UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA
ESCUELA DE INGENIERÍA ELECTRÓNICA

TRABAJO DE GRADUACION:

"ANALIZADOR DE TASACIÓN EN LÍNEAS DE TELEFONÍA FIJA"

PARA OPTAR AL GRADO DE:

INGENIERO EN TELECOMUNICACIONES

Presentado Por:
DAVID ENRIQUE TORRES CALDERÓN
JULIO BALTAZAR VENTURA PAREDES

Asesor:
ING. JUAN CARLOS CASTRO CHÁVEZ

MAYO 2010

SOYAPANGO – EL SALVADOR – CENTRO AMÉRICA

ÍNDICE

CONTENIDO **Página.**

CAPÍTULO I

INTRODUCCION

El teléfono es uno de los más grandes inventos del hombre, y uno de los más sencillos. Actualmente en nuestra sociedad es casi imposible encontrar a una persona adulta que no haya realizado al menos una llamada telefónica en su vida. Para la gran mayoría de las personas resulta indispensable poseer una línea telefónica, ya sea fija o móvil. Es tan importante el teléfono que la densidad telefónica (el número de líneas telefónicas por cada cien habitantes) es un indicador del progreso de una nación.

Según datos de la SIGET[1], en 2009 había en El Salvador casi 1.1 millones de líneas de telefonía fija, es decir una densidad telefónica de 15.7; y más de 7.56 millones de líneas de telefonía móvil, es decir una densidad telefónica de 108.1. En ambos casos se asume una población de 7 millones de habitantes. Si bien es cierto que hay casi siete veces más líneas de telefonía móvil que fija, eso no significa que la telefonía fija sea menos útil, ya que juega un rol muy importante en el desarrollo de las empresas.

En el presente capítulo se describe de manera básica el funcionamiento de la red de telefonía pública conmutada, así como el funcionamiento del teléfono mismo.

1.1 El Teléfono

En su forma más básica un teléfono consta de tres partes: un micrófono, un auricular y un interruptor para conectarse/desconectarse a la red telefónica. Un teléfono moderno tiene además un teclado de marcación, un generador de tonos para el teclado, un timbre que avisa cuando se recibe una llamada, un amplificador para la señal del micrófono, y una bobina híbrida (duplex coil en inglés). Por el mismo par de cable trenzado que conecta al teléfono con la red telefónica viajan las señales que van hacia el teléfono y las señales que salen de éste, por lo que una persona podría oír su propia voz al hacer una llamada; para evitar esto se

[1] Superintendencia General de Electricidad y Telecomunicaciones, "Boletín Estadístico 2009"

utiliza la bobina híbrida. En la figura 1.1 se muestra el circuito de un teléfono sin tomar en consideración el amplificador, ni filtros, ni circuitos de memoria para almacenar números telefónicos.

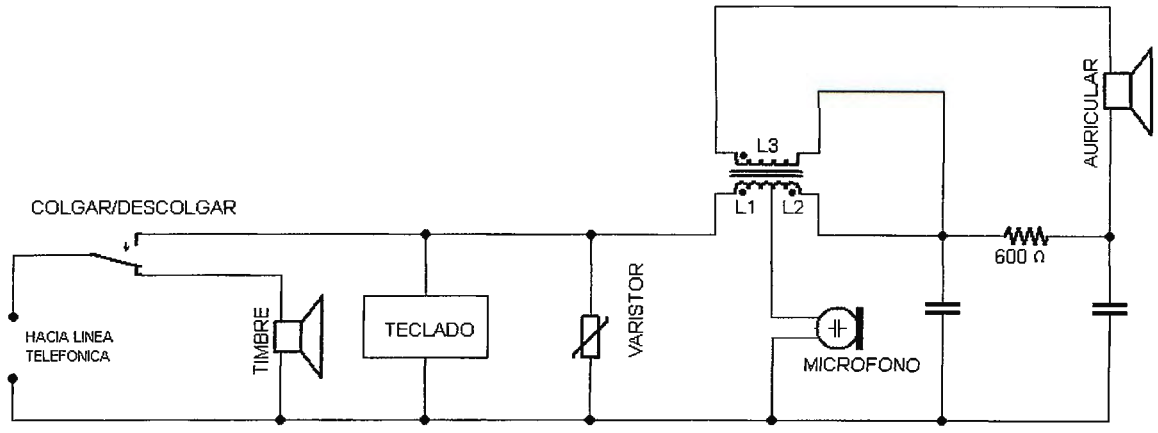


Figura 1.1 Circuito de teléfono.

