7
2 ej.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"SIMULACION DEL FORMATO Y CODIFICACION DE LA FUENTE DE UN SISTEMA DE COMUNICACIONES DIGITALES"

TESIS

QUE PARA OBTENER EL TITULO DE

Ingeniero Mecánico Electricista e Ingeniero en Computación

PRESENTAN:

BARRON TREJO ROBERTO

ROSAS RAMIREZ C. SERGIO

VILLAMIL CORDOVA SALVADOR

VILLARREAL RODRIGUEZ MARIO

DIRECTOR: M. EN I. CAUPOLICAN MUÑOZ GAMBOA

MEXICO, D. F. ABRIL DE 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

1 Introducción 1

2 Esquema General de un Sistema de Comunicaciones . . . 3

3 Simulación del Formato y Codificación de la Fuente . . 11

3.1 Muestreo 11

3.2 Cuantización 13

3.3 Modulación por Codificación de Pulsos 18

3.4 Modulación por Codificación de Pulsos Diferencial 22

3.5 Modulación Delta 24

3.6 Modulación Delta Doble Paso 27

3.7 Codificación de Caracteres 28

3.8 Señalización de Respuesta Parcial 30

4 Programación de los procesos 38

5 Resultados 64

6 Conclusiones y alternativas 77

7 Bibliografía 78

Apéndice A Programa Fuente

Apéndice B Guía de operación

1 Introducción

Para transmitir información por medios eléctricos y electrónicos, se emplean principalmente tres sistemas de comunicación: analógicos, digitales e híbridos.

Un sistema de comunicación analógico opera con señales de naturaleza analógica. Un sistema de comunicación digital, opera con señales de naturaleza digital, como las producidas por una computadora. También opera con señales que inicialmente son de naturaleza analógica, como puede ser la señal de voz, pero se requiere un proceso previo de conversion analógica/digital. Las señales digitales se cuantifican posteriormente en un número predeterminado de niveles, los cuales son codificados y transmitidos. Los sistemas de comunicación híbridos utilizan esquemas de transmisión digital para transmitir información analógica.

La primera propuesta para convertir señales analógicas en un formato digital fue hecha por H.Reeves en 1938, por medio de tubos de vacío; sin embargo, no fue hasta la aparición del transistor en 1948, cuando se vislumbraron las ventajas reales del procesamiento digital, las cuales se consolidaron con el advenimiento de los circuitos integrados.

El presente trabajo contempla solamente los sistemas de comunicaciones digitales en sus fases de formato y codificación de la fuente. El formato es el primer proceso que se realiza sobre la información a transmitir, cuya función es adecuar la señal de entrada a una forma digital. La codificación de la fuente tiene como función optimizar esta representación digital, al comprimir la información y eliminar de ésta la redundancia, mediante distintas técnicas de codificación.

La simulación es un procedimiento en el cual un sistema es sustituido por otro sistema que imita ciertos aspectos importantes. Entre las principales razones para utilizar la simulación se encuentran: el sistema real puede no existir, hecho que permite adelantar decisiones sobre el diseño final; trabajar con sistemas reales puede resultar costoso debido a posibles riesgos en la operación, así como consideraciones físicas irrealizables del mismo. Además, la manera más económica y conveniente de realizar este procedimiento es por medio de una computadora.

Las fases del formato y codificación de la fuente de un sistema de comunicaciones digitales, se simulan en este trabajo mediante un programa de computadora que permite procesar señales almacenadas en archivos de disco. Los cambios que sufren las señales en su paso por los procesos de simulación implementados, se pueden observar en el monitor de

la computadora, para su interpretación y análisis. Cabe señalar que en el receptor no se considera el ruido, la distorsión y la interferencia.

Los procesos simulados en ambas fases son: Muestreo, Cuantización, Modulación por Codificación de Pulsos (MCP), Modulación por Codificación de Pulsos Diferencial (MCPD), Modulación Delta (MD), Modulación Delta Doble Paso (MDDP), Codificación de Caracteres y Señalización de Respuesta Parcial, tanto en la fuente como en el destino. Dentro de la Modulación por Codificación de Pulsos, se simulan en la etapa transmisora los códigos RZ, RB, AMI(BRZ), Fase Partida (Manchester), Fase Partida (marca), NRZ(L), NRZ(M), NRZ(S) y Retraso de Modulación (Miller). El programa fue desarrollado en el lenguaje de programación PASCAL, y puede utilizarse en cualquier microcomputadora del tipo personal compatible con IBM-PC, facilitando así su disponibilidad de uso.

El presente trabajo pretende apoyar el proceso de aprendizaje en el área de comunicaciones, específicamente en el laboratorio de Comunicaciones Digitales que se imparte en la División de Ingeniería Mecánica y Eléctrica de la Facultad de Ingeniería.

2 Esquema General de un Sistema de Comunicaciones

Un sistema de comunicaciones consta de las operaciones que proporcionan el enlace para la información entre la fuente y el destino. Su objetivo principal es el de proporcionar una réplica aceptable de la información en el destino. En el caso de las comunicaciones digitales, estas operaciones se realizan por medios eléctricos y electrónicos que utilizan técnicas de procesamiento digital.

En un sistema de comunicaciones, se distinguen tres elementos en forma general:

Las funciones principales que realiza la Fuente son: amplificación, muestreo, codificación de la fuente, modulación, filtrado y codificación del canal.

El canal de transmisión tiene como función enlazar el transmisor y el receptor. Este puede ser, por ejemplo: par trenzado, cable telefónico, cable coaxial, fibra óptica o una onda radiada al espacio.

Las funciones principales que realiza el Destino son: decodificación del canal, filtrado, demodulación, decodificación de la fuente y amplificación.

En la actualidad, las técnicas digitales son ampliamente utilizadas para comunicaciones, control de procesos y procesamiento de datos. Las ventajas que ofrecen son:

- Permiten la transmisión de información a mayores distancias y con niveles de potencia menores que las comunicaciones analógicas(en condiciones equivalentes de calidad), pudiéndose regenerar con probabilidad de error mínima.

- La información en forma digital puede ser procesada por computadoras.

- La transmisión de información por medios sofisticados de comunicación, tales como rayos láser, fibras ópticas se puede realizar de mejor manera en forma digital.

- El ruido no se acumula en los repetidores, volviéndose una consideración secundaria para efectos de diseño.

- Se pueden tolerar niveles de ruido considerablemente mayores.

La principal desventaja, es el requerimiento de un mayor ancho de banda; así como el ruido de cuantización, aunque este se reduce notablemente al codificar con un mayor número de bits; por lo que uno de los objetivos en el procesamiento de la información es tratar de compactarla mediante cualquiera de las técnicas de codificación existentes.

La Figura 2.1 ilustra el diagrama de bloques de un Sistema de Comunicaciones Digitales (SCD).

Los bloques de la izquierda representan etapas de procesamiento desde la fuente de información hacia la antena del transmisor, mientras que los bloques de la derecha representan la función inversa. Los bloques indispensables de un sistema de comunicaciones son el formato, la modulación, la demodulación, el transmisor y el receptor.

A continuación se explica la función de cada uno de los bloques que forman un Sistema de Comunicaciones Digitales:

- Formato.

Es el proceso que se realiza a las señales de entrada (ya sean analógicas o texto) para que éstas adquieran una forma digital adecuada a la transmisión. Normalmente el proceso consiste en asignarle un valor digital a la señal analógica posterior al muestreo al que se le somete, en tanto que a la entrada tipo texto se le asigna un código previamente definido.

En el caso de la transmisión digital, la información a transmitir debe ser adecuada a símbolos digitales, por lo que, si es de naturaleza analógica, deberá primero ser muestreada a una velocidad de por lo menos 2 veces la frecuencia máxima contenida en la señal (frecuencia de Nyquist) y representarse después con símbolos binarios. Es decir, que el formato involucra cualquier operación que transforma datos en símbolos digitales o una adecuación para su procesamiento digital.

Los procesos de formato son: muestreo y cuantización.

El proceso de muestro se realiza mediante la conmutación de la señal a intervalos periódicos de tiempo. El proceso de cuantización se realiza mediante la conversión de una señal analógica a una serie de pulsos de amplitud discreta, de acuerdo a determinado código que usualmente es binario.

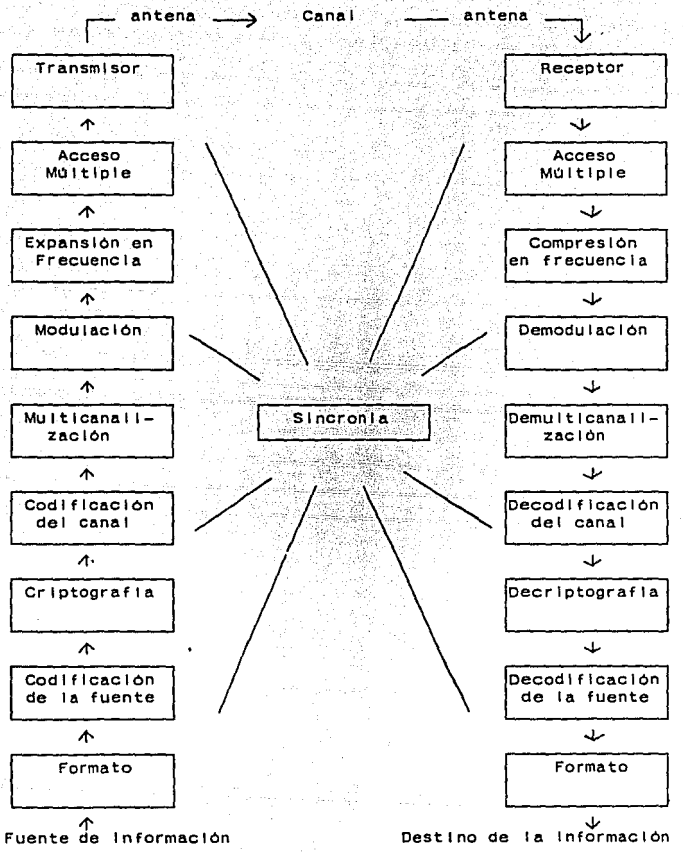


Fig. 2.1 Esquema General de un Sistema de Comunicaciones Digitales

- Codificación de la fuente.

Una característica inherente a la información transmitida se refiere a la redundancia, consistente en la existencia de símbolos no igualmente probables o símbolos no estadísticamente independientes. Esta redundancia en la información es la base para que se pueda reducir la velocidad en el número de datos transmitidos, o en otras palabras, se puedan comprimir los datos. Es decir, que la codificación de la fuente involucra cualquier operación en la compresión de los datos. Se hace notar que esta compresión no implica pérdida de información.

Los procesos de codificación son: Modulación por Codificación de Pulsos (MCP), Modulación por Codificación de Pulsos Diferencial (MCPD), Modulación Delta, Modulación Delta Doble Paso, Codificación de Caracteres y Señalización de Respuesta Parcial.

- Criptografía.

Las técnicas para criptografiar información evitan la posibilidad de que personas no autorizadas extraigan información o introduzcan información al canal, logrando así privacidad y autenticidad en la información transmitida y recibida.

La criptografía se realiza aplicando una función de transformación al mensaje, la que es conocida sólo por los usuarios. Con ello la señal queda lista para enviarse a través de un canal público.

El receptor autorizado elimina la criptografía aplicando la función de transformación inversa. Cualquier persona no autorizada, sólo podrá efectuar una estimación del mensaje a partir del texto criptografiado si quiere descubrir la información.

Existen 2 esquemas para criptografiar información digital:

- Criptografía por bloques.
- Criptografía por flujo de datos.

En la Criptografía por Bloques, el texto es segmentado en bloques, los cuales son criptografiados en forma independiente. Una característica de ésta técnica, es que al cambiar un bit por lo menos, equivale a cambiar aproximadamente un 50 % de bits en el texto criptografiado, lo cual implica que el mensaje aparece fuertemente alterado en el texto criptografiado. Se utilizan dos técnicas para criptografiar los bloques de información.

Substitución: Donde n bits se representan como alguno de 2^n caracteres, este se substituye por otro caracter y se regresa criptografiado en n bits de salida.

Permutación: Los datos de entrada son simplemente arreglados o permutados en otro orden.

Shannon introdujo la substitución y la permutación, aunque sugirió la combinación de ambas técnicas, para lograr una criptografía mejor sobre la información.

La criptografía por flujo de datos no trata los símbolos de entrada en forma independiente ya que la criptografía depende del estado interno del dispositivo implementado (registro de corrimiento). Después que el símbolo es criptografiado el dispositivo cambia de estado de acuerdo a determinada regla.

- Codificación del Canal.

La codificación del canal se refiere a la transformación de datos, desarrollada después de la codificación de la fuente pero antes de la modulación. Su función es transformar la fuente de bits en un canal de bits.

La codificación del canal puede, para una velocidad de datos dada: disminuir la probabilidad de error a expensas de la potencia o el ancho de banda; reducir los requerimientos de ancho de banda a expensas de la potencia o la probabilidad de error; reducir los requerimientos de potencia a expensas del ancho de banda o la probabilidad de error.

La codificación del canal esta dividida en dos grupos, la codificación de la forma de onda, y las secuencias estructuradas. La señalización M-aria puede considerarse como un procedimiento de codificación de la forma de onda. Las secuencias estructuradas, es un método que introduce redundancia en la fuente de datos, de tal manera que pueden ser identificados errores en la transmisión.

- Multicanalización y Acceso Múltiple.

Los sistemas de comunicación usados para las comunicaciones por satélite proporcionan una aplicación tecnológicamente significativa tanto de las técnicas de modulación como de banda base. Además, debido a las grandes distancias a la que están situados los satélites, han tenido que desarrollarse nuevos métodos para lograr la entrada al sistema del satélite. Estos se llaman Métodos de Acceso.

Varios modos de acceso al satélite se encuentran en uso actualmente. El procedimiento más común, llamado Acceso Múltiple por División de Frecuencia (FDMA), es similar a la

Multicanalización por División de Frecuencia (FDM). Esta es una técnica en la que se combinan varias señales independientes separadas en frecuencia de manera que pueden transmitirse por un canal común sin interferirse.

Las estaciones terrestres tienen asignaciones de canales específicos y deben usar estas frecuencias al transmitir o al recibir señales. Una estación terrestre transmitirá una señal modulada en frecuencia, usando una portadora específica del intervalo de 6 GHz (por ejemplo una banda del satélite). La señal moduladora o de banda base para la señal transmisora de FM consta de la suma de un cierto número de canales de 4 kHz (telefónicos) multicanalizados por división de frecuencia (FDM). Todas las estaciones terrestres tienen sus portadoras de FM espaciales, de modo que las señales de FM de cada una de ellas ocupen bandas adyacentes, cubriendo la señal compuesta de FDMA en el satélite la banda completa de 36 MHz del transponder. El procedimiento completo de acceso se conoce como FDM/FM/FDMA.

Recientemente se ha introducido una nueva técnica para manejar situaciones de poco tráfico, llamada Acceso Múltiple de Asignación de Demanda (DAMA). En este caso las bandas de frecuencia se asignan, según la demanda, de entre una bolsa de bandas disponibles, a las estaciones que las estén requiriendo. Al contrario de FDMA, donde los canales desocupados van sin usarse, en la técnica de asignación por demanda todos los canales están disponibles para todas las estaciones, de manera que existe una mayor probabilidad de que se usen (Cuando se trata de una situación de tráfico denso, esto no es necesario, puesto que un canal asignado rara vez se encontrará ocioso). La tercera técnica de acceso usada en las comunicaciones por satélite es la de Acceso Múltiple por División de Tiempo, análoga a TDM, donde las diversas estaciones tienen asignados espacios de tiempo durante los cuales pueden transmitir.

- Modulación.

Antes de transmitir por medio de un canal de comunicación una señal portadora de información, se utiliza algún tipo de procedimiento de modulación para generar una señal que pueda ser fácilmente adaptada al canal.

En general la modulación es el proceso por el cual algunas características de la forma de onda de la portadora se varían en una correspondencia uno a uno con la señal que contiene la información, que se conoce como mensaje. Dicho de otra manera, modulación es la variación sistemática de alguna característica de la onda portadora, como la frecuencia o la fase, la amplitud, o una adecuada combinación de ellas, y esto va a ser de acuerdo con la función de la señal mensaje. Algunas de las razones existentes que se consideran importantes para modular una señal son para:

- facilidad de radiación
- asignación de canales
- transmisión múltiplex
- reducción de ruido e interferencia
- superar las limitaciones del equipo para una transmisión eficiente.

Así también, dependiendo de las características de la señal mensaje, del canal, de la respuesta que se desea obtener del sistema total de comunicaciones, del uso de los datos transmitidos y de factores económicos, es como se aplicarán las diferentes técnicas existentes para modular una señal, las cuales ayudarán a seleccionar el tipo de modulación que deberá aplicarse.

Los dos tipos básicos de modulación analógica son: la modulación de onda continua y la modulación por pulsos. Hay dos tipos básicos de modulación de onda continua: modulación angular y modulación lineal.

En la modulación por onda continua un parámetro de la portadora varía en proporción con la señal mensaje o moduladora, de tal manera que deberá existir una correspondencia de uno a uno entre el parámetro y la señal mensaje. Cuando la amplitud se encuentra linealmente relacionada con la señal moduladora, el resultado será una modulación lineal. Cuando la fase y la frecuencia están linealmente relacionados con la señal moduladora en forma colectiva se conoce como modulación angular.

Existen tres tipos de modulación analógica por pulsos: Modulación por Amplitud de Pulsos (PAM), Modulación por Ancho de Pulsos (PWM) y Modulación por Posición de Pulsos (PPM). La modulación por pulsos es el resultado que se obtiene cuando se muestrea la señal y se usa una portadora de tren de pulsos. Un parámetro de cada pulso se varía en una correspondencia uno a uno con el valor de cada muestra.

Debido al avance que existe en la tecnología de las comunicaciones, se hace común usar frecuentemente técnicas de modulación digital, ya que al aplicar estas técnicas se logra minimizar algunos efectos, tales como el ruido y la interferencia.

Como ya se mencionó antes la importancia que tienen las comunicaciones digitales, solo nos dirigiremos a algunas técnicas básicas de modulación digital. Se tiene que estas se encuentran divididas en dos grupos: modulación coherente y modulación no coherente.

Se llama coherente al sistema digital, si está disponible una refencia local de demodulación, que esté en fase con la portadora transmitida. Si no sucede así al sistema se le llama no coherente.

Dentro de los sistemas coherentes se encuentran, la Manipulación por Corrimiento de Fase (PSK), Manipulación por Corrimiento de Frecuencia (FSK), Manipulación por Corrimiento de Amplitud (ASK), y los llamados híbridos: Manipulación por Encendido y Apagado (OOK) y Manipulación de Mínima Variación (MSK). Los no coherentes son: Manipulación de Variación de Fase Diferencial (DPSK), y Manipulación por Variación de Frecuencia.

- Expansión en frecuencia.

Las técnicas de expansión en frecuencia permiten que múltiples señales ocupen el mismo ancho de banda de radio frecuencia, sin que exista interferencia entre ellas.

Hay dos técnicas comunes para expansión en frecuencia. La primera llamada secuenciamiento directo, multiplica los datos de entrada con una secuencia binaria pseudoaleatoria. La segunda utiliza una portadora salta-frecuencias, la cual permanece durante determinado tiempo en una frecuencia fija y salta a otra frecuencia en otro instante.

- Sincronía.

Puede ser definida como la alineación de escalas de tiempo de procesos espacialmente separados, siendo para los sistemas digitales tanto en tiempo como en frecuencia. Existen principalmente cuatro esquemas de sincronía:

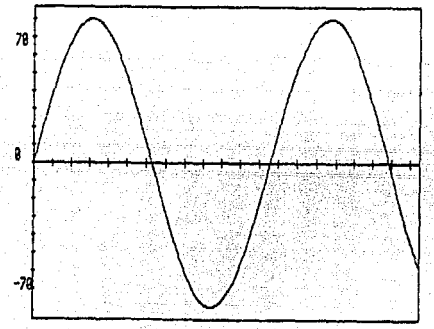
- Sincronía de portadora
- Sincronía por símbolo
- Sincronía por marco
- Sincronía por red

3 Simulación del Formato y Codificación de la Fuente

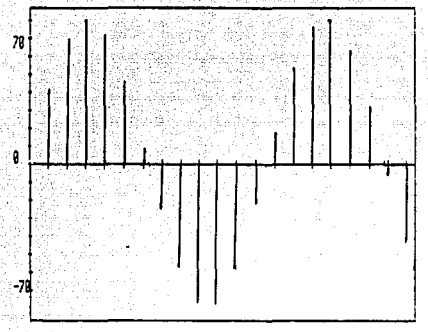
Se presenta a continuación la descripción teórica de cada proceso simulado en el presente trabajo, la cual se utilizó como base en la realización de las rutinas correspondientes.

3.1 Muestreo

Una operación básica para el diseño de todos los sistemas de modulación es el proceso de muestreo, por medio del cual una señal analógica se convierte en una sucesión correspondiente de números que por lo general, se encuentran espaciados uniformemente en el tiempo. Para que este procedimiento tenga utilidad práctica, es necesario que se elija el índice del muestreo en forma apropiada, de tal manera que esta sucesión de números defina unívocamente a la señal analógica original. La Figura 3.1 ilustra el proceso de muestreo.



a) Señal analógica



b) Señal muestreada instantáneamente

Fig. 3.1 Proceso de muestreo

El proceso de muestrear uniformemente una señal en el dominio del tiempo produce un espectro periódico en el dominio de la frecuencia, con un periodo igual a la razón de muestreo. También debe notarse que para que no haya interferencia, la razón de muestreo no debe ser menor al índice de Nyquist.

En la práctica, el muestreo de una señal analógica se lleva a cabo por medio de circuitos transistorizados de conmutación de alta velocidad. De acuerdo con esto, se encuentra que la forma de onda muestreada resultante se desvía de la forma ideal de muestreo instantáneo, debido a que la operación de un circuito físico de conmutación, aunque rápido, requiere de un intervalo de tiempo diferente de cero. Además, a menudo se encuentra que las muestras de una señal analógica son alargadas intencionalmente por conveniencia en la instrumentación y en la transmisión.

3.2 Cuantización

En la transmisión de información, existen señales de comunicaciones que son digitales en su origen, sin embargo, otras señales son analógicas. Para transmitir este tipo de señales, deben ser muestreadas en forma periódica y posteriormente convertidas a muestras de amplitud discreta. La cuantización es la transformación de una señal analógica a digital. Este proceso le convierte en una serie de pulsos rectangulares, que permite a cada muestra ser expresada como un nivel de un número finito de niveles predeterminados, cada uno de los cuales puede ser expresado por un símbolo digital. Después de la cuantización, la forma de onda analógica puede ser recuperada, pero no con precisión a causa del error de cuantización. El mejoramiento de la fidelidad de reconstrucción de la forma de onda analógica se logra incrementando el número de niveles de cuantización, lo cual requiere incrementar el ancho de banda de la transmisión.

La Figura 3.2 muestra el diagrama de bloques en el transmisor de un sistema de comunicaciones de Modulación por Codificación de Pulsos (MPC), del cual forma parte la cuantización. A la entrada del transmisor se tiene una señal analógica g(t) limitada en banda y con frecuencia máxima en su espectro de fc Hz. Esta señal se muestrea a una frecuencia de fm Hz y se transforma así en la señal gm(t), formada por una serie de pulsos rectangulares cuyas amplitudes son las mismas que las de g(t) en los instantes de muestreo.

