) entero

Partiendo en dos el rango anterior, se tiene el intervalo (Vc)I

(Vc) puede tomar alguno de estos valores

$$(Vc)I = \{ S, S+1,..., (S\ (3/2) -1) \}$$

Y el intervalo (Vc)II

$$(Vc)II = \{ S(3/2),..., 2S-1 \}$$

LEMA 3

Si en alguno de los vectores generadores se repite su valor octal (gi), el vector corrector (Vc) tomará el mismo valor que se repitió. Si y sólo si, el valor octal del los vectores generadores repetidos está en el intervalo I.

Sea un ejemplo:

Donde:

g1 , g2 , g3 un código cualquiera

Con $k0 = 1$, $N = 3$:

$$(g1)= 5 , (g2)= 5 , (g3)= 7$$

La memoria requerida será: $L = N - 1 = 2$

El número de estados: S = 4

Como el valor octal de los vectores generadores repetidos (g1) y (g2) caen en el intervalo (Vc)I :

$$(g1) = (g2) = 5$$

el intervalo (Vc)I : $(Vc)\,I = \{ S , S+1, \ldots , (S(3/2) - 1) \}$

$$= \{ 4, 5 \}$$

Por lo tanto el valor octal de (Vc) es: $(Vc) = 5$

LEMA 4

Si para alguno de los vectores generadores se repite su valor octal (gi), y su valor octal que se repitió, se encuentra en el intervalo (Vc)II :

Donde $(Vc)II = (S (3/2) , \ldots , (2 S-1)$

El valor octal del vector generador será :

$$(Vc) = S + ((S(5/2) -1) - (gi))$$

Por ejemplo:

Sea:

g1 , g2 , g3 un código cualquiera

Con $k0 = 1$, $N = 4$:

$$(g1)= 15 , (g2)=14 , (g3)= 14$$

La memoria requerida será: $L = N - 1 = 3$

El número de estados: $S = 8$

Como el valor octal de los vectores generadores repetidos (g1) y (g2) caen en el intervalo (Vc)II.

$$(Vc)II = \{ S(3/2) , ... , (2S -1) \}$$

$$(Vc)II = \{ 12, 13, 14, 15 \}$$

entonces $(Vc) = S + ((S(5/2)-1)-(g1))$

$= 8 + ((8(5/2)-1)-(14)) = 13$

$(Vc) = 13$

LEMA 5

El orden de los vectores generadores no tiene trascendencia al determinar el valor del vector corrector.

Ejemplo:

Realizando un análisis general para códigos con N=3 y k0=1:

La memoria L=N-1=3-1=2, el número de estados $S = 2^2 = 4$ y se tienen ty=2S =2(4) = 8 trayectorias de entrada y salida.

En donde G es un código cualquiera, formado por los vectores generadores g1, g2, g3:

$$G = (g1, g2, g3)$$

Donde los valores que pueden tomar los vectores generadores están en el siguiente intervalo:

$$S \leq (gv) \leq 2S - 1$$

Para el ejemplo (g1), (g2) y (g3) toman los siguientes valores en el conjunto:

$$(g1) = (4 \text{ ó } 5 \text{ ó } 6 \text{ ó } 7)$$

$$(g2) = (4 \text{ ó } 5 \text{ ó } 6 \text{ ó } 7)$$

$$(g3) = (4 \text{ ó } 5 \text{ ó } 6 \text{ ó } 7)$$

Y G existe para cualquier combinación de (g1), (g2) y (g3).

La tabla A3 resume todas las posibilidades de combinación:

(g1)	4	5	6	7
(g2)	4 5 6 7	4 5 6 7	4 5 6 7	4 5 6 7
(g3)	4			
	5			
	6			
	7			

Tabla A3

Para la tabla A3 los valores de (g1) se escriben en un renglón sólo una vez, para (g2) se escriben en un renglón 4 veces y para (g3) en columna una vez.

Desarrollando todos los valores de (Vc) por los lemas 3 y 4 se tiene la tabla siguiente:

(g1)		4	5	6	7
(g2)		4 5 6 7	4 5 6 7	4 5 6 7	4 5 6 7
	4	4 4 4 4	4 5 * *	4 * 7 *	4 * * 6
	5	4 5 * *	5 5 5 5	* 5 7 *	* 5 * 6
(g3)	6	4 * 7 *	* 5 7 *	7 7 7 7	* * 7 6
	7	4 * * 6	* 5 * 6	* * 7 6	6 6 6 6

Tabla A4
(El *, indica que es no puede determinar
el valor con los lemas 3 y 4.)