En el circuito mostrado la bobina híbrida está formada por L1, L2 y L3. Cuando el usuario habla a través del micrófono, la señal eléctrica se reparte equitativamente entre L1 y L2. La señal de L1 va hacia la central telefónica, mientras que la de L2 se disipa en la resistencia de carga, y de esta manera no hay señal en L3 y por lo tanto tampoco en el auricular. Cuando entra señal hacia el teléfono, la corriente pasa por L1, y ésta induce una corriente igual sobre L2 evitando que pase corriente por el micrófono, pero ambas bobinas inducen corriente sobre L3 y así se activa el auricular. El varistor tiene como función mantener constante la polarización del micrófono. La impedancia de entrada de un teléfono varía entre 1.3 KΩ y 1.8 KΩ.

1.2 Red Telefónica Pública Conmutada

La red telefónica pública conmutada (Public Switched Telephone Network, o PSTN por sus siglas en inglés) es la red que enlaza a todas las redes públicas del mundo conmutadas por circuitos y sirve para ofrecer principalmente a los clientes el transporte de comunicaciones de voz.

Inicialmente la PSTN era analógica, pero en la actualidad es casi completamente digital e incluye tanto a la telefonía fija como a la telefonía móvil.

La PSTN utiliza conmutación de circuitos, en la cual se establece un canal dedicado (también llamado circuito) entre dos abonados que se comunican y, mientras dura la llamada, dicho canal no puede ser utilizado por otra comunicación, es decir que es exclusivo. Al terminar la llamada se libera el canal para que pueda ser utilizado por otro par de abonados. El circuito consiste en enlaces de cables que forman una trayectoria única durante el periodo de duración de la llamada entre un par de abonados. La conmutación de circuitos es idónea cuando se necesita que la información sea transmitida en tiempo real.

En la figura 1.2 se muestra un esquema de la red telefónica pública conmutada. El teléfono representa a un abonado. Cada abonado se identifica por un número telefónico único de ocho cifras según el plan de numeración de la siguiente manera[2]:

NDC	X	Y	Z	M	C	D	U
Código nacional de destino	Unidad de millón	Centena de millar	Decena de millar	Unidad de millar	Centena	Decena	Unidad

Tabla 1.1 Estructura de la numeración y atribución de cifras del plan de numeración para telefonía fija y telefonía móvil

El código nacional de destino indica si un número es de telefonía fija o de telefonía móvil. Las demás cifras, para telefonía fija, indican la zona geográfica del país en la que se encuentra la central a la que pertenece el número telefónico.

Los números telefónicos de cobro revertido constan de siete cifras, mientras que los de sobrecuota constan de once cifras.

---

[2] Superintendencia General de Electricidad y Telecomunicaciones, "Plan de Numeración Nacional"