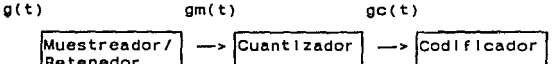


Fig. 3.2 Transmisor de un sistema MPC

Se requiere ahora que el número de posibles alturas de pulsos sea finito, para poder asignar a cada una un símbolo de un código o alfabeto finito (codificador). Se hace para esto una partición de un intervalo, que contenga todas las posibles amplitudes de g(t) y se asigna a cada amplitud de gm(t) el elemento de dicha partición más cercano en valor (redondeo u otro criterio). A esta operación se le denomina cuantización y a la señal gc(t) obtenida se le llama señal cuantizada.

Después la señal gc(t) se codifica, modula y es enviada a través de un canal de transmisión. En el receptor se efectúan operaciones inversas a las del transmisor. La señal recibida se demodula y decodifica obteniéndose de nuevo gc(t) (suponiendo un canal sin ruido), la cual se hace pasar a través de un filtro pasabajas y se obtiene así una aproximación $\hat{g}(t)$ de g(t).

La cuantización puede ser:

a) Uniforme, si el espaciamiento entre niveles es uniforme.
b) No uniforme, si el espaciamiento entre niveles es no uniforme.

Cuantización Uniforme.

Considérese una señal f(t) cuyo valor medio es cero, es decir, que no tiene componente de C.D. como se muestra en la Figura 3.3.

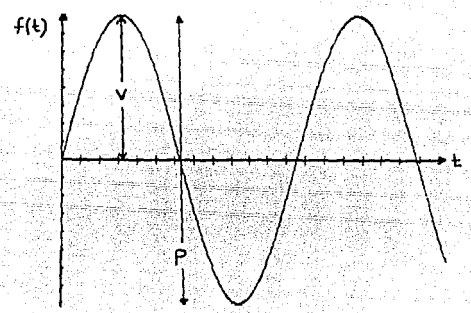


Fig. 3.3 Señal continua f(t)

donde:

P - Máxima excursión de la señal entre valores negativos y positivos.
V - Máxima excursión positiva o negativa de la señal.

Supóngase que la señal es muestreada y que las muestras se introducen a un cuantizador. Si en el cuantizador se utiliza un total de M niveles de cuantización espaciados uniformemente una cantidad a, entonces:

$$a = \frac{2V}{M} = \frac{P}{M}$$

A este espaciamiento de niveles adyacentes se le denomina escalón cuántico. En un sistema MPC binario, el número de niveles de cuantización M esta dado por:

$$M = 2^{m}$$

donde m es el número de bits que forman cada palabra de código, por lo tanto M es una cantidad par. La distribución de los niveles de cuantización depende del diseño del cuantizador.

Cuantización No Uniforme.

En ocasiones los valores de la señal de entrada al cuantizador no se presentan con la misma probabilidad dentro de un cierto intervalo, sino que tienden a ocurrir en una parte de éste, normalmente los valores inferiores. Si se cuantizara uniformemente este tipo de señales, seguramente se perderían muchos de estos valores, debido a que se les asignaría un mismo nivel. Para aprovechar mejor al cuantizador convendría tener un mayor número de niveles de decisión en el intervalo donde tengan mayor probabilidad de ocurrencia, y menor número fuera de éste. Desde un punto de vista probabilístico, debe amoldarse la característica del cuantizador a la Función Densidad de Probabilidad (FDP) de la señal de entrada. En este caso el cuantizador resultante será no-uniforme si la FDP es no uniforme. Sin embargo implementar un cuantizador no uniforme es mas difícil que implementar uno uniforme; en particular se muestra a continuación un sistema que permite implementar al primero usando el segundo.

Un cuantizador no uniforme puede modelarse por medio de una no-linealidad denotada K(x) y llamada característica del compresor, seguida de un cuantizador uniforme y por último la no-linealidad inversa $K^{-1}(x)$, llamada característica del expansor. Este sistema recibe el nombre de sistema compansor y se muestra en la Figura 3.4.

K(x) tiene la función de ampliar el intervalo de valores pequeños de x y disminuir los mayores. Los valores obtenidos se convierten después en la entrada al cuantizador uniforme, el cual, debido a la no linealidad de K(x), tomará más en cuenta a los valores pequeños de |x|. Una vez realizada la cuantización uniforme, en el receptor, pasa finalmente a $K^{-1}(x)$, la cual regresará los intervalos uniformes a su tamaño original.

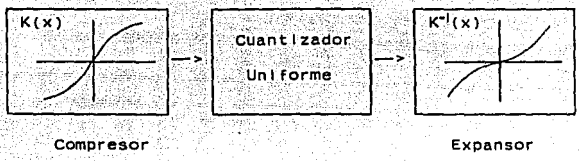


Fig. 3.4 Sistema compansor

En la práctica puede ser difícil implementar la función K(x). Debido a esto, en ocasiones se transforma en una curva seccionalmente lineal, la cual se aproxima más a la original mientras más segmentos rectilíneos se utilicen. A esta característica se le llama piezolineal.

Por otra parte, al diseñar un cuantizador se desea que éste fuera muy general, en el sentido de que cuantizara eficientemente al mayor número de señales distintas. Dado que cada señal puede tener asociada una FDP distinta, el cuantizador deberá ser "insensible" dentro de cierto margen, a las variaciones de los parámetros de una determinada familia de FDP.

La característica de compresión típica de un cuantizador no-uniforme, tiene la forma logarítmica, y dos formas comunes de compresión son la ley μ y la ley A.

La curva de la ley μ esta dada por:

$$y(x) = \frac{\ln(1 + \mu x)}{\ln(1 + \mu)} \qquad 0 \leq x \leq 1$$

donde μ es el factor de compresión.

La curva de la ley A es:

$$y(x) = \begin{cases} \dfrac{1 + \ln Ax}{1 + \ln A} & 1/A \leq x \leq 1 \\ \dfrac{Ax}{1 + \ln A} & 0 \leq x \leq 1/A \end{cases}$$

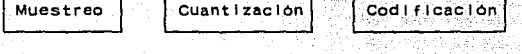
donde x es el valor normalizado con relación a la máxima señal de entrada, para ambas leyes.

Los sistemas MPC en E.U., Canadá y Japón, utilizan la característica de compresión μ, donde un valor típico de μ es 255 y se utiliza un versión piezolineal de 15 segmentos. La ley A se utiliza en sistemas MPC de telefonía europeos, y un valor típico es A = 87.6, con una versión piezolineal de 13 segmentos.

3.3 Modulación por Codificación de Pulsos

La Modulación por Codificación de Pulsos (MPC) es una técnica en la cual una onda analógica es transmitida de un modo digital equivalente.

Para generar una señal MPC se requieren 3 procesos:



Su mayor desventaja radica en el hecho de requerir un mayor ancho de banda.

Por ejemplo,en el caso de la industria telefónica se ha establecido la siguiente estandarización. Para una señal de voz de 4000 Hz(en realidad es menor), muestreada a razón de 8000 muestras por segundo, con 8 bits se requieren 64000 bps, los cuales implican 16 veces el ancho de banda de su equivalente analógico(4000 Hz).

La codificación consiste en convertir la muestras cuantizadas en grupos de pulsos(2 niveles) de amplitud fija. EL resultado de aplicar la codificación es una secuencia de bits, los cuales se pueden aplicar a la línea con o sin pasos de modulación adicionales, dependiendo del sistema de comunicaciones que se emplee.

Su funcionalidad está basada en el teorema del muestreo. Tiene aplicación en la transmisión de voz, en donde existe mayor probabilidad de encontrar señales de amplitud pequeña que señales de amplitud grande.

Se pueden elegir diferentes formas de representación en código binario, antes de introducir la señal al modulador. Los formatos de codificación más importantes, que se ilustran en la Figura 3.5 son:

RZ (Regreso a cero) - El "0" se representa como un nivel igual al nivel de referencia en todo el intervalo de bit. El "1" se representa como un cambio a nivel "alto" en la primera mitad del intervalo de bit y un cambio al nivel de referencia en la mitad restante.

RB(Regreso a polarización) - El "0" se representa como un cambio al nivel "bajo" en la primera mitad del intervalo de bit y un cambio al nivel de polarización en la mitad restante. El "1" se representa como un cambio al nivel "alto" en la primera mitad del intervalo de bit y un regreso al nivel de

polarización en la mitad restante. El nivel de polarización mencionado puede ser diferente de cero o estar entre los valores del nivel "alto" y "bajo".

AMI - El "0" se representa como un nivel igual al nivel de referencia. El "1" se representa como nivel "alto" y "bajo" en forma alternada durante la primera mitad del intervalo de bit y un regreso al nivel de referencia en la mitad restante. Se le conoce también como una representación bipolar regreso a cero (BRZ).

Fase partida (Manchester) - El "0" se representa como el nivel de referencia durante la primera mitad del intervalo de bit y un cambio al nivel "alto" en la segunda mitad. El "1" se representa por un nivel "alto" durante la primera mitad del intervalo de bit y un regreso al nivel de referencia en la mitad restante.

Fase partida (Marca) - El "0" se representa como un cambio al nivel "alto" durante la primera mitad del intervalo de bit y un regreso al nivel de referencia durante la mitad restante. El "1" se representa como el "0" en un primer intervalo y en forma inversa en un segundo intervalo, es decir, un nivel de referencia durante la primera mitad y un cambio a un nivel "alto" en la segunda mitad, todo esto realizado en forma alterna.

NRZ (L) - El "0" se representa como un nivel igual al nivel de referencia en todo el intervalo de bit. El "1" se representa como un nivel "alto" en todo el intevalo de bit.

NRZ (M) - El "0" se representa como ningún cambio de nivel. El "1" se representa como un cambio de nivel.

NRZ (S) - Se usa la misma representación que NRZ(M), excepto que el cambio de nivel se emplea para indicar un "0".

MILLER - Un "1" se representa por una transición de señal en el centro del intervalo de un bit. Un "0" se representa por ninguna transición, a menos que esté seguido por otro "0", en cuyo caso la transición de señal ocurre al final del intervalo del bit.

El caso general de MPC se tiene cuando cada muestra cuantizada se codifica con N pulsos, siendo cada uno de éstos de M niveles, esta muestra puede provenir de diferentes canales en el caso de que se esté utilizando un esquema de multicanalización en el tiempo.

Una vez que las muestras son cuantizadas, se convierten a grupos de bits, cuyo número depende de la codificación empleada. El número de niveles de cuantización determinará el número de bits utilizados en la codificación. Por ejemplo:

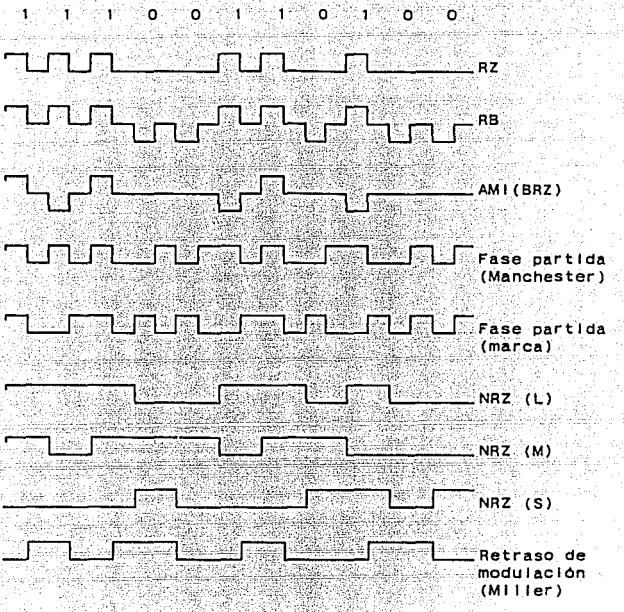


Fig. 3.5 Formatos de codificación

```
8 Niveles    <------> 3 bits
16 Niveles   <------> 4 bits
    .                  .
    .                  .
    .                  .
128 Niveles  <------> 7 Bits
```

Esto implica que se aumentará el ancho de banda ya que se tienen varios pulsos asignados al mismo instante de muestreo, pero por otra parte, disminuirá el ruido de cuantización, permitiendo reconstruir la señal a intervalos periódicos.

Para dar consistencia a la información transmitida se debe establecer un compromiso al asignar el número de niveles, el nivel del ruido de cuantización y la precisión requerida en la información, ya que de éstos parámetros dependerá el ancho de banda necesario.

En el caso particular de la señal de voz, se utilizan 128 niveles para los cuales se requieren 7 bits, aunque cuando se usa código de paridad son realmente 8 bits.

3.4 Modulación por Codificación de Pulsos Diferencial

Es una técnica principalmente utilizada para codificación de voz en los sistemas comerciales telefónicos y en la transmisión muestreada de información en imagen. Una característica de este tipo de señales al ser muestreadas a una frecuencia de Nyquist es que observan una buena correlación entre muestras permitiendo que la diferencia ponderada entre la muestra n-ésima X_n y la siguiente X_n sea más pequeña que la variación de la señal misma. Es decir

$$D_n(1) = X_n - a_1 X_{n-1}$$

$a_1 < 1$ - es la constante de ponderación, que se escoge para hacer mínima la variación de la diferencia.

El diagrama de bloques de la técnica MCPD se muestra en la Figura 3.6.

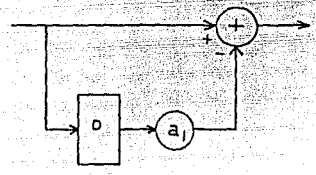


Fig. 3.6 Diagrama de bloques MCPD

donde D representa un elemento de retraso.

En el caso de la técnica MCPD, la salida del sistema es la primera diferencia de la señal de entrada, o sea:

$$D_n(a_1) = X_n - a_1 X_{n-1}$$

Existe una técnica que utiliza esta diferencia como el minuendo de una segunda diferencia y se le conoce como Modulación por Codificación de Pulsos Diferencial Doble (MCPDD) en donde:

$$D_n(a_1, a_2) = (X_n - a_1 X_{n-1}) - a_2 (X_{n-1} - a_1 X_{n-2})$$

El diagrama de bloques de la técnica MCPDD se muestra en la Figura 3.7.

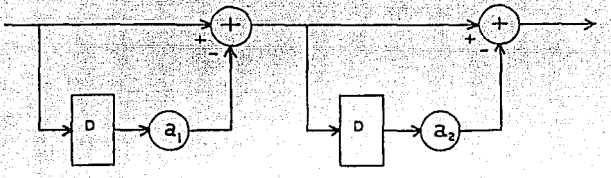


Fig. 3.7 Diagrama de bloques de MCPDD.

La ventaja en el empleo de ésta técnica de codificación, es el hecho de incrementar la capacidad del canal al reducir el ruido de cuantización por la reducción de la magnitud de la señal diferencia.

3.6 Modulación Delta

Es esencialmente una conversión analógica a digital de una señal, donde la información contenida en los instantes de muestreo son representados por palabras de código en una secuencia de bits.

Un esquema general de la modulación delta, se muestra en la Figura 3.8. En el transmisor, el valor muestreado X(kT) de X(t) se compara con un valor de predicción $\hat{X}(kT)$, y la diferencia $X(kT)-\hat{X}(kT)$ se cuantiza en uno de dos valores +A o - A. La salida del cuantizador se codifica usando un código binario por muestra y después enviada al receptor. En el receptor, el valor decodificado de la señal diferencia es sumado al valor predicción inmediato de la salida del receptor.

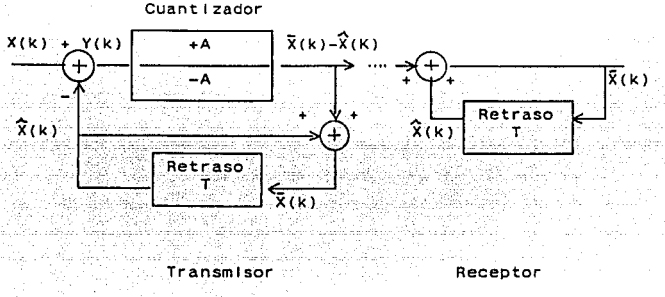


Fig. 3.8 Sistema de Modulación Delta. Razón de muestreo=1/T.

La operación de la modulación delta puede visualizarse con las señales que se muestran en la Figura 3.9. La señal de mensaje X(t) se compara con una aproximación $\hat{X}(t)$, y la señal diferencia $Y(t) = X(t) - \hat{X}(t)$ se cuantiza en dos niveles:

$$\text{si } X(t) - \hat{X}(t) < 0 \text{ se envía } -A$$

$$\text{si } X(t) - \hat{X}(t) > 0 \text{ se envía } +A$$

por lo tanto hay solo dos posibles valores de impulso para Ysq(t) (señal del cuantizador delta); esta señal puede transmitirse usando una secuencia binaria.

La demodulación de la señal, se realiza por la integración de Ysq(t), para formar la aproximación por escalones de X(t). La señal puede entonces hacerse pasar por un filtro paso bajas para suprimir los saltos discretos de $\hat{X}(t)$.

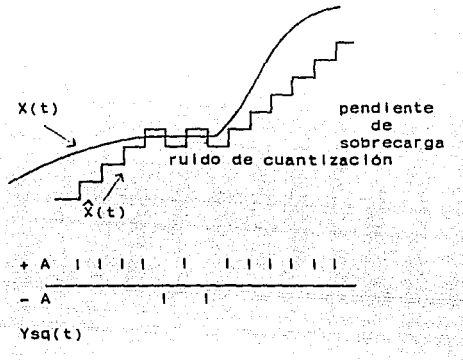


Fig. 3.9 Formas de onda de la modulación delta

En las aplicaciones prácticas, la salida del generador de pulsos no es naturalmente una secuencia de funciones pulso, mas bien, es una secuencia de pulsos que son estrechos con respecto a sus periodos, lo que requiere amplios anchos de banda.

Por otra parte, el esquema de la modulación delta sufre dos tipos de ruido:

- ruido de cuantización
- sobrecarga de pendiente.

El ruido de cuantización se origina al asignar un nivel de salida a la diferencia entre las muestras y el valor de predicción, ya que ésta diferencia no es exactamente el nivel asignado +A o -A, sino que es una aproximación a éstos. En la modulación delta, la reducción del ruido de cuantización sólo puede reducirse muestreando a una mayor frecuencia.

La sobrecarga de pendiente se produce cuando la pendiente de la señal de entrada excede la capacidad del sistema de seguir de cerca a la fuente analógica a una velocidad de muestreo dada.

La ventaja de los sistemas delta es que los circuitos para la modulación en el transmisor y la demodulación en el receptor son sustancialmente más simples que otros sistemas de modulación, como por ejemplo MCP o MCPD. Esta sencillez electrónica hace a la modulación delta una técnica atractiva. La modulación delta se usa principalmente para transmisión de señales de telefonía y telemetría.

Una de las desventajas la constituye el hecho de que Ysq(t) (señal enviada en bits) esta formada por pulsos estrechos, lo que requiere utilizar amplios anchos de banda. Sin embargo éste puede ser menor que MCP (por ejemplo, 32 kbps).

3.8 Modulación Delta Doble Paso

Una de las formas de solucionar la pendiente de sobrecarga que se presenta en la modulación delta, es variar el tamaño de los pulsos de salida con que se codifica la señal diferencia. Esto es, utilizar dos tamaños de escalones de tal forma que cuando la diferencia sea pequeña, se genere un escalón pequeño, y utilizar un tamaño de escalón mayor cuando la señal diferencia sea mayor. Esto implica que se utilicen dos bits para la codificación, teniendo 4 posibles valores.

A este procedimiento se le denomina modulación delta de doble paso y su funcionamiento puede visualizarse en la Figura 3.10.

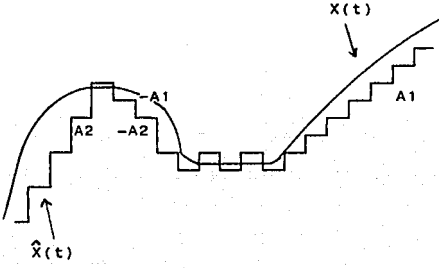


Fig. 3.10 Operación de la modulación delta de doble paso

La señal mensaje X(t) se compara con una aproximación $\hat{X}(t)$ y la señal diferencia $Y(t) = X(t) - \hat{X}(t)$ se cuantiza en 4 niveles:

si $A1 < X(t) - \hat{X}(t) < 0$ se envia $-A2$

si $A1 > X(t) - \hat{X}(t) < 0$ se envia $-A1$

si $A1 > X(t) - \hat{X}(t) > 0$ se envia $+A1$

si $A1 < X(t) - \hat{X}(t) > 0$ se envia $+A2$

donde A1 es un escalón pequeño y A2 es un escalón grande.

3.7 Codificación de Caracteres

Es necesario primero que se distinga entre la conversión de un número a binario y la codificación binaria del signo que representa a dicho número, aunque al final de cada caso se obtiene una serie de bits.

Los bits que se obtienen en la codificación son combinaciones de unos y ceros arreglados en base a las reglas del código que se está utilizando. La información binaria, son señales de pulsos modulados o señales de entrada y salida de algún sistema, y pueden ser transmitidos a través de un medio de comunicación tal como ondas de radio o a través de alambres eléctricos.

En algunos sistemas de comunicación, se requiere manejar datos que consisten no solamente de números sino también de letras y otros caracteres alfanuméricos, por lo que es necesario representar estos por medio de un código binario para alfabeto, además de números y algunos caracteres especiales. Un código alfanumérico es un código binario de un grupo de elementos consistente de los diez números decimales, los veintiseis caracteres del alfabeto y de cierto número de símbolos especiales (Morris M. Mano 1982) lo que da un total de 64 caractes.

La necesidad de representar más de 64 caracteres, dió origen a los códigos alfanuméricos de siete u ocho bits. Uno de los más conocidos y también muy importante es el código ASCII (American Standard Code for Information Interchange), que quiere decir Código Normalizado Americano para el Intercambio de Información. Otro de igual importancia, es el código EBCDIC (Extended BCD Interchange Code), que significa Código de Intercambio BDC Aumentado.

Los caracteres se transmiten en forma asincrona, por lo que las máquinas realizan esta función se denominan, dispositivos asincrónicos (uno la vez). Ejemplos tipicos de ellas,son las consolas de telex y de otras terminales de teclado.

Es necesario que cada carácter tenga su propia información de inicio y término para que una vez transmitido se reciba en forma adecuada. Existe una gran variedad de dispositivos que tienen diferentes longitudes de caracteres que varian entre cinco y ocho digitos mas dos o tres para funciones de inicio y parada. En Estados Unidos actualmente se encuentran estandarizados a caracteres de diez digitos, que constan de un cero de partida, siete bits de datos, un bit de comprobación de paridad y un bit de parada al final de la serie.

El código ASCII consta de siete bits, pero para efectos prácticos se usan ocho bits, ya que el octavo bit se agrega para efectos de comprobación de paridad.