Los valores faltantes de la tabla A5, se obtuvieron por medio de un programa que genera el vector corrector (Vc) para cada código.

Dicho programa parte con una secuencia de N bits, la que se codifica en triples, que después se modifica agregando dos diferencias (error debido al canal) en dos de los bits de los triples recibidos. Posteriormente se efectúa la decodificación y se acumulan las

diferencias (D.A.) entre los triples supuestos y las recibidas. Con los N bits decodificados y los N bits enviados se realiza una suma módulo 2 y el resultado de dicha operación da como resultado el vector corrector (Vc).

Para éste conjunto de códigos en la tabla 4, existe una función que relaciona el valor octal de los vectores generadores con el (Vc) vector corrector, en el caso de que no se repita alguno de los vectores generadores (asteriscos en la tabla 4).

La función sólo se cumple para éste conjunto de códigos, debido a que el conjunto de valores octales que puede tomar cada vector generador tiene cuatro elementos, que son 4,5,6 y 7. Como sólo hay tres vectores (g1,g2 y g3), un valor no se usa.

El siguiente lema da los valores de (Vc) faltantes en la tabla 4

LEMA 6: Para tal caso:

$$(Vc) = S\,(5/2) + 1 - St$$

Donde $St = S + S+1 + \ldots + 2S-1 - (g1) - (g2) - (g3)$

Sea por ejemplo:

g1,g2 y g3 los vectores generadores de un código con N=3 y S=4.

Donde g1 = (1 0 1), g2 = (1 1 1) y g3 = (1 1 0)

el valor octal de cada vector generador es:

$$(g1) = 5\,,\ (g2) = 7 \text{ y } (g3) = 6$$

El conjunto de valores que puede tomar (gi) es el siguiente:

$$\{ (gi) \} = \{\ S, S+1, \ldots, 2S-1 \}$$

$$\{ (gi) \} = \{\ 4, 5, 6, 7\ \}$$

$$St = (S + S+1 + \ldots + 2S-1 - (g1) - (g2) - (g3))$$

$$St = (4 + 5 + 6 + 7 - 5 - 7 - 6) = 4$$

$$(Vc) = S\,(5/2) + 1 - St$$

$$(Vc) = 4\,(5/2) + 1 - 4 = 7$$

En la tabla A5 se marca los valores que se obtuvieron con la función anterior.

(g1)		4	5	6	7
(g2)		4 5 6 7	4 5 6 7	4 5 6 7	4 5 6 7
	4	4 4 4 4	4 5 4 5	4 4 7 6	4 5 6 6
	5	4 5 4 5	5 5 5 5	4 5 7 6	5 5 7 6
(g3)	6	4 4 7 6	4 5 7 7	7 7 7 7	6 7 7 6
	7	4 5 6 6	5 5 7 6	6 7 7 6	6 6 6 6

Tabla A5
(Vc) = función [(g1), (g(2), (g3)]

Los valores calculados con este lema, coinciden con los calculados mediante el programa.

Tomando los vectores generadores g1, g2 y g3 como coordenadas cartesianas de un espacio vectorial, se tiene lo siguiente:

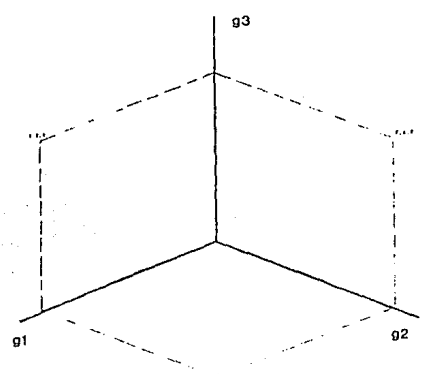


Fig. A7

Graficando cada punto de la tabla A5 en el espacio vectorial se tiene la siguiente figura:

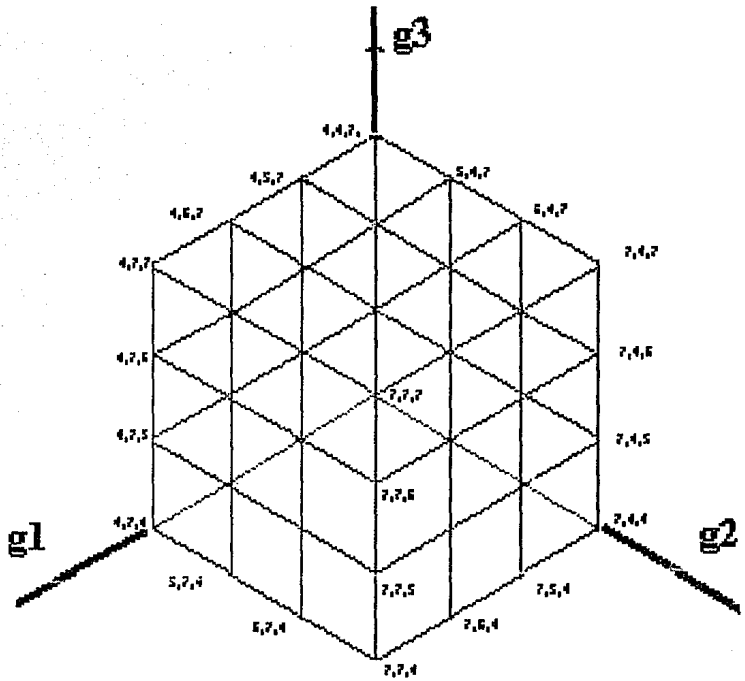
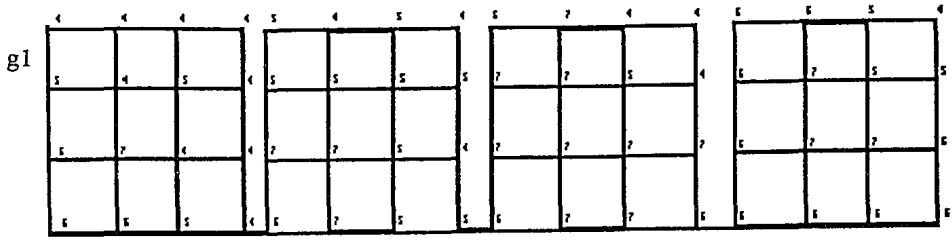


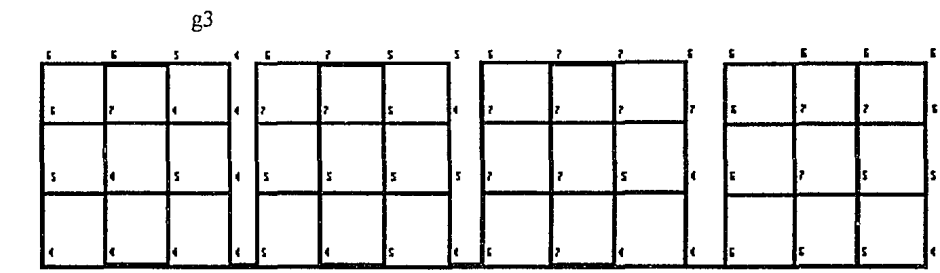
Fig.A8

Para aclarar un poco la anterior figura, se desarrollará la figura A8 por planos paralelos a los eje y asignando su (Vc) asociado, se tiene lo siguiente:

Para los planos paralelos a los ejes g1 y g2 :



Para los planos paralelos a los ejes g1 y g3:



Para los planos paralelos a los ejes g2 y g3:

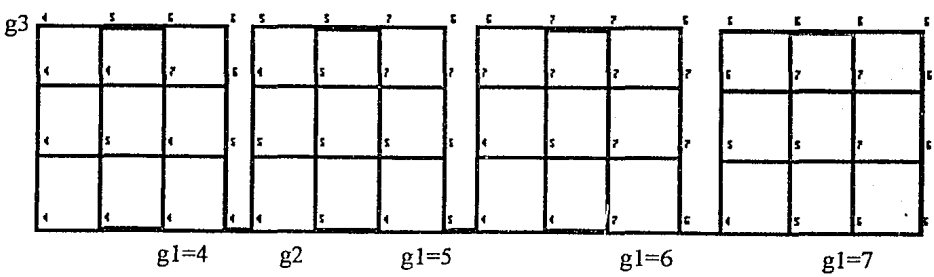


Fig.A9

Los valores indicados en los cuadros son (Vc)

Se observa que existen, en forma colineal los códigos con iguales (Vc), que son en realidad los códigos en los que por lo menos dos de sus vectores generadores son iguales.