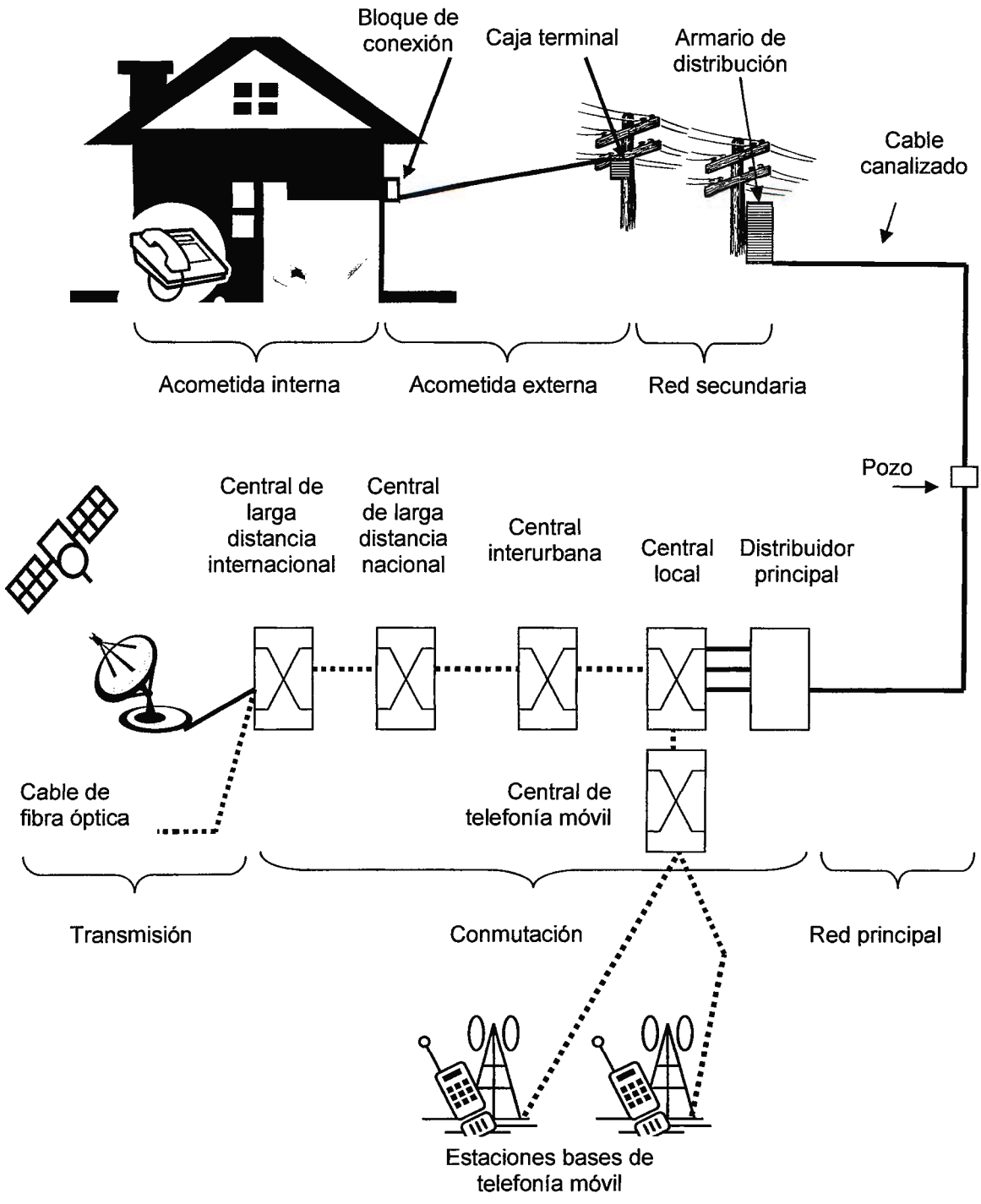


Figura 1.2 Esquema de la red telefónica pública conmutada

La acometida interior es el cable que se encuentra entre el bloque de conexión y la roseta. La roseta sirve de límite entre la acometida interior y el aparato terminal del abonado, el cual es conectado a la roseta por medio de un cable de par trenzado que tiene en sus extremos conectores RJ11.

El cable de par trenzado está formado por hilos de cobre recubiertos de plástico u otro aislante, y se trenzan en pares con el fin de evitar la diafonía, el ruido y otras interferencias.

La acometida exterior es el cable que conecta la caja terminal y el bloque de conexión. La caja terminal sirve de límite entre la red secundaria y la acometida exterior, y es posible conectar entre diez y veinte abonados. La red secundaria está formada por los cables que conectan el armario de distribución con las cajas terminales. Los cables de la red secundaria casi siempre son aéreos, aunque a veces son canalizados, y tienen desde 10 hasta 200 pares. El armario es el límite entre la red primaria y la red secundaria. El armario sirve a una zona determinada llamada distrito telefónico. En el armario se digitalizan las señales analógicas.

La voz tiene un rango de frecuencia de entre 300 y 3,400 hertz. La recomendación G.711 de la ITU-T indica que para digitalizar la señal analógica de la voz se usa una señal que muestrea la señal analógica 8,000 veces por segundo. Luego la señal muestreada es codificada con una resolución de 8 bits, obteniendo entonces una señal digital de 64,000 bits por segundo o 64 kbps. Luego las señales digitales de las conversaciones de los abonados son multiplexadas para optimizar el uso del ancho de banda de transmisión.

La red primaria (o red principal) está formada por los cables que unen el distribuidor principal de la central con el armario de distribución. Casi siempre la red primaria es canalizada porque los cables son de gran capacidad, teniendo entre 300 y 2,400 pares. El distribuidor principal está dentro de la central telefónica, y es el límite entre la planta interna y la planta externa. La planta

externa es el conjunto de elementos de la red que sirven para unir físicamente el distribuidor principal con el equipo terminal del abonado, y se llama externa porque dichos elementos están fuera de la central telefónica.

La central telefónica es donde se realizan las operaciones de conmutación necesarias para poder comunicar un abonado con otro. Las centrales telefónicas según su localización geográfica pueden clasificarse principalmente en:

- Centrales locales: son aquéllas en las que están conectados los abonados.
- Centrales urbanas: son aquéllas que conectan centrales locales de la misma ciudad. Pueden tener abonados.
- Centrales interurbanas: son aquéllas que conectan centrales locales de diferentes ciudades entre sí. Pueden tener abonados.
- Centrales de tránsito: son aquéllas utilizadas para conectar las centrales locales entre sí, pero no tienen abonados conectados. También se conocen como centrales tándem. Pueden ser urbanas o interurbanas y ofrecen rutas alternativas en caso de que haya falla en el enlace entre una central y otra.
- Centrales internacionales: son aquéllas que conectan la red telefónica nacional con las de otros países.

1.3 Sincronismo

Las redes digitales de telecomunicación necesitan estar sincronizadas para que no se degrade la calidad del servicio prestado al abonado. Cuando falla la sincronización hace que la información se distorsione, provocando pérdida de información. A esto se le conoce como deslizamiento.