3.8 Señalización de Respuesta Parcial

Con el avance de las computadoras y las máquinas de procesamiento de datos, así como el crecimiento necesario para la transmisión de grandes volúmenes de datos binarios, en base al actual progreso en las vías de comunicación, las simples técnicas de transmisión binaria tienden a ser y son comúnmente usadas para aceptar más datos. Sin embargo, la capacidad de rapidez de los sistemas de transmisión binaria son limitadas e insuficientes, para satisfacer en la actualidad el requerimiento de datos a alta velocidad, surgiendo en consecuencia el sistema de multiniveles, la cual usa más de dos estados de señal; principios que fueron dados por Nyquist (1924).

Los sistemas multinivel tienen en principio, n veces la eficiencia de velocidad para sistemas binarios. Esto implica, gran sensibilidad al ruido y un funcionamiento deficiente de error, además requieren de un equipo complejo. Debido a esto se ha tratado a la interferencia intersimbólica como un fenómeno indeseable que produce una degradación en el funcionamiento del sistema. En realidad, su mismo nombre connota un efecto desagradable; sin embargo, al agregar interferencia intersimbólica a la señal transmitida, en una forma controlada, es posible lograr un índice de señalización de 2BT símbolos por segundo en un canal de ancho de banda de BT hertz. A estos esquemas se les llama codificación correlativa o señalización de respuesta parcial. El diseño de estos esquemas se basa en la premisa de que ya que la interferencia intersimbólica que se introduce en la señal transmitida se conoce, su efecto puede interpretarse en el receptor. De esta manera, la codificación correlativa puede considerarse como un medio práctico de lograr el índice de señalización máximo teórico de 2BT símbolos por segundo en un ancho de banda de BT hertz, al utilizar filtros realizables.

La idea básica de la codificación correlativa implica duplicación de la capacidad de transmisión de un sistema binario rectilíneo. Las técnicas duobinarias han sido aplicadas tanto en la transmisión de datos de alta velocidad por canales telefónicos, como a la transmisión de dos hileras de datos T1 por un canal que normalmente puede manejar una hilera T1 a la velocidad de 1 544 Mbits/s. Nótese que en la combinación de bits sucesivos en el uso de la técnica duobinaria, o cualquier otra técnica de codificación correlativa, los bits no tienen que ser necesariamente de la misma hilera de datos. Pueden mezclarse dos hileras independientes de datos, como se hace normalmente con la multicanalización en tiempo, tomando bits sucesivos de cada una de las hileras en forma secuencial; las hileras de datos se separan entonces en el punto de recepción.

Señalización Duobinaria

Considerando una sucesión de entrada binaria (bk) que consta de dígitos binarios no correlacionados, cada uno con duración de Tb segundos; el símbolo 1 representado por un pulso de amplitud de +1 volt, y el símbolo 0 mediante un pulso de amplitud de -1 volt. Cuando esta sucesión se aplica a un codificador duobinario, se convierte en una salida de tres niveles, que son, -2, 0 y +2 volts. Para producir esta transformación, se puede utilizar el esquema de la Figura 3.11

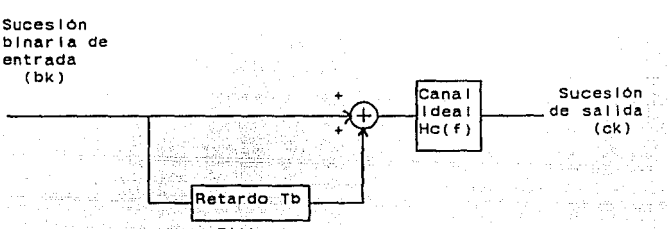


Fig. 3.11 Filtro duobinario de conversión H(f)

La sucesión binaria (bk) se hace pasar primero a través de un filtro simple que comprende un solo elemento de retardo. Por cada impulso unitario que se aplica a la entrada de ese filtro, se obtienen dos impulsos unitarios repartidos en Tb segundos a la salida del filtro. Se puede expresar entonces el dígito ck a la salida del codificador duobinario como la suma del dígito binario presente bk y su valor anterior bk-1), como se demuestra mediante

$$ck = bk + bk-1)$$

Uno de los efectos de la transformación que describe esta ecuación es cambiar la sucesión de entrada (bk) de los dígitos binarios no correlacionados en una sucesión (ck) de dígitos correlacionados. Esta correlación entre los niveles adyacentes transmitidos se puede considerar como la introducción de manera artificial de interferencia intersimbólica en la señal transmitida. Sin embargo, esta interferencia intersimbólica se encuentra bajo el control del diseñador.

Los datos originales (bk) pueden detectarse de la sucesión codificada duobinaria (ck) restando el dígito binario descodificado anterior del dígito que se va recibiendo ck de acuerdo con la ecuación (1). Específicamente, si bk representa la estimación del dígito binario original bk como lo concibió el receptor en el tiempo t = kTb, se tendrá

$$bk = ck - \acute{b}k-1)$$

Es evidente que si ck se recibe sin error y si también la estimación anterior $\acute{b}k-1)$ en el tiempo t = (k - 1)Tb corresponde a una decisión correcta, entonces la estimación actual bk será también correcta. La técnica de utilizar una estimación almacenada del símbolo anterior se llama realimentación de decisión.

Sin embargo, se presenta un problema potencial, en donde no solamente se inducirá un error de decisión en el receptor si yk se recibe incorrectamente, sino que este error puede tender a propagarse.

Un error en un espacio de tiempo provoca un error en el siguiente espacio, el cual puede continuar propagándose por los espacios contiguos. Para eliminar este problema potencial (característico de los esquemas de realimentación de decisión) se emplea un procedimiento de precodificación.

Sucesión binaria (bk) (entrada)	-1	-1	1	1	-1	1	-1	-1	1
Sucesión duobinaria (ck) (salida)		-2	0	2	0	0	0	-2	0
Sucesión obtenida al aplicar la ecuación(2)	-1	-1	1	1	-1	1	-1	-1	1

Tabla 1. Señalización duobinaria

Precodificación Duobinaria

Un medio práctico de evitar la propagación de errores es el uso de precodificación antes de la codificación duobinaria, como se muestra en la Figura 3.12

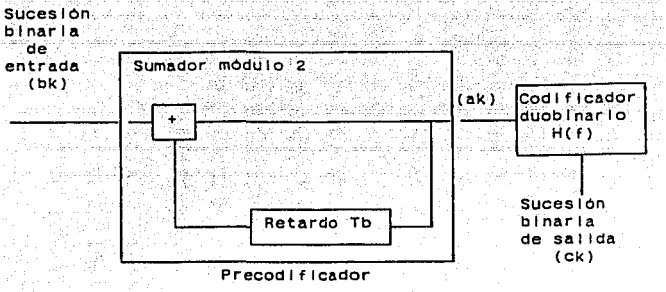


Fig. 3.12 Precodificación duobinaria

La operación de precodificación realizada sobre la sucesión binaria de entrada (bk) la convierte en otra sucesión binaria (ak) definida por

$$ak = bk \oplus a(k-1)$$

en donde el simbolo O (or exclusivo) representa adición módulo dos de los digitos binarios bk y a(k-1). Esta operación equivale a la operación de or exclusivo. Una compuerta de or exclusivo opera de la manera siguiente: la salida de una compuerta or exclusiva de dos entradas es un 1 si una y solamente una entrada es un 1; de otra manera, la salida sigue siendo un 0. La salida resultante del precodificador (ak) se aplica en seguida al codificador duobinario, produciendo así la sucesión (ck) que está relacionada con (ak) en la forma siguiente

$$ck = ak + a(k-1)$$

Nótese que a diferencia de la operación lineal de codificación duobinaria, la precodificación es una operación no lineal.

A la salida del precodificador ak puede tomar valores de 1 ó 0 y en consecuencia ck puede tomar los valores de 0, 1 y 2.

De la ecuación (3) se deduce la siguiente regla de decisión para detectar la sucesión binaria original de entrada (bk) a partir de (ck):

bk = 0 si ck = 0 ó 2

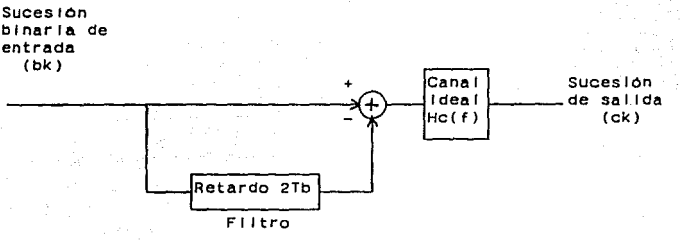
bk = 1 si ck = 1

Sucesión binaria (bk) (entrada)		0	0	1	1	0	1	0	0
Sucesión binaria (ak)	1	1	1	0	1	1	0	0	0
Sucesión módulo 2 (ak)		1	1	0	1	1	0	0	0
Sucesión de la precodificación duobinaria (ck) (salida)		2	2	1	1	2	1	0	0
Sucesión obtenida al aplicar la regla de decisión		0	0	1	1	0	1	0	0

Tabla 2. Precodificación duobinaria

Señalización Duobinaria Modificada

La técnica duobinaria modificada comprende un intervalo de correlación de dos dígitos binarios. Estos se logra restando los dígitos binarios de entrada separados 2Tb segundos, como se indica en el diagrama de bloques de la Figura 3.13.



Filtro de conversión duobinario modificado

Fig. 3.13 Señalización duobinaria modificada

La salida del filtro de conversión duobinaria modificada esta relacionada con la sucesión (ak) de su entrada en la forma siguiente:

$$ck = bk - b(k-2)$$

Aquí, otra vez, se encuentra que se genera una señal de tres niveles. Si ak = q 1, como antes se había supesto, ck toma uno de los valores: 2, 0 y -2. Una característica útil del codificador duobinario modificado es el hecho de que se salida no tiene componente de CD. Esta propiedad es importante ya que, en la práctica, muchos canales de comunicación no pueden transmitir una componente de CD.

Para extraer los símbolos binarios originales de entre las muestras duobinarias modificadas ck, recuérdese que ck se generó haciendo ck = bk - b(k-2), es decir, el valor binario actual es el mismo que se ha estimado dos espacios de tiempo anteriores; se tiene entonces como una regla de decodificación:

Si ck = 1 entonces bk = 1

ck = -1 entonces bk = 0

ck = 0 entonces bk = b(k-2)

Sucesión binaria (bk) (entrada)	0	1	0	0	1	1	1	0	0	1
Sucesión duobinaria modificada (ck) (salida)			0	-1	1	1	0	-1	-1	1
Sucesión obtenida al aplicar la regla de decodificación				0	1	1	1	0	0	1

Tabla 3. Señalización duobinaria modificada

La tabla 3 muestra la entrada binaria de diez bits, y en la recuperación de ésta sólo hay siete bits, esto se debe a que los tres primeros se usan como bits de identificación.

Precodificación Duobinaria Modificada

Para eliminar la posibilidad de propagación de error en el sistema duobinario modificado, se usa un procedimiento de precodificación similar al que se utilizó en el caso duobinario. Específicamente, antes de la generación de la señal duobinaria modificada, se usa una adición lógica módulo dos sobre las señales separadas 2Tb segundos, como se demuestra mediante la siguiente ecuación

$$ak = bk \oplus a(k-2)$$

en donde (bk) es la sucesión binaria de entrada y (ak) es la sucesión binaria a la salida del precodificador, de donde se obtiene la ecuación de salida de la precodificación duobinaria modificada:

$$ck = ak - a(k-2)$$

Nótese que la adición módulo dos y la sustracción módulo dos son las mismas. La sucesión (ak) así producida se aplica en seguida al filtro de conversión duobinario modificado como se muestra en la Figura 3.14

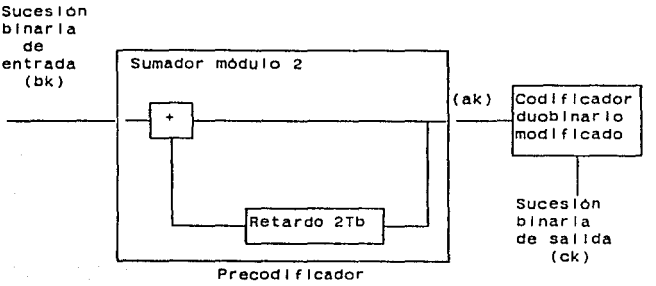


Fig. 3.14 Precodificación duobinaria modificada

en el caso de la figura, el dígito de salida ck es igual a 0, +1 ó -1. Se encuentra también que bk se puede extraer de ck no tomando en cuenta la polaridad de ck. Ya que las operaciones más y menos son las mismas, en la lógica módulo dos, el valor módulo dos de ck proporciona también el valor deseado de bk. De acuerdo con esto, en el receptor se puede extraer la sucesión original (bk) al realizar:

$$bk = | ck |$$

Sucesión binaria (bk) (entrada)			1	1	1	1	0	1	0	0	1
Sucesión binaria (ak)	1	0	0	1	1	0	1	1	1	0	1
Sucesión módulo 2			0	1	1	0	1	1	1	1	0
Sucesión de la precodificación duobinaria modificada (salida)			-1	1	1	-1	0	1	0	0	-1
Sucesión obtenida al aplicar el valor absoluto			1	1	1	1	0	1	0	0	1

Tabla 4. Precodificación duobinaria modificada

4 Programación de los procesos

La programación de los procesos se realizó usando el compilador TURBO-PASCAL 3.0, bajo el sistema operativo MS-DOS versión 2.11. Para la operación del programa es recomendable que la microcomputadora cuente con el coprocesador aritmético 8087 que permite utilizar la versión TURBO-87, la cual reduce considerablemente el tiempo de compilación y ejecución.

El programa corre bajo el nombre de SIMULA y su operación es interactiva con el usuario, de tal manera que éste simplemente selecciona la opción de un menú. Para facilitar el desarrollo y mantenimiento del programa se utilizó la programación estructurada y se estableció la siguiente nomenclatura:

- las variables y los nombres de las rutinas se escribieron con letra minúscula.
- las palabras reservadas y funciones, con letra mayúscula.

Las señales de comunicaciones utilizadas para la simulación de los procesos, están representadas por archivos tipo entero de 512 datos.

La presentación inicial del programa es el menú:

1) Genera señal de prueba

2) Muestreo

3) Cuantización (compansión)

4) Codificación

5) Decodificación

6) Cuantización (expansión)

7) Despliegue

0) Fin

1) Generación de la señal de prueba

Además de leer cualquier señal proporcionada por el usuario, el programa contempla la construcción de tres tipos de señales: Senoidal, Aleatoria y Datos del usuario.

La señal senoidal se generó mediante la programación de la siguiente ecuación:

y(x) = SENO (x)

El código correspondiente es:

```
m_actual:=ROUND(magnitud * SIN(expansion * i));
```

El parámetro "i" permite aumentar o disminuir la frecuencia de la senoidal y su rango de valores está entre 0.02 y 0.1. La variable "magnitud" define la amplitud de la señal y su rango de valores puede estar entre 0 y 87.

La señal aleatoria se generó mediante el uso de la función RANDOM del compilador, el código correspondiente es:

```
m_actual:= ROUND(RANDOM * 87);
```

La señal construida por el usuario, se forma con valores enteros proporcionados desde el teclado. El rango de valores que se puede introducir va de -87 a +87, de esta manera, si se desea conocer el resultado de aplicar un proceso a un conjunto de datos no muy grande, se puede utilizar esta opción.

El código correspondiente es:

```
WRITE('Dame dato',cont+1,' en el rango (-87,87) ó 9999');
WRITE(' para Terminar');
READLN(m_actual);
```

2) Muestreo

Para la utilización del proceso de muestreo se debe tener en cuenta los siguientes pasos.

Lambda, es el valor en el cual se determina la distancia (velocidad) entre los puntos de una y otra muestra.

```
WRITE('Dame el valor de lambda (0.1 - 1.5) lambda = ');
```

En esta sección del programa se determina el valor de la muestra por su posición.

```
pos:=i*lambda;
j:=TRUNC(pos);
```

Cuando el valor de la posición es un entero y no es necesario interpolar se tiene:

```
mn[i]:=ma[j];
```

En el caso de tener una posición con valor decimal o fraccionario, en el que no se encuentra determinada la muestra, su valor se obtendrá por medio de la interpolación lineal:

```
mn[i]:=ROUND(ma[j]+FRAC(pos)*(-ma[j]+ma[j+1]));
```

3) Cuantización(Compansión)

La entrada y salida de datos son archivos de 512 elementos, donde el archivo de salida tiene valores dentro de los niveles de cuantización elegidos; ambos archivos son de tipo entero. La cuantización se realiza de manera uniforme, y no uniforme por las leyes μ y A. Como primer paso se obtiene el valor máximo del archivo de entrada:

```
max:=0;
FOR h:=1 to 512 DO
  BEGIN
    READ(entrada,h);
    IF h > max THEN
       max:=h;
  END;
```

Hecho ésto, se normalizan los datos con el fin de tener valores menores o iguales a 1; la normalización se hace sólo para la cuantización no uniforme. Habiendo escogido el número de niveles y el tipo de cuantización, se realiza la asignación de niveles:

```
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,lectura);
   IF opcion='1' THEN
      dato1:=lectura / nivel
   ELSE
      dato1:=lectura / max;
   CASE opcion OF
   '1': escritura:=ROUND(dato1) * ROUND(nivel);
   '2': BEGIN
        datox:=LN(1 + (mu * dato1)) / mudiv;
        datox:=datox * (niveles DIV 2);
        escritura:=ROUND(datox);
        escritura:=escritura * ROUND(nivel);
        END;
   '3': BEGIN
        IF dato1<=0.01 THEN
           datox:=(a * dato1) / adiv
        ELSE
           datox:= (1+ LN(a * dato1)) / adiv;
        datox:=datox*(niveles DIV 2);
        escritura:=ROUND(datox);
        escritura:=escritura * ROUND(nivel);
        END;
   END;
   WRITE(salida,escritura);
   END;
```

El código anterior corresponde a la cuantización en el proceso de compansión.

4) Codificación

Se presenta el siguiente submenú:

1) PCM
2) DPCM
3) Modulación delta
4) Modulación delta doble paso
5) Codificación de textos
6) Señalización duobinaria
7) Señalización duobinaria modificada
8) Precodificación duobinaria
9) Precodificación duobinaria modificada

A continuación se muestran los algoritmos y códigos correspondientes a cada una de las opciones presentadas, en la mayoría de los casos, sólo se consideran los segmentos de código correspondientes al modelo simulado.

a) Modulación por Codificación de Pulsos

El programa realiza diferentes tipos de codificación dependiendo de la selección del usuario en el siguiente menú:

1) RZ
2) RB
3) AMI (BRZ)
4) Fase Partida (Manchester)
5) Fase Partida (marca)
6) NRZ (L)
7) NRZ (M)
8) NRZ (S)
9) Retraso de modulación (Miller)
0) Regresar al menú anterior

Cada bit del dato a desplegar, junto con la técnica seleccionada en el menú anterior, se envían como parámetros en el llamado de la rutina "dibuja":

```
FOR i:=1 to 8 DO
   IF bitsal[i]='1' THEN
      dibuja(1,co)
   ELSE
      dibuja(0,co);
```

La rutina "dibuja" despliega cada bit en el código deseado por el usuario:

```
CASE metodo OF
```

para RZ, se tiene:

```
'1': IF bit=1 THEN
        cual(1)
     ELSE
        DRAW(xh,y1,xh+40,y1,1);
```

para RB, se tiene:

```
'2': IF bit=1 THEN
        cual(1)
     ELSE
        cual(3);
```

para AMI (BRZ), se almacena en la variable "toca", el valor del bit desplegado, por lo que se tiene:

```
'3': CASE bit OF
     0: DRAW(xh,y1,xh+40,y1,1);
     1: IF toca='arriba' THEN
           BEGIN
           toca:='abajo';  cual(1);
           END
        ELSE
           BEGIN
           toca:='arriba';  cual(3);
           END;
     END;
```

para Fase partida (Manchester), se almacena en la variable "fue1", el valor del bit desplegado, por lo que se tiene:

```
'4': IF bit=1 THEN
        BEGIN
        IF fue1='uno' THEN
           DRAW(xh,y1,xh,y0,1);
        cual(4);  fue1:='uno';
        END
     ELSE
        BEGIN
        IF fue1='cero' THEN
           DRAW(xh,y0,xh,y1,1);
        cual(5);  fue1:='cero';
        END;
```

para Fase Partida (marca), se almacena en la variable "fue", el valor y la fase del bit desplegado:

```
'5': CASE BIT OF
     1: IF fue='unolz' THEN
           BEGIN
           fue:='unode';  cual(5);
           END
        ELSE
           IF fue='unode' THEN
              BEGIN
              fue:='unolz';  cual(4);
              END
           ELSE
              IF fue='cerolz' THEN
                 BEGIN
                 fue:='unode';  cual(5);
                 END
              ELSE
                 BEGIN
                 fue:='unolz';  cual(4);
                 END;
     0: IF fue='unolz' THEN
           BEGIN
           fue:='cerolz';  DRAW(xh,y1,xh,y0,1);
           cual(4);
           END
        ELSE
           IF fue='unode' THEN
              BEGIN
              fue:='cerode';  DRAW(xh,y1,xh,y0,1);
              cual(5);
              END
           ELSE
              IF fue='cerolz' THEN
                 BEGIN
                 fue:='cerolz';  DRAW(xh,y1,xh,y0,1);
                 cual(4);
                 END
              ELSE
                 BEGIN
                 fue:='cerode';  DRAW(xh,y1,xh,y0,1);
                 cual(5);
                 END;
END
```

para NRZ (L) cada dato del archivo de entrada representa una muestra cuantizada, la cual se convierte a un patrón de "1"s y "0"s. Estos son presentados en la pantalla en forma de pulsos rectangulares. Siendo la codificación NRZ (L) una de las técnicas más comunes, se programó el despliegue de la información en forma condensada o a detalle. Al usuario se le presenta el siguiente submenú:

1) 52 muestras por pantalla (CONDENSADO)
2) 4 muestras por pantalla (DETALLE)

En el caso del despliegue a detalle, se indica en la variable "fue2" que bit se desplegó, con el objeto de tener el trazo adecuado del siguiente bit:

```
'6': CASE BIT OF
     1: IF fue2='uno' THEN
           BEGIN
           fue2:='uno';  DRAW(xh,y0,xh+40,y0,1);
           END
        ELSE
           IF fue2='cero' THEN
              BEGIN
              fue2:='uno';  cual(7);
              END;
     0: IF fue2='cero' THEN
           BEGIN
           fue2:='cero';  DRAW(xh,y1,xh+40,y1,1);
           END
        ELSE
           IF fue2='uno' THEN
              BEGIN
              fue2:='cero';  cual(9);
              END;
     END;
```

El despliegue condensado se explica en la página 46 ya que se utiliza una rutina diferente.

para NRZ (M), se almacena en la variable "fue3", el valor del bit desplegado, por lo que se tiene:

```
'7': CASE BIT OF
     1: IF fue3='arriba' THEN
           BEGIN
           fue3:='abajo';  cual(9);
           END
        ELSE
           BEGIN
           fue3:='arriba';  cual(7);
           END;
     0: IF fue3='arriba' THEN
           BEGIN
           fue3:='arriba';  DRAW(xh,y0,xh+40,y0,1);
           END
        ELSE
           BEGIN
           fue3:='abajo';  DRAW(xh,y1,xh+40,y1,1);
           END;
     END;
```

para NRZ (S), se almacena en la variable "fue3", el valor del bit desplegado, por lo que se tiene:

```
'8': CASE BIT OF
     0: IF fue3='arriba' THEN
          BEGIN
          fue3:='abajo';  cual(9);
          END
        ELSE
          BEGIN
          fue3:='arriba';  cual(7);
          END;
     1: IF fue3='arriba' THEN
          BEGIN
          fue3:='arriba';  DRAW(xh,yO,xh+4O,yO,1);
          END
        ELSE
          BEGIN
          fue3:='abajo';  DRAW(xh,y1,xh+4O,y1,1);
          END;
     END;
```

para el Retraso de Modulación (Miller), se almacena en la variable "fue4", el valor y la fase del bit desplegado (Figura 4.1).