Agregando a la figura A8 , planos a 45° P1, P2 y P3.
P1 tiene un ángulo de 45° con respecto del plano g1,g2 y 45° con el plano g2,g3.
P2 tiene un ángulo de 45° con respecto del plano g1,g3 y 45° con el plano g1,g2.
P3 tiene un ángulo de 45° con respecto del plano g1,g3y 45° con el plano g3,g2.
Realizando la figura 1 pero ahora con los planos que intersectan las aristas de cubo completo tenemos que:

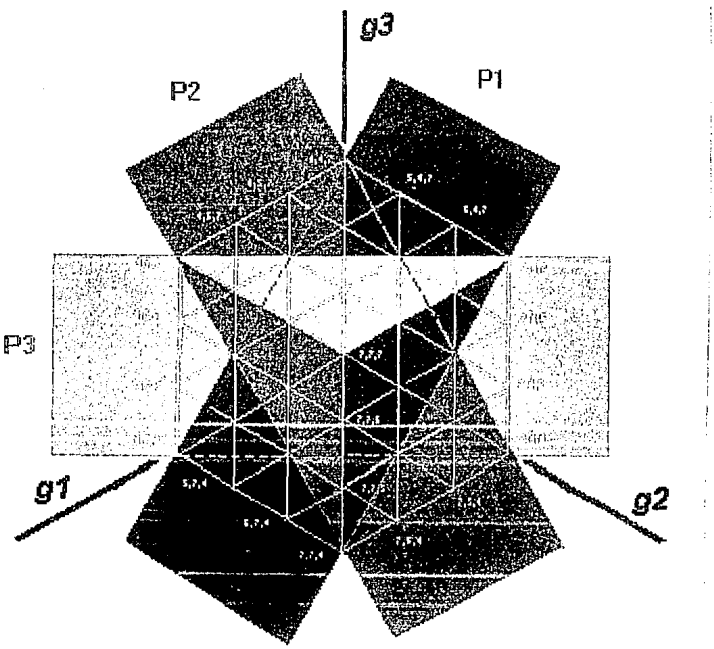
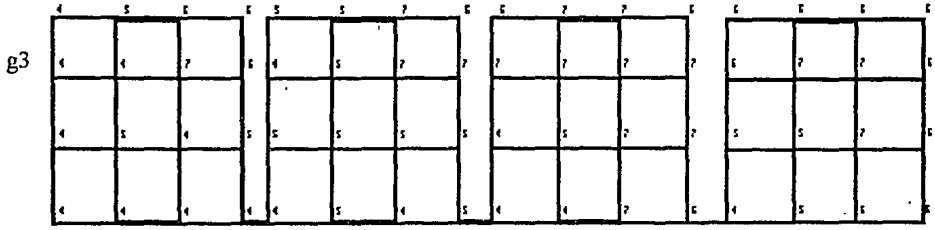


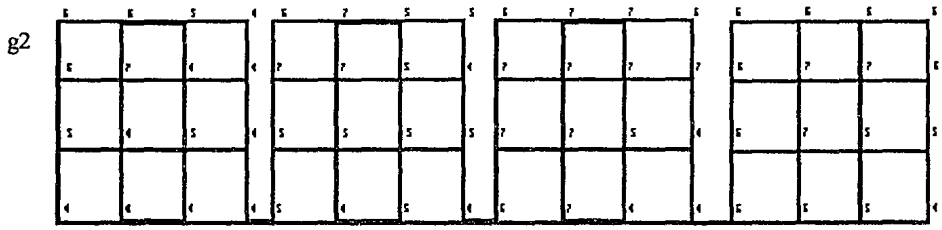
Fig. A12

Se observa que en las aristas de la figura A12 se tienen códigos con valores repetidos (6, 6, 5, 4) de (Vc) . Pasando planos que contengan a estos vectores se describen planos a 45° con respecto a los ejes g1, g2 y g3.
Revisando la intersección de planos paralelos a los anteriores, se nota que contienen códigos que también repite sus (Vc).

Para los planos que cortan a 45° los ejes g3 y g2:



Para los planos que cortan a 45° los ejes g1 y g2:



Para los planos que cortan a 45° los ejes g1 y g3:

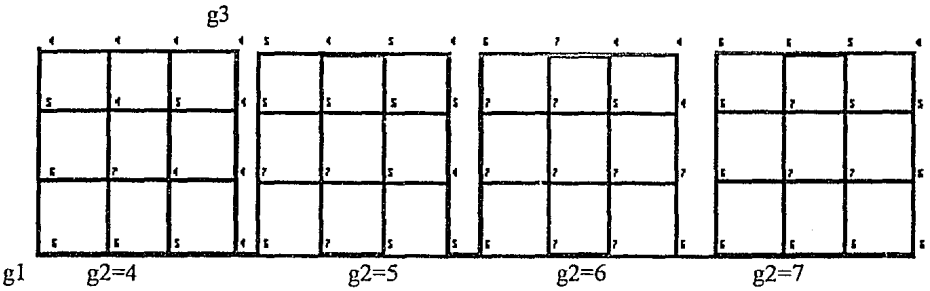


Fig.A13

Los valores indicadosen los cuadros son (Vc)