En el caso de NRZ (L) en forma condensada, se utiliza una rutina especial llamada "condensado", en donde la muestra leída se convierte en una secuencia binaria mediante la rutina auxiliar "letr" (Figura 4.2), que recibe como entrada el número "n1" y entrega a su salida el arreglo de caracteres "letra". En caso de tenerse un valor negativo, se realiza una representación en complemento a dos mediante la rutina "comp2" (Figura 4.3).

```
'9': CASE BIT OF
     1: IF fue4='unolz' THEN
           BEGIN
           fue4:='unode';  cual(5);
           END
        ELSE
           IF fue4='unode' THEN
              BEGIN
              fue4:='unolz';  cual(4);
              END
           ELSE
              IF fue4='ceroar' THEN
                 BEGIN
                 fue4:='unolz';  cual(4);
                 END
              ELSE
                 BEGIN
                 fue4:='unode';  cual(5);
                 END;
     0: IF fue4='unolz' THEN
           BEGIN
           fue4:='ceroab';  DRAW(xh,y1,xh+40,y1,1);
           END
        ELSE
           IF fue4='unode' THEN
              BEGIN
              fue4:='ceroar';  DRAW(xh,y0,xh+40,y0,1);
              END
           ELSE
              IF fue4='ceroar' THEN
                 BEGIN
                 fue4:='ceroab';  cual(9);
                 END
              ELSE
                 BEGIN
                 fue4:='ceroar';  cual(7);
                 END;
     END;
```

Fig. 4.1 Retraso de modulación (Miller)

```
i:=8;
REPEAT
   res := n1 MOD 2; letra[i] := CHR(48 + res);
   n1 := n1 DIV 2;  i := i - 1;
UNTIL n1=0;
WHILE i >=1 DO
   BEGIN
   letra[i] := '0'; res:=0; i:=i-1;
   END;
```

Fig. 4.2 Rutina "letr"

```
i:=n+1;
REPEAT
   i:=i-1;
UNTIL letra[i]='1';
i:=i-1;
WHILE i>=1 DO
   BEGIN
   IF letra[i]='0' THEN
      letra[i]:='1'
   ELSE
      letra[i]:='0';
i:=i-1;
END;
```

Fig. 4.3 Rutina "comp2"

Una vez que se tiene el equivalente binario de la muestra, se despliega en la pantalla mediante la rutina "pulsos", la cual dibuja 52 muestras en la pantalla, distribuidas en 4 renglones. El código correspondiente a la rutina "pulsos" es:

Para cada muestra se dibujan siete bits

```
FOR i:=1 TO 7 DO
   BEGIN
```

Se dibuja un pulso rectangular si el bit es '1':

```
   IF letra[i]='1' THEN
      BEGIN
      IF uno=false THEN
         BEGIN
         uno:=TRUE;  DRAW(xh,y1,xh,y0,1);
         END;
      DRAW(xh,y1,xh+5,y1,1);
```

Una vez dibujado, se prepara el trazo del siguiente bit, sea cero o uno:

```
      IF letra[i+1]='0' THEN
         DRAW(xh+5,y1,xh+5,y0,1)
      ELSE
         DRAW(xh+5,y1+9,xh+5,y1+11,1);
      END
```

Se dibuja un '0' si el bit representa un nivel bajo:

```
ELSE
   BEGIN
    IF uno=TRUE THEN
       BEGIN
       uno:=false;  DRAW(xh,y1,xh,yO,1);
       END;
   DRAW(xh,yO,xh+5,yO,1);
```

Una vez dibujado, se prepara el trazo del siguiente bit sea cero o uno:

```
      IF (letra[l+1]='1') THEN
         DRAW(xh+5,yO,xh+5,y1,1)
      ELSE
         DRAW(xh+5,y1+9,xh+5,y1+11,1);
      END;
```

Después de desplegar 13 muestras en un renglón, se actualizan las variables para poder dibujar el siguiente renglón,

```
    IF xh>500 THEN
       BEGIN
       GOTOXY(70,1); WRITE('MUESTRA'); GOTOXY(zy,zx);
       WRITE(lm,' - ',lm+12); xh:=0; yO:=yO+48; y1:=y1+48;
       zx:=zx+6; lm:=lm+13; nx:=nx+6; ny:=3;
       END;
```

al finalizar cada pantalla, se actualizan los valores para desplegar la siguiente pantalla,

```
    IF yO>200 THEN
       BEGIN
       GOTOXY(28,24);  WRITE('<1>  Para  continuar   <2> Para
       Terminar');
       READ(KBD,ya);
       HIRES; HIRESCOLOR(13); yO:=30; y1:=20; xh:=0; zx:=4;
       ny:=3; nx:=2;
       END;
```

b) Modulación por Codificación de Pulsos Diferencial

El algoritmo consiste en tomar dos datos del archivo de entrada y restar el primero del segundo, almacenando el resultado en un archivo de salida especificado por el usuario, además el resultado se despliega en la pantalla, en forma de secuencias de bits, hecho que permite observar que se requieren menos bits al utilizar esta técnica de codificación. El archivo codificado contiene la primera muestra del archivo de entrada, con el fin de reconstruir fielmente la señal original.

El código correspondiente es:

```
READ(entrada,anterior); WRITE(salida,anterior);
WHILE NOT EOF(entrada) DO
   BEGIN
   RAED(entrada,numero1);
   sal1:=numero1-anterior;
   WRITE(salida,sal1);
   END;
```

el número "sal1" se escribe en el archivo de salida especificado por el usuario y la salida se muestra gráficamente en la pantalla, mediante el uso de la rutina pulsos, explicada en el inciso anterior.

c) Modulación Delta

Los archivos de entrada y salida son de 512 enteros. El predictor utilizado es el que considera al valor predicción como la muestra anterior; el código asociado es:

```
READ(entrada,ent);
base:= ent;
k:=2;
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,ent2);
   sal := ent2 - base;
   IF sal>=0 THEN
      escritura:=k
   ELSE
      escritura:=-k;
   WRITE(salida,escritura);
   END;
```

Cabe mencionar, que la salida gráfica, es la señal acumulada para visualizar su transformación, y el archivo de salida sólo tiene secuencias de +k ó -k, que se puede observar con la opción de despliegue en bits.

d) Modulación Delta Doble Paso

La programación de este esquema es similar a la modulación delta, sólo que en este caso se utilizan dos tamaños de escalón para la salida codificada.

```
READ(entrada,ent);
base:=ent;
k1:=2; k2:=4;
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,ent2)
   sal:=ent2-base
   IF sal>0 THEN
      IF sal<=k1 THEN
         escritura:=k1
      ELSE
         escritura:=k2
   ELSE
      IF sal>=-k1 THEN
         escritura:=-k1
      ELSE
         escritura:=-k2
   WRITE(salida,escritura);
   END;
```

e) Señalización Duobinaria

Esta parte del programa realiza el proceso de señalización duobinaria por medio del siguiente algoritmo. El mensaje tiene x[i] bits de información, se toma el primer valor que es x[1] y se le suma el valor anterior que es x[i-1], este se da en el programa como una condición inicial en este proceso.

```
WRITELN;
i := 1;
x1 := -1;
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,x);
   y:=x+x1;  x1:=x;
   WRITE(y,'  ');
   WRITE(salida,y);
   i:=i+1;
   END;
```

f) Señalización Duobinaria Modificada

El algoritmo que se utiliza en esta sección presenta condiciones implícitas representadas por x[0],...,x[3], y son usadas sin alterar el mensaje. Al tomar los valores del mensaje x[i], a cada uno se le resta el valor x[i-2].

```
x[0]:= 0;
x[1]:= 1;
x[2]:= 0;
x[3]:= 0;
FOR i= 2 TO n DO
   BEGIN
   y[i]:=x[i]-x[i-2];
   WRITE(y[i],'  ');
   END;
```

g) Precodificación Duobinaria

Primero se lee el mensaje y un archivo implícito en el programa, el tamano de ambos es de 512 símbolos, con estos se realiza una suma módulo -2 o exclusivo.

```
FOR k:=0 TO n DO
   REPEAT
      IF d[k]=1 THEN
         BEGIN
         IF y[k]=1 THEN
            x[k]:=0
         ELSE
            IF y[k]=0 THEN
               x[k]:=1;
         END
      ELSE
         IF d[k]=0 THEN
            BEGIN
            IF y[k]=1 THEN
               x[k]:=1
            ELSE
               IF y[k]=0 THEN
                  x[k]:= 0;
            END
   UNTIL n=512;
```

Los valores encontrados con la suma módulo-2 o exclusivo, se suman con los valores del mensaje que se va a codificar. El código correspondiente es:

```
FOR k:= 0 TO n-1 DO
   BEGIN
   z[k]:=x[k]+y[k];
   WRITE(z[k],'  ');
   END;
```

h) Precodificación Duobinaria Modificada

En este proceso se utiliza el primer paso usado en el método de precodificación duobinaria, que es la suma módulo-2 o exclusivo.

La representación del algoritmo empleado es similar al de la codificación duobinaria, pero en este caso se trata de una sustracción.

```
FOR k:=0 TO n-1 DO
   BEGIN
   z[k]:=y[k]-x[k];
   WRITE(z[k],'  ');
   END;
```

i) Codificación de textos

Primero se lee el texto que contiene los caracteres que se van codificar en algún código de ocho bits, dichos caracteres pueden encontrarse en algún archivo ó escribirlos en ese momento. A cada caracter se le aplica la función ORD y se obtiene su correspondiente valor numérico. En seguida se leen los archivos que contienen los códigos correspondientes necesarios para realizar la conversión de un texto que contiene caracteres, que serán convertidos cada uno de ellos en un código de ocho bits. Los códigos de que se disponen son: ASCII, EBCDIC y CODIGO INTERNO.

El código ASCII contiene los primeros 125 caracteres de todo el conjunto del código, el código EBCDIC contiene sólo algunos de los caracteres de todo su conjunto y el CODIGO INTERNO contiene el alfabeto en letras mayúsculas, los números del cero al nueve y algunos otros como *, + =,), (, $, /, ., . Cada caracter se encuentra almacenado en 8 bits.

Una vez seleccionado el código en que se desea codificar el texto, se procede a leer el código correspondiente. Una vez que se ha leído el texto y se ha seleccionado el código, se asigna el valor numérico obtenido de la función ORD en la posición donde se encuentra el mismo valor numérico en el código seleccionado previamente, el cual toma el valor que contiene esa posición, dicha posición contiene el código en 8 bits. Por medio de la instrucción CASE es posible seleccionar el tipo de código, los cuales se encuentran bajo los nombres: ASCII123, EBCDIC21 y CCODDINT (Figura 4.4).

Los códigos mencionados que se utilizan para la codificación de textos, son previamente almacenados por medio de un programa que se encuentra en el apéndice B. Por medio de este programa es posible el almacenamiento de códigos de 6, 7, 8 o 9 bits.

```
CASE ||1 OF
1: nombre := 'ASCII123';
2: nombre := 'EBCDIC21';
3: nombre := 'CCOODINT';
END;
ASSIGN(salida,nombre);
RESET(salida);
FOR I:=0 to 125 do
   FOR J:=0 TO 7 DO;
      BEGIN
      READ(salida,I);
      g[I,J]:=I;
      END;
WHILE NOT EOF(texto) DO
   BEGIN
   READ(texto,w);
   x:=ORD(w);
   WRITE(w,'= ');
   FOR I:=0 TO 7 DO
      BEGIN
      WRITE(entrada,g[x,I];
      WRITE(g[x,I]);
      END;
   END;
```

Fig. 4.4 Codificación de textos

5) Decodificación

Se presenta el siguiente submenú:

1) PCM
2) DPCM
3) Modulación delta
4) Modulación delta doble paso
5) Codificación de textos
6) Señalización duobinaria
7) Señalización duobinaria modificada
8) Precodificación duobinaria
9) Precodificación duobinaria modificada

A continuación se muestran los algoritmos y códigos correspondientes a cada una de las opciones presentadas, en la mayoría de los casos, sólo se consideran los segmentos de código correspondientes al modelo simulado.

a) Modulación por Codificación de Pulsos

Se lee un valor del archivo de entrada y se despliega en forma gráfica. El código correspondiente es:

```
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(archivo1,m_actual);
   DRAW(x1,174-y1,x,174-m_actual,1);  x1:=x;
   y1:=m_actual;  x:=x+1;
   END;
```

b) Modulación por Codificación de Pulsos Diferencial

Para obtener la señal original, se requiere que el usuario proporcione el valor inicial a partir del cual se reconstruirá la señal. Cada dato leído del archivo de entrada se va sumando a un valor inicial, lo que va formando la señal original. El resultado de cada suma se almacena en el archivo de salida, el código correspondiente es:

```
READ(entrada,v1); WRITE(salida,v1);
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,m_actual);
   v1:=v1+m_actual;
   DRAW(x1,174-y1,x,174-v1,1);  x1:=x;  y1:=v1;  x:=x+1;
   WRITE(salida,v1);
   END;
```

c) Modulación Delta

Esta decodificación se realizó acumulando la muestra leída en la variable "escritura", y se almacenó en el archivo de salida "salida".

```
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,ent2)
   base:=base+ent2;
   escritura:=base;
   WRITE(salida,escritura);
   END;
```

d) Modulación Delta Doble Paso

Las muestras del archivo de entrada, se acumularon en la variable "escritura" y se almacenaron en el archivo de salida "salida". Se muestra además, su formación gráfica.

```
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,ent2)
   base:=base+ent2;
   escritura:=base;
   WRITE(salida,escritura);
   END;
```

e) Decodificación de la Señalización duobinaria

El algoritmo que realiza este proceso es similar al de señalización duobinaria, difiere ya que en lugar de sumarse el término anterior, se le resta.

```
i:=1;
x1:=-1;
WHILE NOT EOF(entrada) DO
   BEGIN
   READ(entrada,x);
   y:=x-x1;
   x1:=y;
   WRITE(y,'  ');
   WRITE(salida,y);
   i:=i+1;
   END;
```

f) Decodificación de la Señalización Duobinaria Modificada

Para extraer el mensaje original se utilizaron las siguientes reglas:

Si y[i]=1 entonces z[i]= 1

Si y[i]=-1 entonces z[i]= 0

Si y[i]=0 entonces z[i]= z[i-2]

de donde el algoritmo resultante es

```
IF y[i]=1 THEN
   BEGIN
   z[i]:= 1;
   WRITE(z[i],' ');
   END
ELSE
   IF y[i]=-1 THEN
      BEGIN
      z[i]:=0;
      WRITE(z[i],' ');
      END
   ELSE
      BEGIN
      z[i]:= z[i-2];
      WRITE(z[i],' ');
      END;
```

g) Decodificación de la Precodificación Duobinaria

Para extraer los valores binarios originales del mensaje, se utilizarón las siguientes dos reglas.

Si y[i]=0 ó 2 entonces x[i]=0

Si y[i]=1 entonces x[i]=1

resultando el siguiente algoritmo:

```
FOR i:=0 TO n-1 DO
   BEGIN
   IF y[i]=1 THEN
      x[i]:=1
   ELSE
      IF (y[i]=0) OR (y[i]=2) THEN
         x[i]:= 0;
   WRITE(x[i],' ');
   END;
```

h) Decodificación de la Precodificación Duobinaria Modificada.

Para determinar el mensaje original, unicamente se aplica la función del valor absoluto.

```
FOR i:=0 TO n-1 DO
   BEGIN
   x[i]:=ABS(y[i]);
   WRITE(x[i],'  ');
   END;
```

l) Decodificación de caracteres.

Por medio de este algoritmo se realiza la conversión de un texto codificado en ASCII, a un texto de caracteres alfanuméricos, dicho texto codificado contiene un total de 512 bits, lo que da un total de 64 caracteres alfanuméricos. Los primeros 8 bits se convierten a su valor en decimal. Este procedimiento es válido en los tres casos, ASCII, EBCDIC y CODIGO INTERNO, al inicio de cada uno de ellos. A continuación se realiza la conversión de un texto codificado en EBCDIC, a un texto de 64 caracteres alfanumé- ricos.

```
j:=0;
REPEAT
    e:=p[j]*128+p[j+1]*64+p[j+2]*32+p[j+3]*16;
    e:=e+p[j+4]*8+p[j+5]*4+p[j+6]*2+p[j+7]*1; a:=e;
```

Una vez que se tiene el valor decimal del caracter, se obtiene el valor EBCDIC correspondiente, realizando la conversión indicada en el siguiente estructura de control (Figura 4.5).

Una vez obtenido el resultado numérico, se realiza la conversión de este resultado por medio de la instrucción CHAR, que convierte el valor decimal numérico en un caracter alfanumérico.

```
WRITE(CHAR(a));
```

Se repite este procedimiento cada 8 bits hasta terminar con los 512 bits codificados en EBCDIC.

```
j:=j+8;
UNTIL j:=512;
```

```
CASE a OF
64:a:=a-32;
75:a:=a-29;
76:a:=a-16;
77:a:=a-37;
78:a:=a-35;
80:a:=a-42;
90:a:=a-57;
91:a:=a-55;
93:a:=a-52;
94:a:=a-35;
96:a:=a-51;
107:a:=a-63;
108:a:=a-71;
109:a:=a-14;
122:a:=a-64;
123:a:=a-88;
124:a:=a-60;
126:a:=a-65;
127:a:=a-93;
92,97:a:=a-50;
110,111:a:=a-48;
END;
IF (a>=129) AND (a<=137) THEN
    a:=a-32;
IF (a>=145) AND (a<=153) THEN
    a:=a-39;
IF (a>=162) AND (a<=169) THEN
    a:=a-47;
IF (a>=193) AND (a<=201) THEN
    a:=a-128;
IF (a>=209) AND (a<=217) THEN
    a:=a-135;
IF (a>=226) AND (a<=233) THEN
    a:=a-143;
IF (a>=240) AND (a<=249) THEN
    a:=a-192;
```

Fig. 4.5 Decodificación de textos en EBCDIC

El algoritmo que a continuación se muestra, convierte textos codificados en CODIGO INTERNO de 8 bits, a un texto de caracteres alfanuméricos. Como primer paso se realiza la conversión de bits a números decimales, con el procedimiento que se describió al inicio de la decodificación de caracteres. A continuación se busca el valor numérico en cada caso, se realiza la operación correspondiente y al mismo tiempo se aplica la función CHAR para obtener el caracter alfanumérico de ese valor numérico.

```
CASE b OF
11:=WRITE(CHAR(a+50));
16:=WRITE(CHAR(a+27));
27:=WRITE(CHAR(a+19));
28:=WRITE(CHAR(a+13));
32:=WRITE(CHAR(a+13));
43:=WRITE(CHAR(a-7));
44:=WRITE(CHAR(a-2));
48:=WRITE(CHAR(a-16));
49:=WRITE(CHAR(a-2));
59:=WRITE(CHAR(a-15));
60:=WRITE(CHAR(a-20));
END;
IF (a>=17) AND (a<=25) THEN
   WRITE(CHAR(a+48));
IF (a>=33) AND (a<=41) THEN
   WRITE(CHAR(a+41));
IF (a>=50) AND (a<=57) THEN
   WRITE(CHAR(a+33));
```

Una vez obtenido el caracter alfanumérico de los primeros 8 bits, se escribe y se inicia el proceso completo con los siguientes 8 bits, hasta terminar con los 512 bits:

```
j:=j+8;
UNTIL j=512;
```

6) Cuantización (expansión)

La expansión de la señal se efectúo mediante el siguiente código, tomando en cuenta que sólo es para cuantización no uniformme. También en este caso se realizó una normalización con respecto al máximo valor de entrada.

```
WHILE NOT EOF (entrada) DO
  BEGIN
  CASE opcion OF
  '1':BEGIN
      dat:=mud * dato1;
      datox:=(EXP(dat)-1)/mu;
      datox:=datox * max;
  '2':BEGIN
      IF dato<0.0; THEN
         datox:=(dato1*adiv)/a
      ELSE
         datox:=EXP(dato1*adiv-1)/a;
      datox:=datox * max;
      END;
  END;
  datoy:=ROUND(datox);
  WRITE(salida,datoy);
  END;
```

7) Despliegue

Esta rutina despliega una señal ya sea en forma de código binario, o como una gráfica del tipo XY. El despliegue binario le permite al usuario corroborar el proceso de codificación realizado sobre una senal.

Con el objeto de tener diferentes señales disponibles, se realiza un procedimiento de lectura previo, utilizando el arreglo d para almacenar los datos. Se contempla el despliegue de cuatro señales máximo. El código correspondiente es:

```
FOR I:=1 to 512 do
   BEGIN
   READ(archivo,lectura);
   d[I]:=lectura;
   END;
```

La rutina de despliegue, ofrece diferentes maneras de presentar las señales. Se puede desplegar la señal con sus magnitudes reales, o bien se pueden ajustar al tamano de la pantalla; se puede desplegar solo una porción de la señal y ésta a su vez ser amplificada horizontalmente. Para realizar el ajuste del tamano de la señal, se requiere conocer los valores mínimo y máximo de la señal. El código correspondiente es.

```
min:=1E10+36
maxy:=-miny;
FOR I:=1 to numpun DO
   BEGIN
   IF d[I]<miny THEN miny:=d[I];
   IF d[I]>maxy THEN maxy:=d[I];
   END;
```

Con estos valores se puede encontrar el rango de valores que será distribuido en la escala vertical (174 pixels)

```
minmax(miny,maxy);
rango:=maxy-miny;
IF rango<>0 THEN escala:=174/rango
ELSE escala:=1;
```

Se procede a graficar la señal:

```
FOR I:=1 to numpun-1 DO
   DRAW(ROUND((I*eschor)+23),ROUND(174-(d[I]-miny)*escala),
   ROUND(((I+1)*eschor)+23),ROUND(174-(d[I+1]-miny)*escala),1)
```

Si se va a desplegar la magnitud real de los datos, se utiliza el siguiente código

```
FOR I:=1 to numpun-1 DO
 DRAW(I+23,ROUND(174-d[I]*0.04),I+24,ROUND(174-d[I+1]*0.04),1)
```

Programa Principal

Las rutinas anteriormente expuestas, se accesan desde el programa principal, por medio de la estructura de control CASE, que a continuación se muestra:

```
READ(KBD,ele);
CASE ele OF
'1': genera; (* Genera señal de prueba *)
'2': muestreo;
'3': cuantiza; (* técnicas de cuantización *)
'4': codificación; (* técnicas de codificación *)
'5': decodificación; (* técnicas de decodificación *)
'6': decuantiza; (* expansión *)
'7': despliegue;
END;
```

Además de las rutinas descritas, se implementaron rutinas auxiliares que realizan labores no involucradas directamente con los procesos principales. Estas rutinas se pueden consultar en el Apendice A, así como el programa fuente.

5 Resultados

Para mostrar los resultados de los distintos procesos implementados, se utilizarán dos señales de prueba como datos de entrada, y se obtendrán las señales de salida correspondientes, para observar sus transformaciones. Considérense como señales de prueba, las señales mostradas en las figuras 5.1 y 5.2, que corresponden a señales de baja y alta frecuencia respectivamente.

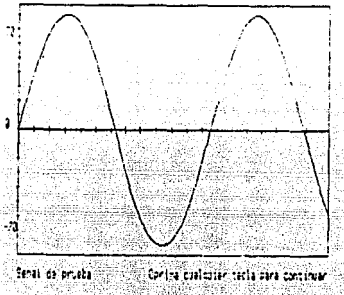


Fig. 5.1 Senoidal de baja frecuencia

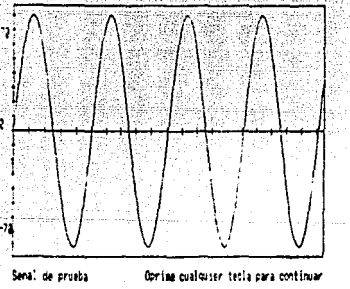


Fig. 5.2 Senoidal de alta frecuencia

Muestreo

Tomando como señal de entrada la senoidal de alta frecuencia mostrada en la Figura 5.2, se realiza el muestreo con lambda igual a 0.5, obteniendose la señal de la Figura 5.3, la cual representa una señal senoidal de baja frecuencia, debido a que lambda menor a uno provoca un efecto de expansión como resultado de la interpolación entre muestras.

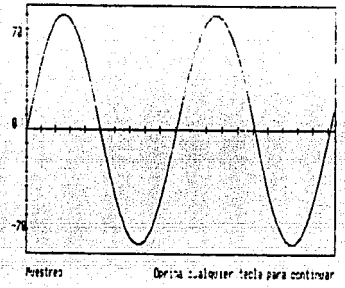


Fig. 5.3 Señal muestreada con lambda de 0.5

Si se muestrea con lambda igual a uno, se genera como resultado la misma señal de entrada. Si se muestrea con lambda igual a 1.5, se genera una señal de alta frecuencia, debido a que lambda mayor a uno provoca un efecto de contracción como resultado de saltar 1.5 muestras en el archivo de entrada. La señal de salida se trunca al final como se puede observar en la Figura 5.4, por la restricción en el número de datos de entrada.

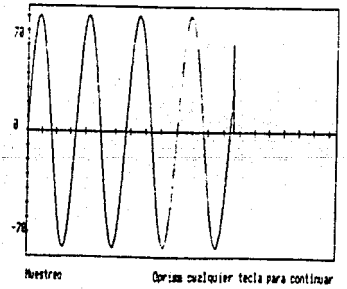


Fig. 5.4 Señal muestreada con lambda de 1.5

Si la señal de la Figura 5.1, se cuantiza uniformemente, y se utilizan 16 niveles para ello, se obtendrá la señal de la Figura 5.5, donde se observa la adecuación de la señal al número de niveles empleados.

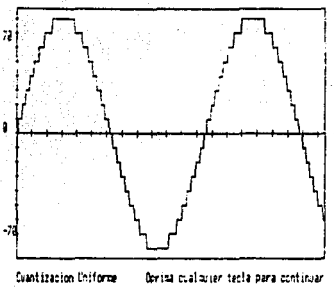


Fig. 5.5 Cuantización uniforme

Ahora, empleando la misma señal de entrada al proceso de cuantización no uniforme, y utilizando la ley μ con 128 de niveles, se obtendrá la señal de la Figura 5.6. En este caso se observa una deformación no lineal, debido a la característica del compresor.

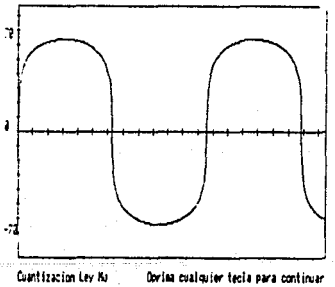


Fig. 5.6 Cuantización no uniforme (Ley μ)

Si se utiliza la cuantización por medio de la ley A, con 128 niveles, se obtiene la señal de la Figura 5.7. Estas señales corresponden a la Compansión del sistema Compansor referido en este trabajo.

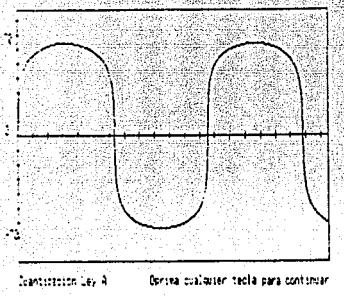


Fig. 5.7 Cuantización no uniforme (Ley A)

Al realizar la decuantización de la señal generada por la ley μ, se obtiene la señal de la Figura 5.8. Se puede notar que la forma de onda de esta señal, se aproxima a la señal de entrada original (Figura 5.1), aunque es notoria también, la deformación causada por el error de cuantización, el cual puede reducirse incrementando el número de niveles de cuantización.

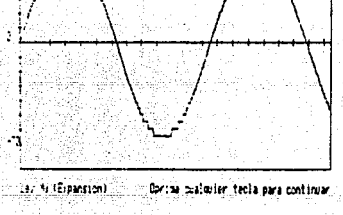


Fig. 5.8 Decuantización (Ley μ)

Al realizar la decuantización de la señal generada por la ley A, se obtiene la señal de la Figura 5.9. Notandose también el error de cuantización que sufre este esquema.

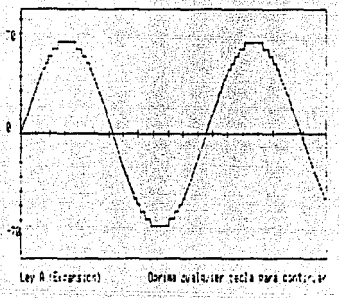


Fig. 5.9 Decuantización (Ley A)

Modificación por Codificación de Pulsos

A continuación se muestran los resultados de aplicar la técnica NRZ(L) a la señal de entrada de la Figura 5.6, aunque se despliegan solo cuatro muestras por pantalla, el programa permite ver todo el archivo procesado al oprimir la tecla 1. El programa contempla la codificación de las técnicas RZ, RB, AMI(BRZ), Fase partida (Manchester), Fase partida (marca), NRZ (M), NRZ(S) y Retraso de modulación (Miller).

NRZ(L)

Para ilustrar este proceso, se utiliza la señal resultante de la cuantización no uniforme que se muestra en la Figura 5.6. Al efectuar la codificación, se puede observar que cada muestra cuantizada es representada mediante un código binario de ocho bits. En caso de presentarse un valor negativo, se utiliza una representación en complemento a dos. El resultado se muestra en la Figura 5.10 en forma condensada y en la Figura 5.11 en forma detallada.

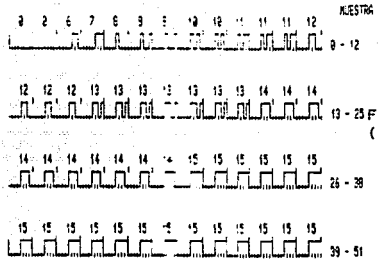


Fig. 5.10 Codificación NRZ (L) (condensado)

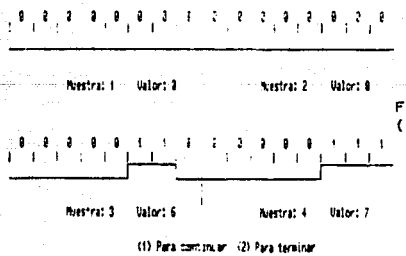


Fig. 5.11 Codificación NRZ (L) (detalle)

Tomando como señal de entrada la salida codificada en bits de la Figura 5.10, se transforma cada patrón de unos y ceros en su equivalente analógico. Este resultado se muestra en la Figura 5.12. Se hace notar que el archivo de entrada es el mismo, tanto para el proceso de codificación como para la decodificación, debido a que no se almacena en un archivo de disco el resultado de la codificación, sólo se muestra la transformación en el monitor.

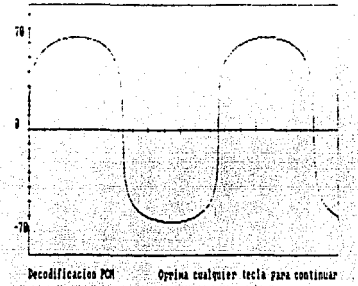


Fig. 5.12 Decodificación NRZ (L)

Modulación por Codificación de Pulsos Diferencial.

Tomando como entrada la señal mostrada en la Figura 5.6, se puede observar que al realizar el proceso de codificación en la Figura 5.13, se requiere un número menor de bits para codificar cada muestra. No obstante que se utiliza un patrón de 8 bits, en la mayoría de los casos solamente se utilizan 4 bits de salida.

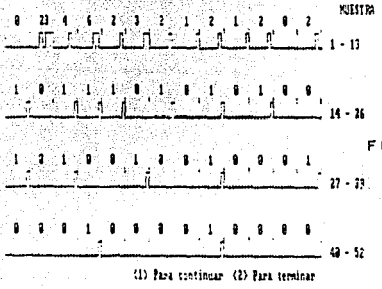


Fig. 5.13 Codificación MCPD

Utilizando como entrada la señal que se muestra en la Figura 5.13, se realiza la decodificación MCPD, mostrandose el resultado en la Figura 5.14. Como puede observarse, en el proceso de decodificación se conserva tanto la forma como la magnitud de la señal original, debido a que se almacena la primera muestra de la señal original en el archivo codificado.

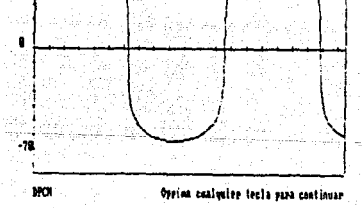


Fig. 5.14 Decodificación MCPD

Modulación Delta

Al utilizar la señal de la Figura 5.1 en la modulación delta, se obtiene la señal de salida mostrada en la Figura 5.15, donde se puede observar que ésta señal tiene una buena aproximación con respecto a la señal de entrada. Se observa también, el error de cuantización debido al tamaño del escalón utilizado.

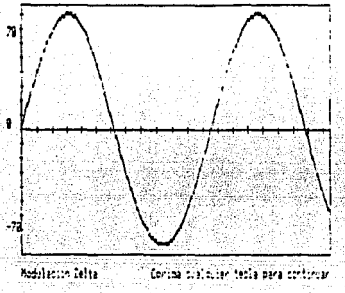


Fig. 5.15 Modulación Delta

Ahora utilizando la señal de alta frecuencia (Figura 5.2), como entrada al proceso de modulación delta, se obtiene como salida la señal de la Figura 5.16. Aquí se observa que dicha señal no corresponde a la señal de entrada, o sea que la modulación delta no opera correctamente con una señal de entrada de alta frecuencia. Esto es a causa del error de sobrecarga que sufre éste esquema.

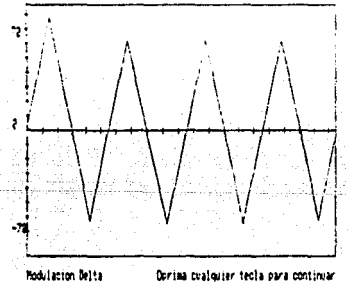


Fig. 5.16 Modulación Delta

Modulación Delta Doble Paso

Al utilizar como señal de entrada una señal de baja frecuencia (Figura 5.1), se obtiene la señal de salida mostrada en la Figura 5.17, donde se observa que es similar a la señal de entrada, y a la obtenida utilizando la modulación delta.

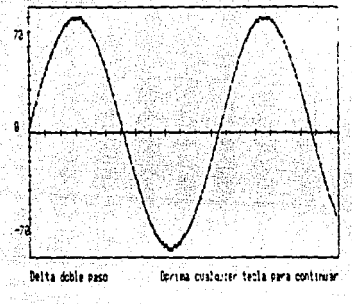


Fig. 5.17 Modulación Delta Doble Paso

Un resultado distinto es generado al utilizar como señal de entrada la señal de la Figura 5.2, al obtenerse como salida la señal mostrada en la Figura 5.18. En este resultado se observa como la modulación delta de doble paso sigue más estrechamente a la señal de entrada, debido a que utiliza dos posibles tamaños de escalón. Nótese, sin embargo, que el error de sobrecarga aún esta presente, aunque muy disminuido.

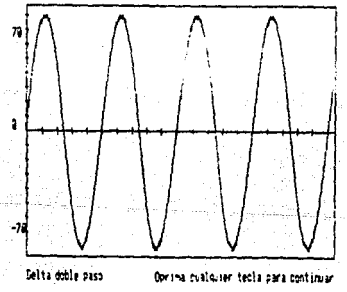


Fig 5.18 Modulación Delta Doble Paso

Codificación de caracteres

El texto que se muestra en la Figura 5.19 tiene caracteres comunes para los códigos ASCII, EBCDIC y código interno. Al codificar en ASCII dicho texto se obtiene el resultado mostrado en la Figura 5.20. Para verificar el caracter y su correspondiente valor ASCII, ambos se muestran simultáneamente. Es posible codificar en cualquiera de los otros dos códigos.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO, FACULTAD DE INGENIERIA.

Fig. 5.19 Texto de entrada

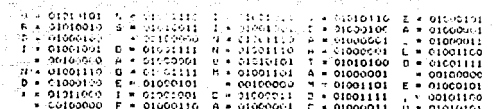


Fig. 5.20 Texto codificado en ASCII

Mediante el algoritmo de decodificación de textos se recupera el mensaje original, codificado previamente en ASCII. El resultado se muestra en la Figura 5.21.

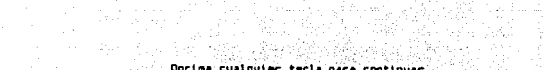


Fig. 5.21 Texto decodificado

Señalización de Respuesta Parcial

Para el uso de estos procesos, se deben tener datos codificados en cualquiera de los códigos mostrados en la sección anterior (ASCII, EBCIDIC, ..). Tomando como mensaje de entrada los datos de la Figura 5.20 se realiza la señalización duobinaria, obteniéndose los datos codificados en la Figura 5.22.

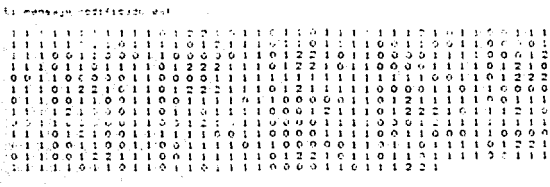


Fig. 5.22 Señalización Duobinaria

En la realización del proceso inverso, es decir, recuperar los datos originales, se utilizan los datos de salida de la Figura 5.22, usandolos como entrada en el proceso inverso, obteniéndose el mensaje original. El resultado se muestra en la Figura 5.23.

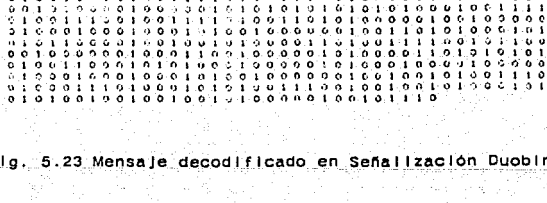


Fig. 5.23 Mensaje decodificado en Señalización Duobinaria

6 Conclusiones y alternativas

Tomando como referencia los resultados obtenidos en los procesos de simulación del Formato y Codificación de la Fuente, se puede concluir que los procedimientos implementados, proporcionan resultados que se aproximan al marco teórico planteado.

La presentación del programa a través de menues constituye un medio accesible para interactuar con el programa. El hecho de almacenar los resultados en archivos de disco, permite al usuario, por un lado, llevar un registro de los resultados obtenidos en los distintos procesos, y por otro, manipular los archivos de salida como entrada en una gama amplia de opciones. La operación del programa acelera el proceso de aprendizaje, al realizar procedimientos que permiten interpretar el comportamiento del modelo físico.

Entre sus limitaciones se tienen: la velocidad de operación, en cuanto al despliegue y escritura/lectura de los datos; al manejar sólo valores enteros se está perdiendo precisión en los datos procesados; en el manejo de señales analógicas, se contemplan sólo 512 elementos por cada señal; en el proceso de muestreo particularmente, no se contempla el análisis de señales dentro de un rango específico, debido a la restricción en el número de datos de salida. Además, la utilización de un índice de muestreo provoca una ampliación o reducción en la señal de salida, causando posibles pérdidas de información, dependiendo del índice de muestreo seleccionado.

Las alternativas para mejorar el proceso de simulación son: desarrollar programas en ensamblador, para mejorar la velocidad de operación y así poder acoplarlos con dispositivos operando en tiempo real; utilizar estos modelos bajo ciertas condiciones de prueba, como pueden ser el ruido, interferencia y distorsión; incorporación de rutinas que simulen los procesos no contemplados en este trabajo.

El programa representa la simulación de una parte del esquema general de un sistema de comunicaciones digitales, pero constituye la base para la formación de un programa integral que cubra la simulación de todos los procesos que involucran la transmisión de información. Tomando en cuenta las ventajas en el uso de las microcomputadoras actualmente, es posible encaminar la simulación dentro del área de comunicaciones de tal forma que ayude al estudio y comprensión de los sistemas de comunicaciones; además, la aparición de nuevas herramientas de programación e inovaciones tecnológicas, contribuirán a un mejor desempeño.

7 Bibliografía

Abate, J. E., 1967. "Linear and Adaptive Delta Modulation" Proc. IEEE, Marzo.

Carroll, M. J., 1987. "Simulation Using Personal Computers", Prentice-Hall.

Dehesa, S. M., 1982. "Cuantizacion escalar no uniforme de señales estocásticas". Tesis Maestría. DEPFI, UNAM.

Ferrel, G. S., 1985. "Sistemas de comunicación". Fondo Educativo Interamericano.

Freeman, L. R., 1980. "Telecommunications System Engineering" Wiley-Interscience.

Haykin, S., 1985. "Sistemas de comunicación". Interamericana, México.

Lathi, B. P., 1980. "Sistemas de comunicación", Limusa, México.

Mitra, A. D., Srimani, P.K., 1979. "Differential Pulse-Code Modulation". Int.J.Electronics, 46(6): 633-637.

Morris, M. M., 1982. "Logica Digital y Diseño de Computadores" Prentice Hall.

Schindler, H. R., 1970. "Delta Modulation". IEEE Spectrum, Octubre

Schwartz, M., 1980. "Transmisión de Información Modulación y Ruido". Mc Graw-Hill, México.

Sklar, B., 1983. "A Structured Overview of Digital Communications". Part I IEEE communications magazine.

Sklar, B., 1983. "A Structured Overview of Digital Communications".Part II IEEE communications magazine.

Apendice A. Programa Fuente

```
PROGRAM simula;
(*                  UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

                        FACULTAD DE INGENIERIA

    INGENIERIA EN COMPUTACION E INGENIERIA MECANICA ELECTRICA Y ELECTRONICA

            "SIMULACION DEL FORMATO Y CODIFICACION DE LA FUENTE
                DE UN SISTEMA DE COMUNICACIONES DIGITALES"

          INTEGRANTES:
                    BARRON TREJO ROBERTO
                    ROSAS RAMIREZ C. SERGIO
                    VILLAMIL CORDOVA SALVADOR
                    VILLARREAL RODRIGUEZ MARIO
*)

CONST
   n = 8;
TYPE
   cadenas  = PACKED ARRAY[1..n] OF CHAR;
   datos    = ARRAY      [1..512]  OF REAL;
VAR
   da,db,dc,dd,de          : datos;
   si,ele,eli,elu,d,e,ya   : CHAR;
   bitent,bitsal           : cadenas;
   existe,uno              : BOOLEAN;
   archivo,entrada,salida  : FILE OF INTEGER;
   archivo5                : FILE OF cadenas;
   nombre1,nombre2,nombre  : STRING[12];
   m_anterior,m_actual,yi,numero1,anterior,vi,i,cont: INTEGER;
   sal1,numero,j,x,y,x1,y1,xh,y0,zy,zx,im,ny,nx,y2  : INTEGER;

PROCEDURE mensaje;
   BEGIN
   GOTOXY(31,24);  WRITE('Oprima cualquier tecla para continuar');
   REPEAT  UNTIL  KEYPRESSED;
   TEXTMODE;
   END;

PROCEDURE ejes1;
   VAR
      i: INTEGER;
   BEGIN
   FOR i:=1 TO 20 DO
      BEGIN
      IF i<17 THEN
         DRAW(22,167-i*10,26,167-i*10,1);
      DRAW(24+25*i,85,24+25*i,89,1);
      END;
   DRAW(24,0,24,174,1);  DRAW(24,174,536,174,1);
   DRAW(536,0,536,174,1);  DRAW(24,0,536,0,1);
```

```
   DRAW(24,87,536,87,1);
   GOTOXY(1,3);  WRITE('70');
   GOTOXY(1,11);  WRITE('0');
   GOTOXY(1,20);  WRITE('-70');
   END;

PROCEDURE checa(existe:BOOLEAN;VAR si:CHAR);
   BEGIN
   si:=' ';
   IF NOT(existe) THEN
      BEGIN
      WRITELN;
      WRITE('El archivo no existe; vuelve a intentarlo ? (S/N)');
      READ(KBD,si);
      END;
   END;

PROCEDURE genera;
{Genera un archivo de enteros como senal de prueba, y muestra su forma}
{graficamente}
VAR
   i,magnitud: INTEGER;
   expansion,seal: REAL;
   ss: CHAR;

   PROCEDURE senoide;
   {Se construye una senal senoidal con una amplitud y frecuencia variable}
      BEGIN
      REPEAT
         CLRSCR;  WRITELN;  WRITELN(' ':33,'SENAL SENOIDAL');  WRITELN;
         WRITE('Magnitud (0-87): ');  READLN(magnitud);
      UNTIL (magnitud>=0) AND (magnitud<=87);
      REPEAT
         CLRSCR;  WRITELN;  WRITELN(' ':33,'SENAL SENOIDAL');  WRITELN;
         WRITE('Factor de expansión (0.02-0.1): ');  READLN(expansion);
      UNTIL (expansion>=0.02) AND (expansion<=0.1);
      x:=24;  x1:=23;  m_anterior:=0;
      WRITE(entrada,m_anterior);
      HIRES;  HIRESCOLOR(11);  ejes1;
      FOR i:=0 TO 511 DO
         BEGIN
         m_actual:=ROUND(magnitud*SIN(expansion*i));
         DRAW(x1,87-m_anterior,x,87-m_actual,1);
         WRITE(entrada,m_actual);
         x1:=x;  x:=x+1;  m_anterior:=m_actual;
         END;
      END;

   PROCEDURE aleatoria;
   {Se genera una senal con distribución aleatoria}
      BEGIN
      WRITELN;  WRITELN(' ':32,'SENAL ALEATORIA');  WRITELN;
      HIRES;  HIRESCOLOR(11);  ejes1;
      x:=24;  x1:=23;  m_anterior:=ROUND(RANDOM*127);
```

```
    WRITE(entrada,m_anterior);
    FOR i:=0 TO 511 DO
       BEGIN
       REPEAT
          m_actual:=ROUND(RANDOM*87);
       UNTIL ABS(m_anterior-m_actual) < 15;
       WRITE(entrada,m_actual);  DRAW(x1,87-m_anterior,x,87-m_actual,1);
       x1:=x;  x:=x+1;  m_anterior:=m_actual;
       END;
    END;

PROCEDURE usuario;
    BEGIN
    cont:=0;
    WHILE m_actual<>9999 DO
       BEGIN
       CLRSCR;  WRITELN(' ':30,'DATOS DEL USUARIO');  WRITELN;
       WRITE('Dame dato ',cont+1,' en el rango (-87,87) ó 9999');
       WRITE(' para Terminar ');  READLN(m_actual);
       IF (m_actual>=-87) AND (m_actual<=87) THEN
          BEGIN
          cont:=cont+1;  WRITE(entrada,m_actual);
          END
       ELSE
          IF m_actual<>9999 THEN
             WRITELN('Sólo valores en el rango (-87,87), 9999=FIN');
       END;
    m_actual:=0;
    FOR i:=cont TO 512 DO
       WRITE(entrada,m_actual);
    CLOSE(entrada);  RESET(entrada);
    HIRES;  HIRESCOLOR(11);  ejes1;  x:=24;  x1:=23;
    READ(entrada,m_anterior);
    WHILE (NOT(EOF(entrada))) DO
       BEGIN
       READ(entrada,m_actual);  DRAW(x1,87-m_anterior,x,87-m_actual,1);
       x1:=x;  x:=x+1;  m_anterior:=m_actual;
       END;
    END;
BEGIN
CLRSCR;  WRITELN;  WRITELN(' ':29,'GENERA SENAL DE PRUEBA');  WRITELN;
WRITE('Nombre del archivo a generar: ');  READLN(nombre);
ASSIGN(entrada,nombre);  REWRITE(entrada);
REPEAT
    CLRSCR;  WRITELN;  WRITELN(' ':29,'GENERA SENAL DE PRUEBA');
    WRITELN;  WRITELN('Tipo de senal a generar: ');  WRITELN;
    WRITELN('(S) Senoidal,  (A) Aleatoria  (D) Datos del usuario');
    READ(KBD,ss);
UNTIL ss IN ['s','S','d','D','a','A'];
CASE ss OF
's','S': senoide;
'a','A': aleatoria;
'd','D': usuario;
END;
```

```
    CLOSE(entrada);
    GOTOXY(4,24);  WRITE('Senal de prueba');  mensaje;
    END;

PROCEDURE muestreo;
{El programa realiza el muestreo lineal de una senal senoidal,}
{aleatoria o de cualquier tipo que proponga el usuario.}
    VAR
       pos,lambda: REAL;
       i,j,k,te: INTEGER;
       mn,ma: ARRAY[0..512] OF INTEGER;
    BEGIN
    x:=24;  x1:=23;
    CLRSCR;  WRITELN;  WRITELN(' ':35,'MUESTREO');  WRITELN;
    REPEAT
       WRITE('Archivo a Leer: ');
       READLN(nombre);
       ASSIGN(entrada,nombre);
       {$I-} RESET(entrada) {$I+};
       existe:=(IOresult=0);
       checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
       BEGIN
       WRITELN;  WRITE('Nombre del archivo de salida: ');  READLN(nombre);
       ASSIGN(salida,nombre);  REWRITE(salida);
       i:= 0;
       WHILE NOT EOF(entrada) AND (i<=512) DO
          BEGIN
          READ(entrada,ma[i]);  i:=i+1;
          END;
       WRITELN;  WRITELN('Lambda es el valor al que se desea muestrear ');
       WRITELN;  WRITE('Dame el valor de lambda (0.1 - 1.5)   lambda = ');
       READLN(lambda);
       CLRSCR;  WRITELN;
       FOR i:=0 TO 512 DO
          BEGIN
          pos:=i*lambda;  j:=TRUNC(pos);
          IF FRAC(pos)=0 THEN
             mn[i]:=ma[j]
          ELSE
             mn[i]:=ROUND(ma[j]+FRAC(pos)*(-ma[j]+ma[j+1]));
          END;
       IF lambda>1 THEN
          te:=ROUND(512/lambda)
       ELSE
          te:=512;
       HIRES;  HIRESCOLOR(11);  ejes1;  y1:=mn[0];
       FOR i:=1 TO te-1 DO
          BEGIN
          DRAW(x1,87-y1,x,87-mn[i],1);  x1:=x;  y1:=mn[i];  x:=x+1;
          END;
       IF lambda>1 THEN
          FOR i:=te TO 512 DO
```

```
            BEGIN
            an[i]:=0;  DRAW(x1,y1,x,87,1);  x1:=x;  y1:=57;  x:=x+1;
            END;
      GOTOXY(4,24);  WRITE('Muestreo');  mensaje;
      FOR i:=0 TO 512 DO
         WRITE(salida,an[i]);
      CLOSE(salida);
      END;
   END;

PROCEDURE cuantiza;
{cuantiza de manera: Uniforme, No lineal por la Ley f y por la Ley A}
{los valores de f y A, son los tipicos para un sistema de comunicación}
   CONST
      mu=255;  a=100;
   VAR
      lectura,rango,h,max,niveles,escritura:INTEGER;
      nivel,datox,datol,mudiv,adiv:REAL;
      primval,negativo:BOOLEAN;
      opcion: CHAR;
   BEGIN
   CLRSCR;  WRITELN;  WRITELN(' ':25,'CUANTIZACION (Compansión)');  WRITELN;
   mudiv:=LN(1+mu);  adiv:=1+LN(a);
   REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre1);
      ASSIGN(entrada,nombre1);
      {$I-} RESET(entrada) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
   UNTIL (existe) OR ((si='n') OR (si='N'));
   IF existe THEN
      BEGIN
      WRITE('Número de niveles (4 - 128 ): ');
      REPEAT
         READ(niveles);
      UNTIL (niveles>=4)  AND (niveles<=128);
      WRITELN;  WRITE('Archivo a almacenar datos cuantizados: ');
      READLN(nombre2);  WRITELN;
      WRITELN(' ':30, '1) Uniforme');
      WRITELN(' ':30, '2) Ley f');
      WRITELN(' ':30, '3) Ley A');  WRITELN;
      WRITE(' ':33,'Opción deseada: ');
      REPEAT
         opcion:=' ';
         READ(KBD,opcion);  WRITE(opcion);
      UNTIL opcion IN ['1','2','3'];
      ASSIGN(salida,nombre2 );  REWRITE(salida);
      ASSIGN(entrada,nombre1 );  RESET(entrada);
      max:=0;
      FOR h:=1 TO 512 DO
         BEGIN
         READ(entrada,h);
         IF h>max THEN  max:=h;
```

```
      END;
    rango:=2*max;  nivel:=rango/niveles;  RESET(entrada);
    x:=24;  x1:=23;  HIRES;  HIRESCOLOR(f1);  ejes1;  primval:=TRUE;
    GOTOXY(4,24);
    CASE opcion OF
    '1': WRITE('Cuantizacion Uniforme');
    '2': WRITE('Cuantizacion Ley Mu');
    '3': WRITE('Cuantizacion Ley A');
    END;
    WHILE NOT EOF(entrada) DO
      BEGIN
      negativo:=FALSE;  READ(entrada,lectura);
      IF opcion='1' THEN
        dato1:=lectura/nivel
      ELSE
        dato1:=lectura/max;
      IF lectura<0 THEN
        BEGIN
        negativo:=TRUE;  dato1:=ABS(dato1);
        END;
      CASE opcion OF
      '1': escritura:=ROUND(dato1)*ROUND(nivel);
      '2': BEGIN
           datox:=LN(1+(mu*dato1))/mudiv;  datox:=datox*(niveles DIV 2);
           escritura:=ROUND(datox);  escritura:=escritura*ROUND(nivel);
           END;
      '3 : BEGIN
           IF dato1<=0.01 THEN
             datox:=(a*dato1)/adiv
           ELSE
             datox:=(1+LN(a*dato1))/adiv;
           datox:=datox*(niveles DIV 2 );
           escritura:=ROUND(datox);  escritura:=escritura*ROUND(nivel);
           END;
      END;
      IF negativo THEN  escritura:=escritura*(-1);
      IF primval THEN
        BEGIN
        y1:=escritura;  primval:=FALSE;  WRITE(salida,escritura)
        END
      ELSE
        BEGIN
        DRAW(x1,87-y1,x,87-escritura,1);  x1:=x;  y1:=escritura;  x:=x+1;
        WRITE(salida,escritura);
        END;
      END;
    mensaje;  CLOSE(entrada);  CLOSE(salida);
    END;
  END;

PROCEDURE decuantiza;
{decuantiza de las senales cuantizadas por la Ley f y por la Ley A}
  CONST
    mu=255;  a=100;
```

```
VAR
   h,max,lectura,datoy: INTEGER;
   mu,datox,datol,dat,adiv: REAL;
   primval,negativo: BOOLEAN;
   opcion: CHAR;
BEGIN
CLRSCR;  WRITELN;  WRITELN(' ':25,'CUANTIZACION (Expansión)');  WRITELN;
mu:=LN(1+mu);  adiv:=1+LN(a);
REPEAT
   WRITE('Archivo a Leer: ');
   READLN(nombre1);
   ASSIGN(entrada,nombre1);
   {$I-} RESET(entrada) {$I+};
   existe:=(IOresult=0);
   checa(existe,si);
UNTIL (existe) OR ((si='n') OR (si='N'));
IF existe THEN
   BEGIN
   WRITELN;  WRITE('Archivo a almacenar datos cuantizados: ');
   READLN(nombre2);  WRITELN;
   WRITELN(' ':26, '1) Ley f');
   WRITELN(' ':26, '2) Ley A');  WRITELN;
   WRITE  (' ':29, 'opción deseada: ');
   REPEAT
      opcion:=' ';
      READ(KBD,opcion);  WRITE(opcion);
   UNTIL opcion IN [ '1','2'];
   ASSIGN(salida, nombre2 );  REWRITE( salida );
   ASSIGN( entrada, nombre1);  RESET(entrada);  max:=0;
   FOR h:=1 TO 512 DO
      BEGIN
      READ(entrada,h);
      IF h>max THEN  max:=h;
      END;
   RESET(entrada);
   x:=24;  x1:=23;  HIRES;  HIRESCOLOR(11);  ejes1;  primval:=TRUE;
   GOTOXY(4,24);
   CASE opcion of
   '1': WRITE('Ley Mu (Expansion)');
   '2': WRITE('Ley A (Expansion)');
   END;
   WHILE NOT EOF(entrada) DO
      BEGIN
      negativo:=FALSE;  READ(entrada,lectura);  datol:=lectura/max;
      IF lectura<0 THEN
         BEGIN
         negativo:=true;  datol:=ABS(datol);
         END;
      CASE opcion OF
      '1': BEGIN
           dat:=mu*datol;  datox:=(EXP(dat)-1) / mu;  datox:=datox*max;
           END;
      '2': BEGIN
           IF datol<0.01 THEN
```

```
                datox:=(datol*adiv)/a
              ELSE
                datox:=EXP(datol*adiv-1)/a;
              datox:=datox*max;
              END;
          END;
          datoy:=ROUND(datox);
          IF negativo THEN  datoy:=datoy*(-1);
          IF primval THEN
             BEGIN
             y1:=datoy;  primval:=FALSE;  WRITE(salida,datoy)
             END
          ELSE
             BEGIN
             DRAW(x1,87-y1,x,87-datoy,1);  x1:=x;  y1:=datoy;  x:=x+1;
             WRITE(salida,datoy);
             END;
          END;
       mensaje;  CLOSE(entrada);  CLOSE(salida);
       END;
    END;

PROCEDURE letr(n1:INTEGER;VAR letra:cadenas);
{Tranforma un numero entero a secuencia de bits}
    VAR
       i,res: INTEGER;
    BEGIN
    i:=8;
    REPEAT
       res:=n1 MOD 2;  letra[i]:=CHR(48+res);  n1:=n1 DIV 2;  i:=i-1;
    UNTIL n1=0;
    WHILE i>=1 DO
       BEGIN
       letra[i]:='0';  res:=0;  i:=i-1;
       END;
    END;

PROCEDURE comp2(VAR letra:cadenas);
{Transforma una secuencia de bits en su complemento a dos}
    VAR
       i: INTEGER;
    BEGIN
    i:=n+1;
    REPEAT
       i:=i-1;
    UNTIL letra[i]='1';
    i:=i-1;
    WHILE i>=1 DO
       BEGIN
       IF (letra[i]='0') THEN
          letra[i]:='1'
       ELSE
          letra[i]:='0';
       i:=i-1;
```

```
    END;
  END;

PROCEDURE pulsos(letra:cadenas;VAR ya:CHAR);
  VAR
    i,j: INTEGER;
  BEGIN
  ya:='1';
  FOR i:=1 TO 7 DO
    BEGIN
    IF letra[i]='1' THEN
      BEGIN
      IF uno=false THEN
        BEGIN
        uno:=TRUE;  DRAW(xh,y1,xh,y0,1);
        END;
      DRAW(xh,y1,xh+5,y1,1);
      IF letra[i+1]='0' THEN
        DRAW(xh+5,y1,xh+5,y0,1)
      ELSE
        DRAW(xh+5,y1+9,xh+5,y1+11,1);
      END
    ELSE
      BEGIN
      IF uno=true THEN
        BEGIN
        uno:=false;  DRAW(xh,y1,xh,y0,1);
        END;
      DRAW(xh,y0,xh+5,y0,1);
      IF letra[i+1]='1' THEN
        DRAW(xh+5,y0,xh-5,y1,1)
      ELSE
        DRAW(xh+5,y1+9,xh+5,y1+11,1);
      END;
    xh:=xh+5;
    END;
  IF letra[8]='1' THEN
    BEGIN
    uno:=TRUE;  DRAW(xh,y1,xh+5,y1,1);   DRAW(xh+5,y1+9,xh+5,y1+11,1);
    END
  ELSE
    BEGIN
    uno:=FALSE;  DRAW(xh,y0,xh+5,y0,1);  DRAW(xh+5,y1+9,xh+5,y1+11,1);
    END;
  DRAW(xh+5,y1-3,xh+5,y1,1);  xh:=xh+5;  GOTOXY(ny,nx);  WRITE(sal1);
  ny:=ny+5;
  IF xh>500 THEN
    BEGIN
    GOTOXY(70,1);  WRITE('MUESTRA');  GOTOXY(zy,zx);  WRITE(im,' - ',im+12);
    xh:=0;  y0:=y0+48;  y1:=y1+48; zx:=zx+6;  im:=im+13;  nx:=nx+6;  ny:=3;
    END;
  IF y0>200 THEN
    BEGIN
    GOTOXY(28,24);  WRITE('<1> Para continuar  <2> Para terminar');
```

```
      READ(KBD,ya);
      HIRES; HIRESCOLOR(11); y0:=30; y1:=20; xh:=0; zx:=4; ny:=3; nx:=2;
      END;
   END;

PROCEDURE condensado;
   BEGIN
   CLRSCR; WRITELN(' ':35,'DESPLIEGUE EN BITS'); WRITELN; xh:=0;
   REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(entrada,nombre);
      {$I-} RESET(entrada) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
   UNTIL (existe) OR ((si='n') OR (si='N'));
   IF existe THEN
      BEGIN
      HIRES; HIRESCOLOR(11); y2:=0; y0:=30; y1:=20; uno:=true; ya:='1';
      zy:=68; zx:=4; lm:=0; ny:=3; nx:=2;
      WHILE NOT EOF(entrada) AND (y2=0) DO
         BEGIN
         READ(entrada,sal1);
         IF sal1<0 THEN
            BEGIN
            sal1:=ABS(sal1); letr(sal1,bitsal); comp2(bitsal); sal1:=-sal1;
            END
         ELSE letr(sal1,bitsal);
         IF ya<>'2' THEN
            pulsos(bitsal,ya)
         ELSE y2:=1;
         END;
      IF ya<>'2' THEN
         BEGIN
         GOTOXY(70,1); WRITE('MUESTRA'); GOTOXY(zy,zx); WRITE(lm,' - 512');
         GOTOXY(4,24); WRITE('PCM'); mensaje;
         END;
      END;
   END;

PROCEDURE depcm;
   BEGIN
   x:=24; x1:=23; CLRSCR; WRITELN; WRITELN(' ':34,'DECODIFICACION PCM');
   REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(entrada,nombre);
      {$I-} RESET(entrada) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
   UNTIL (existe) OR ((si='n') OR (si='N'));
   IF existe THEN
      BEGIN
      HIRES; HIRESCOLOR(11); ejes1;
```

```
      READ(entrada,m_actual);  y1:=m_actual;
      WHILE NOT EOF(entrada) DO
        BEGIN
        READ(entrada,m_actual);  DRAW(x1,87-y1,x,87-m_actual,1);
        x1:=x;  y1:=m_actual;  x:=x+1;
        END;
      GOTOXY(4,24);  WRITE('Decodificacion PCM');  mensaje;
      END;
  END;

PROCEDURE codigo(co:CHAR);
VAR
  y0,y1,y2,xh,bit,i,cont,w1,w2,w3,w4,w5,k: INTEGER;
  fue,fue1,fue2,fue3,fue4,toca: STRING[7];

PROCEDURE escribe;
  BEGIN
  CASE co OF
  '1': WRITELN('RZ');
  '2': WRITELN('RB');
  '3': WRITELN('AMI (BRZ)');
  '4': WRITELN('Fase Partida (Manchester)');
  '5': WRITELN('Fase partida (marca)');
  '6': WRITELN('NRZ (L)');
  '7': WRITELN('NRZ (M)');
  '8': WRITELN('NRZ (S)');
  '9': WRITELN('Retraso de modulación (Miller)');
  END;
  END;

PROCEDURE lineas;
  VAR
    j,k:INTEGER;
  BEGIN
  k:=0;
  FOR j:=1 TO 16 DO
    BEGIN
    DRAW(k,y0-8,k,y0-2,1);  k:=k+40;
    END;
  DRAW(320,y0-8,320,y2+8,1);
  END;

PROCEDURE dibuja(bit:integer;metodo:CHAR);

  PROCEDURE cual(v:INTEGER);
    BEGIN
    CASE v OF
    1: BEGIN
      DRAW(xh,y1,xh,y0,1);  DRAW(xh,y0,xh+20,y0,1);
      DRAW(xh+20,y0,xh+20,y1,1);  DRAW(xh+20,y1,xh+40,y1,1);
      END;
    3: BEGIN
      DRAW(xh,y1,xh,y2,1);  DRAW(xh,y2,xh+20,y2,1);
      DRAW(xh+20,y2,xh+20,y1,1);  DRAW(xh+20,y1,xh+40,y1,1);
```

```
        END;
    4: BEGIN
        DRAW(xh,y0,xh+20,y0,1);
        DRAW(xh+20,y0,xh+20,y1,1);  DRAW(xh+20,y1,xh+40,y1,1);
        END;
    5: BEGIN
        DRAW(xh,y1,xh+20,y1,1);  DRAW(xh+20,y1,xh+20,y0,1);
        DRAW(xh+20,y0,xh+40,y0,1);
        END;
    7: BEGIN
        DRAW(xh,y1,xh,y0,1);  DRAW(xh,y0,xh+40,y0,1);
        END;
    9: BEGIN
        DRAW(xh,y0,xh,y1,1);  DRAW(xh,y1,xh+40,y1,1);
        END;
        END;
    END;
BEGIN
CASE metodo OF
'1': IF bit=1 THEN
        cual(1)
     ELSE
        DRAW(xh,y1,xh+40,y1,1);
'2': IF bit=1 THEN
        cual(1)
     ELSE
        cual(3);
'3': CASE bit OF
     0: DRAW(xh,y1,xh+40,y1,1);
     1: IF toca='arriba' THEN
           BEGIN
           toca:='abajo';  cual(1);
           END
        ELSE
           BEGIN
           toca:='arriba';  cual(3);
           END;
     END;
'4': IF bit=1 THEN
        BEGIN
        IF fuel='uno' THEN
           DRAW(xh,y1,xh,y0,1);
        cual(4);  fuel:='uno';
        END
     ELSE
        BEGIN
        IF fuel='cero' THEN
           DRAW(xh,y0,xh,y1,1);
        cual(5);  fuel:='cero';
        END;
'5': CASE BIT OF
     1: IF fue='unolz' THEN
           BEGIN
           fue:='unode';  cual(5)
```

```
            END
          ELSE
            IF fue='unode' THEN
              BEGIN
              fue:='unoiz'; cual(4);
              END
            ELSE
              IF fue='ceroiz' THEN
                BEGIN
                fue:='unode'; cual(5);
                END
             ELSE
                BEGIN
                fue:='unoiz'; cual(4);
                END;
      0: IF fue='unoiz' THEN
           BEGIN
           fue:='ceroiz'; DRAW(xh,y1,xh,y0,1); cual(4);
           END
         ELSE
            IF fue='unode' THEN
              BEGIN
              fue:='cerode'; DRAW(xh,y1,xh,y0,1); cual(5);
              END
            ELSE
              IF fue='ceroiz' THEN
                BEGIN
                fue:='ceroiz'; DRAW(xh,y1,xh,y0,1); cual(4);
                END
              ELSE
                BEGIN
                fue:='cerode'; DRAW(xh,y1,xh,y0,1); cual(5);
                END;
    END;
  '6': CASE BIT OF
      1: IF fue2='uno' THEN
           BEGIN
           fue2:='uno'; DRAW(xh,y0,xh+40,y0,1);
           END
         ELSE
            IF fue2='cero' THEN
              BEGIN
              fue2:='uno'; cual(7);
              END;
      0: IF fue2='cero' THEN
           BEGIN
           fue2:='cero'; DRAW(xh,y1,xh+40,y1,1);
           END
         ELSE
            IF fue2='uno' THEN
              BEGIN
              fue2:='cero'; cual(9);
              END;
    END;
```

```
'7': CASE BIT OF
    1: IF fue3='arriba' THEN
         BEGIN
         fue3:='abajo';  cual(9);
         END
       ELSE
         BEGIN
         fue3:='arriba';  cual(7);
         END;
    0: IF fue3='arriba' THEN
         BEGIN
         fue3:='arriba';  DRAW(xh,y0,xh+40,y0,1);
         END
       ELSE
         BEGIN
         fue3:='abajo';  DRAW(xh,y1,xh+40,y1,1);
         END;
    END;
'8': CASE BIT OF
    0: IF fue3='arriba' THEN
         BEGIN
         fue3:='abajo';  cual(9);
         END
       ELSE
         BEGIN
         fue3:='arriba';  cual(7);
         END;
    1: IF fue3='arriba' THEN
         BEGIN
         fue3:='arriba';  DRAW(xh,y0,xh+40,y0,1);
         END
       ELSE
         BEGIN
         fue3:='abajo';  DRAW(xh,y1,xh+40,y1,1);
         END;
    END;
'9': CASE BIT OF
    1: IF fue4='unoiz' THEN
         BEGIN
         fue4:='unode';  cual(5);
         END
       ELSE
          IF fue4='unodo' THEN
            BEGIN
            fue4:='unoiz';  cual(4);
            END
        ELSE
            IF fue4='coroar' THEN
              BEGIN
              fue4:='unoiz';  cual(4);
              END
            ELSE
              BEGIN
              fue4:='unode';  cual(5);
```

```
                END;
      0: IF fue4='unoiz' THEN
           BEGIN
           fue4:='ceroab';  DRAW(xh,y1,xh+40,y1,1);
           END
         ELSE
           IF fue4='unode' THEN
             BEGIN
             fue4:='ceroar';  DRAW(xh,y0,xh+40,y0,1);
             END
           ELSE
             IF fue4='ceroar' THEN
               BEGIN
               fue4:='ceroab';  cual(9);
               END
             ELSE
               BEGIN
               fue4:='ceroar';  cual(7);
               END;
      END;
      END;
  xh:=xh+40;
  END;
  BEGIN
  CLRSCR;  WRITELN;  WRITE(' ':21,'CODIGO ');  escribe;
  REPEAT
    WRITE('Archivo a Leer ? ');
    READLN(nombre);
    ASSIGN(entrada,nombre);
    {$I-} RESET(entrada) {$I+};
    existe:=(IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    cont:=1;  ya:='1';  toca:='arriba';  fue:='unode';  fue1:='cero';
    fue2:='cero';  fue3:='abajo';  fue4:='unoiz';
    IF (co>='3') AND (co<>'4') AND (co<>'6') THEN
      BEGIN
      WRITELN('Que bit viene antes del primero transmitido');
      CASE co OF
      '3': BEGIN
           WRITELN('<0> CERO <1> UNO ');  READ(w5);
           IF w5=0 THEN  toca:='abajo';
           END;
      '4': BEGIN
           WRITELN('<0> CERO <1> UNO ');  READ(w5);
           IF w5=1 THEN  fue1:='uno';
           END;
      '5': BEGIN
           WRITELN('<0> CERODE <1> CEROIZ <2> UNODE <3> UNOIZ');  READ(w5);
           CASE w5 OF
           0: fue:='CERODE';
           1: fue:='CEROIZ';
```

```
            2: fue:='UNODE';
            3: fue:='UNOIZ';
            END;
            END;
        '6': BEGIN
            WRITELN('<0> CERO <1> UNO '); READ(w5);
            IF w5=1 THEN fue2:='UNO';
            END;
        '7','8': BEGIN
            WRITELN('<0> CERO <1> UNO '); READ(w5);
            IF w5=1 THEN fue3:='UNO';
            END;
        '9': BEGIN
            WRITELN('<0> CERODE <1> CEROIZ <2> UNODE <3> UNOIZ'); READ(w5);
            CASE w5 OF
            0: fue:='CERODE';
            1: fue:='CEROIZ';
            2: fue:='UNODE';
            3: fue:='UNOIZ';
            END;
            END;
        END;
        END;
    WHILE NOT EOF(entrada) AND (ya<>'2') DO
        BEGIN
        y0:=40; y1:=50; y2:=60; xh:=0; HIRES; HIRESCOLOR(11);
        GOTOXY(30,1); WRITE('CODIGO '); escribe; w2:=4;
        FOR w3:=1 TO 2 DO
            BEGIN
            w1:=1;
            FOR k:=1 TO 2 DO
                BEGIN
                IF NOT EOF(entrada) THEN
                    BEGIN
                    READ(entrada,numero1); letr(numero1,bitsal); lineas;
                    FOR i:=1 TO 8 DO
                        IF (bitsal[i]='1') THEN dibuja(1,co)
                        ELSE dibuja(0,co);
                    GOTOXY(w1,w2);
                    FOR i:=1 TO 8 DO
                        WRITE(' ',bitsal[i],' ');
                    GOTOXY(w1+12,w2+6);
                    WRITE('Muestra: ',cont,'    Valor: ',numero1);
                    cont:=cont+1; w1:=w1+40;
                    END;
                END;
            y0:=y0+90; y1:=y1+90; y2:=Y2+90; xh:=0; w2:=w2+11;
            END;
        GOTOXY(28,24); WRITE('<1> Para continuar <2> Para terminar');
        READ(kbd,ya);
        END;
    TEXTMODE;
    END;
END;
```

```
PROCEDURE dpcm;
{Transforma archivo en código PCM a archivo en código DPCM}
{o archivo en código DPCM a archivo en código PCM}
  BEGIN
  x:=24;  x1:=23;  xh:=0;  CLRSCR;  WRITELN;
  WRITELN(' ':35,'CODIFICACION DPCM');  WRITELN;
  REPEAT
    WRITE('Archivo a Leer: ');
    READLN(nombre);
    ASSIGN(entrada,nombre);
    {$I-} RESET(entrada) {$I+};
    existe:=(IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    READ(entrada,anterior);  WRITELN;
    WRITE('Nombre del archivo de salida (DPCM) ');  READLN(nombre);
    ASSIGN(salida,nombre);  REWRITE(salida);
    HIRES;  HIRESCOLOR(11);  y2:=0;  y0:=30;  y1:=20;  uno:=true;  ya:='1';
    zy:=68;  zx:=4;  im:=1;  ny:=3;  nx:=2;
    WRITE(salida,anterior);
    WHILE NOT EOF(entrada) DO
      BEGIN
      READ(entrada,numero1);  sal1:=numero1-anterior;
      IF sal1<0 THEN
        BEGIN
        sal1:=ABS(sal1);  letr(sal1,bitsal);  comp2(bitsal);  sal1:=-sal1;
        WRITE(salida,sal1);
        END
      ELSE
        BEGIN
        letr(sal1,bitsal);  WRITE(salida,sal1);
        END;
      IF ya<>'2' THEN
        pulsos(bitsal,ya)
      ELSE
        BEGIN
        GOTOXY(1,23);  WRITE('Procesando ...');
        END;
      anterior := numero1;
      END;
    IF ya<>'2' THEN
      BEGIN
      GOTOXY(4,24);  WRITE('DPCM');  mensaje;
      END;
    CLOSE(salida);
    END;
  END;

PROCEDURE dedpcm;
  BEGIN
```

```
    x:=24;  x1:=23;  CLRSCR;  WRITELN;  WRITELN(' ':34,'DECODIFICACION DPCM');
    REPEAT
       WRITE('Archivo a Leer: ');
       READLN(nombre);
       ASSIGN(entrada,nombre);
       {$I-} RESET(entrada) {$I+};
       existe:=(IOresult=0);
       checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
       BEGIN
       READ(entrada,v1);
       WRITE('Nombre del archivo donde pondré la senal reconstruida: ');
       READLN(nombre);  ASSIGN(salida,nombre);  REWRITE(salida);
       HIRES;  HIRESCOLOR(11);  ejes1;
       y1:=v1;  WRITE(salida,v1);
       WHILE NOT EOF(entrada) DO
          BEGIN
          READ(entrada,m_actual);  v1:=v1+m_actual;  DRAW(x1,87-y1,x,87-v1,1);
          x1:=x;  y1:=v1;  x:=x+1;  WRITE(salida,v1);
          END;
       CLOSE(salida);  GOTOXY(4,24);  WRITE('DPCM');  mensaje;
       END;
    END;

PROCEDURE delta(inversa:integer);
    VAR
       k,base,escritura,kx,ent,ent2,sal: INTEGER;
       primval: BOOLEAN;
    BEGIN
    CLRSCR; WRITELN;  WRITELN(' ':30,'MODULACION DELTA');  WRITELN;


    REPEAT
       WRITE('Archivo a Leer: ');
       READLN(nombre1);
       ASSIGN(entrada,nombre1);
       {$I-} RESET(entrada) {$I+};
       existe:=(IOresult=0);
       checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
       BEGIN
       READ(entrada,ent);  base:=ent;  WRITELN;
       WRITE('Nombre del archivo de salida: ');  READLN(nombre2);
       ASSIGN(salida,nombre2);  REWRITE(salida);
       k:=2;  primval:=TRUE;  x:=24;  x1:=23;
       HIRES;  HIRESCOLOR(11);  ejes1;  GOTOXY(4,24); WRITE('Modulacion Delta');
       WHILE NOT EOF(entrada) DO
          BEGIN
          READ(entrada,ent2);
          IF inversa=0 THEN
             BEGIN
             sal:=ent2-base;
```

```
            IF sal>=0 THEN
              escritura:=k
            ELSE
              escritura:=-k;
            base:=base+escritura;
            END
          ELSE
            BEGIN
            base:=base+ent2;  escritura:=base;
            END;
          IF primval THEN
            BEGIN
            y1:=base;  primval:=FALSE;  WRITE(salida,escritura)
            END
          ELSE
            BEGIN
            DRAW(x1,87-y1,x,87-base,1);  x1:=x;  y1:=base;  x:=x+1;
            WRITE(salida,escritura);
            END;
          END;
        WRITE(salida,escritura);  CLOSE(salida);  CLOSE(entrada);  mensaje;
        END;
    END;

PROCEDURE deltadoble(inversa:integer);
    VAR
      k1,k2,escritura,base,ent,ent2,sal: INTEGER;
      primval: BOOLEAN;
    BEGIN
    CLRSCR;  WRITELN;  WRITELN(' ':25,'MODUALCION DELTA DOBLE PASO');  WRITELN;
    REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre1);
      ASSIGN(entrada,nombre1);
      {$I-} RESET(entrada) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
      BEGIN
      READ(entrada, ent);  base:=ent;  WRITELN;
      WRITE('Nombre del archivo de salida: ');  READLN(nombre2);
      ASSIGN(salida,nombre2);  REWRITE(salida);
      k1:=2;  k2:=4;  primval:=TRUE;  x:=23;  x1:=24;
      HIRES;  HIRESCOLOR(11);  ejes1;
      GOTOXY(4,24);  WRITE('Delta doble paso');
      WHILE NOT EOF(entrada) DO
        BEGIN
        READ(entrada,ent2);
        IF inversa=0 THEN
          BEGIN
          sal:=ent2-base;
          IF SAL>0 THEN
            IF sal<=K1 THEN
```

```
                escritura:=k1
              ELSE
                escritura:=k2
            ELSE
              IF sal>=-K1 THEN
                escritura:=-k1
              ELSE
                escritura:=-k2;
            base:=base+escritura;
            END
          ELSE
            BEGIN
            base:=base+ent2;  escritura:=base;
            END;
          IF primval THEN
            BEGIN
            y1:=base;  primval:=FALSE;  WRITE(salida,escritura)
            END
          ELSE
            BEGIN
            DRAW(x1,87-y1,x,87-base,1);  x1:=x;  y1:=base;  x:=x+1;
            WRITE(salida,escritura);
            END;
          END;
      mensaje;  WRITE(salida,escritura);  CLOSE(salida);  CLOSE(entrada);
      END;
  END;

PROCEDURE duobin;
{el programa realiza la senalizacion duobinaria de una senal digital}
  BEGIN
  CLRSCR;  WRITELN;  WRITELN(' ':25,'CODIFICACION DUOBINARIA');  WRITELN;
  REPEAT
    WRITE('Archivo a Leer: ');
    READLN(nombre);
    ASSIGN(entrada,nombre);
    {$I-} RESET(entrada) {$I+};
    existe:=(IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    WRITELN;  WRITE('Nombre del archivo de salida: ');  READ(nombre);
    ASSIGN(salida,nombre);  REWRITE(salida);
    CLRSCR;  WRITELN('El mensaje codificado es: ');  WRITELN;  i:=1;  x1:=-1;
    WHILE NOT EOF(entrada) DO
      BEGIN
      READ(entrada,x);  y:=x+x1;  x1:=x;  WRITE(y,' ');  WRITE(salida,y);
      i:=i+1;
      END;
    REPEAT UNTIL KEYPRESSED;
    CLOSE(salida);
    END;
  END;
```

```
PROCEDURE deduobin;
{El programa realiza la decodificación de una senal transmitida en senali-}
{zación duobinaria}
  BEGIN
  CLRSCR;  WRITELN;  WRITELN(' ':25,'DECODIFICACION DUOBINARIA');  WRITELN;
  REPEAT
    WRITE('Archivo a Leer: ');
    READLN(nombre);
    ASSIGN(entrada,nombre);
    {$I-} reset(entrada) {$I+};
    existe := (IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    WRITELN;  WRITE('Nombre del archivo de salida: ');  READ(nombre);
    ASSIGN(salida,nombre);  REWRITE(salida);
    CLRSCR;  WRITELN;  WRITELN('El mensaje original es: ');  WRITELN;
    i:=1;  x1:=-1;
    WHILE NOT(EOF(entrada)) DO
      BEGIN
      READ(entrada,x);  y:=x-x1;  x1:=y;  WRITE(y,' ');  WRITE(salida,y);
      i:=i+1;
      END;
    REPEAT UNTIL KEYPRESSED;
    CLOSE(salida);
    END;
  END;

PROCEDURE duobinmo;
{El programa realiza la senalización duobinaria modificada }
{de una senal digital}
  VAR
    x,y: ARRAY[0..516] OF INTEGER;
    i,n: INTEGER;
    bin,fic: FILE OF INTEGER;
  BEGIN
  CLRSCR;  WRITELN;  WRITELN(' ':25,'DUOBINARIO MODIFICADO');  n:= 516;
  WRITELN;
  REPEAT
    WRITE('Archivo a Leer: ');
    READLN(nombre);
    ASSIGN(bin,nombre);
    {$I-} RESET(bin) {$I+};
    existe:=(IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    i:=4;
    WHILE NOT EOF(bin) AND (i<=n) DO
      BEGIN
```

```
      READ(bin,x[i]);  {lee}
      END;
    CLRSCR;  WRITELN;  WRITELN( El mensaje codificado es: );  WRITELN;
    x[0]:= 0;  x[1]:= 1;  x[2]:= 0;  x[3]:= 0;
    FOR i:=2 TO n DO
      BEGIN
      y[i]:=x[i]-x[i-2];  WRITE(y[i],' ');
      END;
    WRITELN;  WRITELN;  WRITE('Nombre del archivo de salida: ');
    READLN(nombre);
    ASSIGN(fic,nombre);  REWRITE(fic);
    FOR i:= 2 TO n DO
      WRITE(fic,y[i]);
    CLOSE(fic);
    END;
  END;

PROCEDURE deduobinso;
{El programa realiza la decodificacion de una senal transaltida en senali-}
{zación duobinaria modificada}
  VAR
    x,y,z: ARRAY[0..517] OF INTEGER;
    i,n: INTEGER;
    bar,ara: FILE OF INTEGER;
  BEGIN
  CLRSCR;  WRITELN;  WRITELN(' ':20,'DECODIFICACION DUOBINARIA MODIFICADA');
  n:= 517;
  WRITELN;
  REPEAT
    WRITE('Archivo a Leer: ');
    READLN(nombre);
    ASSIGN(bar,nombre);
    {$I-} RESET(bar) {$I+};
    existe:=(IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    i:=4;
    WHILE NOT EOF (bar) AND (i<=n) DO
      BEGIN
      READ(bar,y[i]);  i:=i+1;
      END;
    CLRSCR;  WRITELN;  WRITELN('El mensaje original es : ');  WRITELN;
    FOR i:=4 TO n DO
      BEGIN
      IF y[i]=1 THEN
        BEGIN
        z[i]:=1;  WRITE(z[i],' ');
        END
      ELSE
        IF y[i]=-1 THEN
          BEGIN
          z[i]:=0;  WRITE(z[i],' ');
```

```
            END
          ELSE
            BEGIN
            z[i]:=z[i-2];  WRITE(z[i],' ');
            END;
        END;
    WRITELN;  WRITELN;  WRITE('Nombre del archivo de salida: ');
    READLN(nombre);
    ASSIGN(ara,nombre);  REWRITE(ara);
    FOR i:=6 TO n DO
      BEGIN
      x[i]:=z[i];  WRITE(ara,x[i]);
      END;
    CLOSE(ara);
    END;
  END;

PROCEDURE pduobin;
{El programa realiza la precodificación duobinaria de una senal digital}
  VAR
    a,x,y,w,b,d,z: ARRAY[0..512] OF INTEGER;
    i,k,n,j: INTEGER;
    nom: STRING[10];
    mensaje,res,cor,dor: FILE OF INTEGER;
  BEGIN
  CLRSCR;  WRITELN;  WRITELN(' ':26,'PRECODIFICACION DUOBINARIA');  n:= 512;
  WRITELN;
  REPEAT
    WRITE('Archivo a Leer: ');
    READLN(nombre);
    ASSIGN(mensaje,nombre);
    {$I-} RESET(mensaje) {$I+};
    existe:=(IOresult=0);
    checa(existe,si);
  UNTIL (existe) OR ((si='n') OR (si='N'));
  IF existe THEN
    BEGIN
    k:=0;
    WHILE NOT EOF(mensaje) AND (k<=n) DO
      BEGIN
      READ(mensaje,d[k]);  k:=k+1;
      END;
    nombre:='res.2';
    ASSIGN(res,nombre);  RESET(res);  k:=0;
    WHILE NOT EOF(res) AND (k<=n) DO
      BEGIN
      READ(res,y[k]);  k:=k+1;
      END;
    WRITELN;
    FOR k:=0 TO n DO
      REPEAT
        IF d[k]=1 THEN
          BEGIN
          IF y[k]=1 THEN
```

```
                x[k]:=0
              ELSE
                IF y[k]=0 THEN
                  x[k]:=1;
              END
            ELSE
              IF d[k]=0 THEN
                BEGIN
                IF y[k]=1 THEN
                  x[k]:=1
                ELSE
                  IF y[k]=0 THEN
                  x[k]:=0;
                END
          UNTIL n=512;
        CLRSCR;  WRITELN;  WRITELN('El mensaje codificado es: ');  WRITELN;
        FOR k:=0 TO n-1 DO
          BEGIN
          z[k]:=x[k]+y[k];  WRITE(z[k],' ');
          END;
        WRITELN;  WRITELN;  WRITE('Nombre del archivo de salida: ');
        READLN(nombre);
        ASSIGN(dor,nombre);  REWRITE(dor);
        FOR k:=0 TO n-1 DO
          BEGIN
          y[k]:=z[k];  WRITE(dor,y[k]);
          END;
        CLOSE(dor);
        END;
    END;

PROCEDURE depreduob;
{El programa realiza la decodificación de una señal transmitida en precodi-}
{ficación duobinaria}
    VAR
      y,x: ARRAY[0..512] OF INTEGER;
      n,i: INTEGER;
      rob,mar: FILE OF INTEGER;
    BEGIN
    CLRSCR;  WRITELN;  WRITELN(' ':28,'PRECODIFICACION DUOBINARIA');  n:= 512;
    REPEAT
        WRITE('Archivo a Leer: ');
        READLN(nombre);
        ASSIGN(rob,nombre);
        {$I-} RESET(rob) {$I+};
        existe:=(IOresult=0);
        checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
      BEGIN
      WRITELN;  i:= 0;
      WHILE NOT EOF (rob) AND (i<=n) DO
        BEGIN
        READ(rob,y[i]);  i:=i+1;
```

```
        END;
      CLRSCR;  WRITELN;  WRITELN('El mensaje original es:');  WRITELN;
      FOR i:=0 TO n-1 DO
        IF y[i]=1 THEN
          x[i]:= 1
        ELSE
          IF (y[i]=0) OR (y[i]=2) THEN
            x[i]:=0;
      WRITE(x[i],' ');
      WRITELN;  WRITELN;  WRITE('Nombre del archivo de salida: ');
      READLN(nombre);
      ASSIGN(mar,nombre);  REWRITE(mar);
      FOR i:=0 TO n-1 DO
        WRITE(mar,x[i]);
      CLOSE(mar);
      END;
    END;

PROCEDURE pduobinazo;
{El programa realiza la codificación duobinaria modificada de una senal}
{digital}
    VAR
      a,x,y,w,d,z,b: ARRAY[0..512] OF INTEGER;
      i,k,n: INTEGER;
      tre,res,dan: FILE OF INTEGER;
    BEGIN
    CLRSCR;  WRITELN;  WRITELN(' ':26,'CODIFICACION DUOBINARIA MODIFICADA');
    WRITELN;  n:= 512;
    REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(tre,nombre);
      {$I-} RESET(tre) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
    UNTIL (existe) OR ((si='N') OR (si='N'));
    IF existe THEN
      BEGIN
      k:= 0;
      WHILE NOT EOF(tre) AND (k<=n) DO
        BEGIN
        READ(tre,a[k]);  k:=k+1;
        END;
      nombre:='res.2';
      ASSIGN(res,nombre);  RESET(res);  k:=0;
      WHILE NOT EOF(res) AND (k<=n) DO
        BEGIN
        READ(res,y[k]);  k:=k+1;
        END;
      FOR k:=0 TO n DO
        REPEAT
        IF d[k]=0 THEN
          BEGIN
          IF y[k]=1 THEN
```

```
            x[k]:=1
          ELSE
            IF y[k]=0 THEN
              x[k]:=0
          END
        ELSE
          IF d[k]=1 THEN
            BEGIN
            IF y[k]=0 THEN
              x[k]:=1
            ELSE
              IF y[k]=1 THEN
                x[k]:=0
            END;
        UNTIL n=512;
      CLRSCR;  WRITELN;  WRITELN('El mensaje codificado es: ');  WRITELN;
      FOR k:=0 TO n-1 DO
        BEGIN
        z[k]:=y[k]-x[k];   WRITE(z[k],' ');
        END;
      WRITELN;  WRITELN;  WRITE('Nombre del archivo de salida: ');
      READLN(nombre);
      ASSIGN(dan,nombre);  REWRITE(dan);
      FOR k:=0 TO n-1 DO
        BEGIN
        y[k]:=z[k];  WRITE(dan,y[k]);
        END;
      CLOSE(dan);
      END;
    END;

PROCEDURE decpreduobmo;
{El programa realiza la decodificación de una senal transmitida en codifi-}
{cación duobinaria modificada}
    VAR
      x,y: ARRAY[0..512] OF INTEGER;
      n,i: INTEGER;
      pau,dol: FILE OF INTEGER;
    BEGIN
    CLRSCR;  WRITELN;
    WRITELN('DECODIFICACION DE LA PRECODIFICACION DUOBINARIA MODIFICADA');
    WRITELN;  n:= 512;
    REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(pau,nombre);
      {$I-} RESET(pau) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
      BEGIN
      i:=0;
      WHILE NOT EOF (pau) AND (i<=n) DO
```

```
        BEGIN
        READ(pau,y[i]);  i:= i + 1;
        END;
      CLRSCR;  WRITELN;  WRITELN('El mensaje original es: ');  WRITELN;
      FOR i:=0 TO n-1 DO
        BEGIN
        x[i]:=ABS(y[i]);  WRITE(x[i],' ');
        END;
      WRITELN;  WRITELN;  WRITE('Nombre del archivo de salida: ');
      READLN(nombre);
      ASSIGN(dol,nombre);  REWRITE(dol);
      FOR i:=0 TO n-1 DO
        WRITE(dol,x[i]);
      CLOSE(dol);
      END;
  END;

PROCEDURE menu(p:INTEGER);
  BEGIN
  CLRSCR;  WRITELN;
  IF p=0 THEN
    WRITELN(' ':26,'CODIFICACION DE CARACTERES')
  ELSE
    WRITELN(' ':26,'DECODIFICACION DE CARACTERES');
  WRITELN;  WRITELN;
  IF p=0 THEN
    WRITELN(' ':26,'1) Genera archivo de texto');
  WRITELN(' ':26,'2) ASCII');
  WRITELN(' ':26,'3) EBCDIC');
  WRITELN(' ':26,'4) Código interno');
  WRITELN(' ':26,'0) Regresa al menu anterior');
  WRITELN;
  WRITE(' ':29,'Opción deseada: ');
  READ(kbd,d);  WRITE(d);
  END;

PROCEDURE conv;
{Este programa convierte caracteres en códigos binarios de 8 bits}
  VAR
    i,j,x: INTEGER;
    w: CHAR;
    texto: FILE OF CHAR;
    g: ARRAY [0..125,0..7] OF INTEGER;

  PROCEDURE createxto;
    VAR
      az: FILE OF CHAR;
      b: STRING[80];
    BEGIN
    CLRSCR;  WRITELN;  WRITELN;  WRITELN(' ':29,'GENERA ARCHIVO DE TEXTO');
    WRITELN;  WRITE('Archivo a Crear: ');  READLN(nombre);
    ASSIGN(az,nombre);  REWRITE(az);  WRITELN;
    WRITELN('Proporcione el texto a codificar');  READLN(b);
    FOR i:=1 TO LENGTH(b) DO
```

```
      WRITE(az,b[i]);
   CLOSE(az);
   END;

PROCEDURE codificaascii(iii:INTEGER);
   BEGIN
   CLRSCR;  WRITELN;  WRITELN;
   CASE iii OF
   1: WRITELN(' ':29,'CODIFICACION ASCII');
   2: WRITELN(' ':29,'CODIFICACION EBCDIC');
   3: WRITELN(' ':29,'CODIGO INTERNO');
   END;
   WRITELN;
   REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(texto,nombre);
      {$I-} reset(texto) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
   UNTIL (existe) OR ((si<>'n') OR (si='N'));
   IF existe THEN
      BEGIN
      WRITELN;
      WRITELN('Escribe un nombre de archivo para almacenar lo codificado');
      READLN(nombre);  CLRSCR;  WRITELN;
      ASSIGN(salida,nombre);  REWRITE(salida);
      CASE iii OF
      1: nombre:='ASCII123';
      2: nombre:='EBCDIC21';
      3: nombre:='CODOINT';
      END;
      ASSIGN(entrada,nombre);  RESET(entrada);  CLRSCR;
      FOR i:=0 to 125 do
         FOR j:=0 to 7 do
            BEGIN
            READ(entrada,i);  g[i,j]:=i;
            END;
      cont:=1;
      WHILE NOT EOF(texto) DO
         BEGIN
         READ(texto,w);  x:=ORD(w);  WRITE(w,' = ');
         FOR i:=0 to 7 do
            BEGIN
            WRITE(salida,g[x,i]);  WRITE(g[x,i]);
            END;
         IF cont<5  THEN
            BEGIN
            cont:=cont+1;  WRITE('  ');
            END
         ELSE
            BEGIN
            cont:=1;  WRITELN;
            END;
```

```
            END;
          mensaje;  CLOSE(salida);
          END;
      END;
  BEGIN
  CLRSCR;
  REPEAT
    menu(0);
    CASE d OF
    '1': createxto;
    '2': codificaascii(1);
    '3': codificaascii(2);
    '4': codificaascii(3);
    END;
  UNTIL d in ['0'];
  END;

PROCEDURE reconv;
(Este programa convierte textos codificados en ASCII, EBCDIC, CODIGO IN-)
(TERNO de 8 bits, a la forma original del texto.)
  VAR
     i,j,n,e,a: INTEGER;
     p: ARRAY[0..1000] OF INTEGER;

  PROCEDURE ascii;
     BEGIN
     CLRSCR;  WRITELN;  WRITELN;  WRITELN(' ':29,'DECODIFICACION ASCII');
     WRITELN;
     REPEAT
        WRITE('Archivo a Leer: ');
        READLN(nombre);
        ASSIGN(entrada,nombre);
        {$I-} reset(entrada) {$I+};
        existe:=(IOresult=0);
        checa(existe,si);
     UNTIL (existe) OR ((si='n') OR (si='N'));
     IF existe THEN
        BEGIN
        CLRSCR;  j:=0;
        WHILE NOT EOF(entrada) AND (j<512) DO
           BEGIN
           READ(entrada,n);  p[j]:=n;  j:=j+1;
           END;
        j:=0;
        WRITELN;  WRITELN('TEXTO DECODIFICADO');  WRITELN;
        REPEAT
           e:=p[j]*128+p[j+1]*64+p[j+2]*32+p[j+3]*16;
           e:=e+p[j+4]*8+p[j+5]*4+p[j+6]*2+p[j+7]*1; a:=e;
           WRITE(CHAR(a));  j:=j+8;
        UNTIL j=512;
        mensaje;
        END;
     END;
```

```
PROCEDURE codigointerno;
   BEGIN
   CLRSCR;  WRITELN;  WRITELN;
   WRITELN(' ':29,'DECODIFICACION CODIGO INTERNO');  WRITELN;
   REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(entrada,nombre);
      {$I-} reset(entrada) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
   UNTIL (existe) OR ((si='n') OR (si='N'));
   IF existe THEN
      BEGIN
      CLRSCR;  j:=0;
      WHILE NOT EOF(entrada) AND (j<512) DO
         BEGIN
         READ(entrada,n);
         p[j]:=n;  j:=j+1;
         END;
      j:=0;
      WRITELN;  WRITELN('TEXTO DECODIFICADO');  WRITELN;
      REPEAT
         a:=p[j]*128+p[j+1]*64+p[j+2]*32+p[j+3]*16;
         a:=a+p[j+4]*8+p[j+5]*4+p[j+6]*2+p[j+7]*1; a:=a;
         IF (a>=0) AND (a<=9) THEN
            WRITE(a);
         CASE a OF
         11: WRITE(CHAR(a+50));
         16: WRITE(CHAR(a+27));
         27: WRITE(CHAR(a+19));
         28: WRITE(CHAR(a+13));
         32: WRITE(CHAR(a+13));
         43: WRITE(CHAR(a-7));
         44: WRITE(CHAR(a-2));
         48: WRITE(CHAR(a-16));
         49: WRITE(CHAR(a-2));
         59: WRITE(CHAR(a-15));
         60: WRITE(CHAR(a-20));
         END;
         IF (a>=17) AND (a<=25) THEN
            WRITE(CHAR(a+48));
         IF (a>=33) AND (a<=41) THEN
            WRITE(CHAR(a+41));
         IF (a>=50) AND (a<=57) THEN
            WRITE(CHAR(a+33));
            j:=j+8;
      UNTIL j=512;
      mensaje;
      END;
END;
```

```
PROCEDURE ebcdic;
   BEGIN
   CLRSCR;  WRITELN;  WRITELN;  WRITELN(' ':29,'DECODIFICACION EBCDIC');
   WRITELN;
   REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(entrada,nombre);
      {$I-} reset(entrada) {$I+};
      existe:=(IOresult=0);
      choca(existe,si);
   UNTIL (existe) OR ((si='n') OR (si='N'));
   IF existe THEN
      BEGIN
      CLRSCR;  j:=0;
      WHILE NOT EOF(entrada) AND (j<512) DO
         BEGIN
         READ(entrada,n);  p[j]:=n;  j:=j+1;
         END;
      j:=0;
      WRITELN;  WRITELN(' ':29,'TEXTO DECODIFICADO');  WRITELN;
      REPEAT
         e:=p[j]*128+p[j+1]*64+p[j+2]*32+p[j+3]*16;
         e:=e+p[j+4]*8+p[j+5]*4+p[j+6]*2+p[j+7]*1; a:=e;
         CASE a OF
         64: a:=a-32;
         75: a:=a-29;
         76: a:=a-16;
         77: a:=a-37;
         78: a:=a-35;
         80: a:=a-42;
         90: a:=a-57;
         91: a:=a-55;
         93: a:=a-52;
         94: a:=a-35;
         96: a:=a-51;
         107: a:=a-63;
         108: a:=a-71;
         109: a:=a-14;
         122: a:=a-64;
         123: a:=a-88;
         124: a:=a-60;
         126: a:=a-65;
         127: a:=a-93;
         92,97: a:=a-50;
         110,111: a:=a-48;
         END;
         IF (a>=129) AND (a<=137) THEN
            a:=a-32;
         IF (a>=145) AND (a<=153) THEN
            a:=a-39;
         IF (a>=162) AND (a<=169) THEN
```

```
            a:=a-47;
          IF (a>=193) AND (a<=201) THEN
            a:=a-128;
          IF (a>=209) AND (a<=217) THEN
            a:=a-135;
          IF (a>=226) AND (a<=233) THEN
            a:=a-143;
          IF (a>=240) AND (a<=249) THEN
            a:=a-192;
          WRITE(CHAR(a));  j:=j+8;
        UNTIL j=512;
        mensaje;
        END;
    END;
  BEGIN
  CLRSCR;
  REPEAT
    menu(1);
    CASE d OF
    '2': ascii;
    '3': ebcdic;
    '4': codigointerno;
    END;
  UNTIL d IN ['0'];
  END;

PROCEDURE lectura(VAR da,db,dc,dd:datos);
  VAR
    elo,g:CHAR;

  PROCEDURE LeeArchivo(VAR d:datos);
    VAR
      lectura:INTEGER;
    BEGIN
    FOR i:=1 to 512 do
      BEGIN
      READ(archivo,lectura);  d[i]:=lectura;
      END;
    END;
  BEGIN
  elo:='y';
  WHILE (elo<>'N') AND (elo<>'n') DO
    BEGIN
    CLRSCR;  WRITELN;  WRITELN(' ':30,'LECTURA DE DATOS EN DISCO');
    WRITELN;
    REPEAT
      WRITE('Archivo a Leer: ');
      READLN(nombre);
      ASSIGN(archivo,nombre);
      {$I-} reset(archivo) {$I+};
      existe:=(IOresult=0);
      checa(existe,si);
    UNTIL (existe) OR ((si='n') OR (si='N'));
    IF existe THEN
```

```
        BEGIN
        WRITELN;
        REPEAT
          WRITE('En qué arreglo desea depositar los datos (A,B,C,D): ');
          READ(kbd,g);  WRITELN(g);
          UNTIL g in ['A','a','B','b','C','c','D','d'];
        CASE g of
        'A','a': LeeArchivo(da);
        'B','b': LeeArchivo(db);
        'C','c': LeeArchivo(dc);
        'D','d': LeeArchivo(dd);
        END;
        END;
      CLOSE(archivo);  WRITELN;  WRITE('Desea leer otra senal (S/N): ');
      READ(kbd,elo);
      END;
    END;

PROCEDURE Despliegue(VAR da,db,dc,dd:datos);
  VAR
    miny,maxy: REAL;
    elo,f,s,g,h: CHAR;

  PROCEDURE despl(VAR d:datos);
    VAR
      eschor,rango,escala: REAL;
      nuerpun: INTEGER;

    PROCEDURE minmax(VAR miny,maxy:REAL);
      VAR
        i:INTEGER;
      BEGIN
      miny:=1E10+36;  maxy:=-miny;
      FOR i:=1 TO nuerpun DO
        BEGIN
        IF d[i]<miny THEN miny:=d[i];
        IF d[i]>maxy THEN maxy:=d[i];
        END;
      END;

    PROCEDURE ejes;
      VAR
        i:INTEGER;
      BEGIN
      FOR i:=1 TO 20 DO
        BEGIN
        IF i<17 THEN
          DRAW(22,174-i*10,26,174-i*10,1);
        DRAW(24+25*i,35,24+25*i,89,1);
        END;
      CASE s of
      'N',
      'n': BEGIN
           DRAW(24,0,24,174,1);  DRAW(24,174,24+nuerpun,174,1);
```

```
        DRAW(24+numpun,0,24+numpun,174,1);  DRAW(24,0,24+numpun,0,1);
        DRAW(24,87,536,87,1);
        END;
   'S',
   's': BEGIN
        DRAW(24,0,24,174,1);  DRAW(24,174,536,174,1);
        DRAW(536,0,536,174,1); DRAW(24,0,535,0,1);
        DRAW(24,87,536,87,1);
        END;
   END;
   END;
BEGIN
WRITELN;
REPEAT
   WRITE('Desea escalamiento automático (S/N): ');
   READ(kbd,g);  WRITELN(g);
UNTIL g in ['S','s','N','n'];
REPEAT
   WRITELN;
   WRITE('Desea graficar todo el arreglo (S/N): ');
   READ(kbd,h);
   WRITELN(h);
UNTIL h IN ['S','s','N','n'];
CASE h OF
'S',
's': BEGIN
     numpun:=512;  s:='S';  eschor:=1.0;
     END;
'N',
'n': BEGIN
     REPEAT
        WRITELN;  WRITE('Cuántos puntos desea graficar (0-511): ');
        READLN(numpun);
     UNTIL (numpun>=0) OR (numpun<=511);
     REPEAT
        WRITELN;
        WRITE('Desea expansion de la escala horizontal (S/N): ');
        READ(kbd,s);
     UNTIL s in ['S','s','N','n'];
     CASE s OF
     'S','s': eschor:=512/numpun;
     'N','n': eschor:=1;
     END;
     END;
END;
CLRSCR;  HIRES;  HIRESCOLOR(11);  ejes;
CASE g OF
'S',
's': BEGIN
     minmax(miny,maxy);  rango:=maxy-miny;
     IF rango<>0 THEN  escala:=174/rango
     ELSE  escala:=1;
     FOR i:=1 TO numpun-1 DO
        DRAW(ROUND((i*eschor)+23),ROUND(174-(d[i]-miny)*escala),
```

```
                ROUND(((i+1)*escbor)+23),ROUND(174-(d[i+1]-miny)*escala),1);
          END;
      'N',
      'n': BEGIN
          FOR i:=1 TO numpun-1 DO
            DRAW(i+23,ROUND(174-d[i]*0.04),i+24,ROUND(174-d[i+1]*0.04),1);
          END;
      END;
      mensaje;
      END;
  BEGIN
  elo:='y';
  WHILE (elo<>'N') AND (elo<>'n') DO
    BEGIN
    CLRSCR;  WRITELN;  WRITELN(' ':28,'DESPLIEGUE DE DATOS EN PANTALLA');
    REPEAT
      WRITELN;  WRITE('Qué arreglo desea desplegar (A,B,C,D): ');
      READ(kbd,f);  WRITELN(f);
    UNTIL f IN ['A','a','B','b','C','c','D','d'];
    CASE f OF
    'A','a': despl(da);
    'B','b': despl(db);
    'C','c': despl(dc);
    'D','d': despl(dd);
    END;
    WRITELN;  WRITE('Desea desplegar otra senal (S/N): ');  READ(kbd,elo);
    END;
  END;

PROCEDURE formamenu(par:INTEGER);
  BEGIN
  CLRSCR;  WRITELN;  WRITELN;
  IF par=1 THEN
    WRITELN(' ':33,'CODIFICACION')
  ELSE
    WRITELN(' ':31,'DECODIFICACION');
  WRITELN;  WRITELN;
  WRITELN(' ':22,'1) PCM');
  WRITELN(' ':22,'2) DPCM');
  WRITELN(' ':22,'3) Modulación delta');
  WRITELN(' ':22,'4) Modulación delta doble paso');
  WRITELN(' ':22,'5) Codificación de textos');
  WRITELN(' ':22,'6) Senalización duobinaria');
  WRITELN(' ':22,'7) Senalización duobinaria modificada');
  WRITELN(' ':22,'8) Precodificación duobinaria');
  WRITELN(' ':22,'9) Precodificación duobinaria codificada');
  WRITELN(' ':22,'0) Regresar al menú anterior');
  WRITELN;
  WRITE(' ':25,'Opción deseada: ');
  END;
BEGIN
ele:='y';
WHILE ele<>'z' DO
  BEGIN
```

```
CLRSCR;  WRITELN;
WRITELN(' ':13,'Simulación del Formato y Codificación de la Fuente de un');
WRITELN(' ':25,'Sistema de Comunicaciones Digitales');  WRITELN;  WRITELN;
WRITELN(' ':25,'1) Genera senal de prueba');  WRITELN;
WRITELN(' ':25,'2) Muestreo');  WRITELN;
WRITELN(' ':25,'3) Cuantización (compansión)');  WRITELN;
WRITELN(' ':25,'4) Codificación');  WRITELN;
WRITELN(' ':25,'5) Decodificación');  WRITELN;
WRITELN(' ':25,'6) Cuantización (expansión)');  WRITELN;
WRITELN(' ':25,'7) Despliegue');  WRITELN;
WRITELN(' ':25,'0) Fin');  WRITELN;
WRITE(' ':28,'Opción deseada: ');
REPEAT
   READ(kbd,ele);
UNTIL ele IN ['1','2','3','4','5','6','7','0'];
CASE ele OF
'1': genera;
'2': muestreo;
'3': cuantiza;
'4': BEGIN
     eli:='y';
     WHILE eli<>'z' DO
        BEGIN
        formamenu(1);
        REPEAT
           READ(kbd,eli);
        UNTIL eli IN ['1','2','3','4','5','6','7','8','9','0'];
        CASE eli OF
        '1': BEGIN
             elu:='y';
             WHILE elu<>'z' DO
                BEGIN
                CLRSCR; WRITELN; WRITELN;
                WRITELN(' ':21,'MODULACION POR CODIFICACION DE PULSOS');
                WRITELN;
                WRITELN(' ':21,'1) RZ');
                WRITELN(' ':21,'2) RB');
                WRITELN(' ':21,'3) AMI (BRZ)');


                WRITELN(' ':21,'4) Fase Partida (Manchester)');
                WRITELN(' ':21,'5) Fase Partida (marca)');
                WRITELN(' ':21,'6) NRZ (L)');
                WRITELN(' ':21,'7) NRZ (M)');
                WRITELN(' ':21,'8) NRZ (S)');
                WRITELN(' ':21,'9) Retraso de modulación (Miller)');
                WRITELN(' ':21,'0) Regresar al menú anterior');  WRITELN;
                WRITE(' ':24,'Opción deseada: ');
                REPEAT
                   READ(kbd,elu);
                UNTIL elu IN ['1','2','3','4','5','6','7','8','9','0'];
                IF elu='0' THEN  elu:='z'
                ELSE
                   IF elu='8' THEN
```

```
                    BEGIN
                    WRITELN;  WRITELN;
                    WRITE(' ':21,'1) 52 muestras por');
                    WRITELN(' pantalla (CONDENSADO)');
                    WRITE(' ':21,'2) 4 muestras por');
                    WRITELN(' pantalla (DETALLE)'); WRITELN;
                    WRITE(' ':25,'opción deseada: '); READ(kbd,ele);
                    IF ele='1' THEN condensado
                    ELSE  codigo(elu);
                    END
                  ELSE  codigo(elu);
              END;
          END;
        '2': dpcm;
        '3': delta(0);
        '4': deltadoble(0);
        '5': conv;
        '6': duobin;
        '7': duobinmo;
        '8': pduobin;
        '9': pduobinmo;
        '0': eli:='z';
        END;
        END;
    END;
'5': BEGIN
    eli:='y';
    WHILE eli<>'z' DO
       BEGIN
       formatmenu(0);
       REPEAT
          READ(kbd,eli);
       UNTIL eli IN ['1','2','3','4','5','6','7','8','9','0'];
       CASE eli OF
       '1': depcm;
       '2': dedpcm;
       '3': delta(1);
       '4': deltadoble(1);
       '5': reconv;
       '6': deduobin;
       '7': deduobinmo;
       '8': depreduob;
       '9': depreduobmo;
       '0': eli:='z';
       END;
       END;
    END;
'6': decuantiza;
'7': BEGIN
    eli:='y';
    WHILE eli<>'z' DO
       BEGIN
       CLRSCR;  WRITELN;
       WRITE('<1> Despliega en Bits <2> Despliega en gráfica <0> ');
```

```
            WRITELN('Regresar al menu anterior');
            REPEAT
               READ(kbd,eli);
            UNTIL eli IN ['1','2','0'];
            CASE eli OF
            '1': condensado;
            '2': BEGIN
                 lectura(da,db,dc,dd);
                 IF existe THEN
                    despliegue(da,db,dc,dd);
                 END;
            '0': eli:='z';
            END;
            END;
         END;
    '0': BEGIN
         CLRSCR; ele:='z';
         END;
  END;
  END;
END.
```

9 Apendice B Guia de operación

El programa SIMULA se encuentra almacenado en el diskette anexo a éste trabajo y su diseño está orientado para brindar al usuario la mayor facilidad de operación. Para ejecutarlo se teclea:

```
A>SIMULA <Enter>
```

Realizado lo anterior, se despliega el menú principal del programa. La operación del mismo, consiste en seleccionar una opción de éste menú.

Cuando el usuario requiera almacenar el programa en disco duro, debe asegurarse de que se encuentren presentes en el disco, algunos archivos auxiliares que permitirán la operación correcta del programa. A continuación se proporciona la lista de los archivos auxiliares:

```
RES.2
ASCII123
EBCDIC21
CCOODINT
```

Se utilizó el programa ALMACOD para generar los archivos ASCII123, EBCDIC21 y CCOODINT, cuyo código se muestra a continuación:

```
PROGRAM ALMACOD;
   VAR
   q,l,i,j,d,e:INTEGER;
   guard1:ARRAY[0..125,0..7] of INTEGER;
   nombre:STRING[10];
    trans:FILE of INTEGER;
        c:CHAR;

   BEGIN
   WRITE('ESCRIBE LOS BITS QUE CORRESPONDEN AL  CODIGO');
   WRITELN;
   WRITELN;
   WRITELN;
   q:=1;
   WRITE(j,'    ');
   FOR i:=0 to 125 DO
      BEGIN
         WRITE(j,' ');
         FOR i:=0 TO 7 DO
         BEGIN
         REPEAT
            READ(kbd,c)
         UNTIL c in['0','1'];
         VAL(c,d,e);
         WRITE(d);
```

```
            guard1[j,i]:=d;
         END;
      WRITELN;
      WRITELN;
      WRITELN;
      WRITELN('          EL VALOR DE "J" DE LA MATRIZ ES?');
      WRITELN;
      READ(j);
      WRITELN;
      WRITELN;
      WRITELN('          EL VALOR DE "I" DE LA MATRIZ ES?');
      WRITELN;
      READ(i);
      WRITELN;
      WRITELN;
      WRITELN('CUAL ES EL VALOR ENTERO DE LA MATRIZ');
      WRITELN('guard1[',i,',',j,']');
      WRITELN;
      READ(d);
      WRITELN;
      WRITELN;
      guard1[j,i]:=d;
         FOR i:=0 TO 7 DO
         BEGIN
            WRITE(guard1[j,i]);
         END;
      WRITELN;
      WRITELN;
      WRITELN('DESEA USTED HACER CORRECCIONES (S/N) ?');
      READ(kbd,c);
      WRITE(c);
      IF (c='N') OR (C='N') THEN  q:=0;
      END;
      WRITELN;
      WRITELN;
      WRITELN('BAJO QUE NOMBRE SE VA ALMACENAR EL CODIGO');
      WRITELN;
      WRITELN;
      READ(nombre);
      ASSIGN(trans,nombre);
      REWRITE(trans);
      FOR j:=0 TO 125 DO
      BEGIN
         FOR i:=0 TO 7 DO
            BEGIN
               d:=guard1[j,i];
               WRITE(trans,d);
            END;
       END;
       CLOSE(trans);
END.
```