Se necesita el sincronismo en cada red digital para mejorar la calidad de transmisión y conmutación, además de optimizar el uso de sus recursos. Las principales tecnologías que requieren de redes de sincronismo para su adecuada operación son las siguientes[3]:

[3] Symmetricom, "Soluciones de Sincronismo".

- Sistema de Señalización número 7 (SS7)
- Jerarquía Digital Síncrona (SDH, Synchronous Digital Hierarchy)
- Transmisión de Datos

Las redes de sincronismo constan de fuentes de referencia primaria y sistemas de distribución de sincronismo. Según la recomendación ITU-T G.811 la fuente de referencia primaria debe tener una precisión mejor que $1x10^{-11}$ con respecto al tiempo universal coordinado, por lo que se utiliza un reloj atómico de cesio 133, el cual no es radioactivo[4].

Los sistemas de distribución de sincronismo tienen como objetivo seleccionar, procesar y generar señales de temporización para sincronizar los elementos de las redes digitales, y para procesar dichas señales de temporización se utilizan relojes de rubidio o de cuarzo que satisfagan la recomendación ITU-T G.812. Según la recomendación ITU-T G.703 las señales de temporización deben ser codireccionales (en la misma dirección) a las señales de información; o ser contradireccionales cuando se tenga un equipo controlador y un equipo subordinado (en este caso la señal de temporización va siempre hacia el equipo subordinado).

1.4 Señalización

Se llama señalización al conjunto de órdenes e informaciones necesarias para el establecimiento, la supervisión y la finalización de una llamada telefónica. Mediante la señalización la central telefónica detecta que un abonado desea servicio, se le proporcionan los datos necesarios a la central para identificar al abonado con el cual desea comunicarse el primer abonado, y le avisa al segundo abonado que alguien quiere establecer una comunicación con él, enrutando luego la llamada telefónica.

---

[4] Oscilloquartz, "Manual de Fuente de Referencia Primaria modelo 5585".

Hay tres tipos de señalización:

- Señalización de abonado. Es la que se da entre el aparato telefónico del abonado y la central a la que está conectado. Se abordará este tema en el siguiente capítulo.
- Señalización interna de la central. Depende de cada fabricante.
- Señalización entre centrales. Este tipo de señalización se detalla a continuación.

La señalización entre centrales, según la banda en la que funciona, puede ser:

- Señalización en banda. Es cuando la señalización se hace a través del mismo canal que el de la señal de voz. Por ejemplo, la señalización digital multitono o multifrecuencia (abreviada DTMF, por sus siglas en inglés).
- Señalización fuera de banda. La señalización se hace a través de un canal especial que está separado de los canales de voz. Por ejemplo, el sistema de señalización número 7 (SS7).

La señalización entre centrales se caracteriza por su medio de transmisión, que puede ser:

- Señalización por canal asociado. Durante la comunicación se reserva un medio físico para transmitir la señalización. Esta señalización es sólo para un canal portador en particular.
- Señalización por canal común. Utiliza un canal que lleva información relacionada con varios canales portadores al mismo tiempo.

1.4.1 Señalización por Canal Asociado

La señalización por canal asociado se usa para intercambiar dos tipos de señalización:

- Señalización de línea. Se usa para conocer el estado de la línea o canal, por lo que sirve para establecer, supervisar, tasar y liberar la comunicación. Ejemplo: colgar/descolgar el teléfono.

- Señalización de registro. Lleva información de direccionamiento y otra información concerniente al abonado. Ejemplo: estado del abonado, tipo de tráfico, número telefónico, etc.

1.4.2 Señalización por Canal Común

Se hace la señalización por un medio de transmisión común para varias comunicaciones. Esto es posible porque la conmutación se hace de manera electrónica y no de manera mecánica o electromecánica, y así la central puede separar las señales de órdenes de las señales de voz. Un ejemplo de señalización por canal común es el sistema de señalización número 7 (SS7).

Un modo de explotación es la asociación entre la trayectoria que sigue un mensaje de señalización y la trayectoria que sigue la voz a la que se refiere el mensaje. La señalización por canal común tiene dos modos de explotación:

- Modo asociado: los mensajes de señalización que se refieren a una llamada siguen el mismo trayecto de voz entre dos puntos de señalización adyacentes, como se muestra en la figura 1.3.
- Modo cuasi-asociado o no asociado: los mensajes de señalización que se refieren a una llamada son conducidos por dos o más enlaces en tándem pasando a través de uno o más puntos de transferencia de señalización, los cuales no procesan el contenido del mensaje de señalización. Es decir que la señalización sigue un camino diferente al que sigue la señal de voz. Esto se muestra en la figura 1.4.

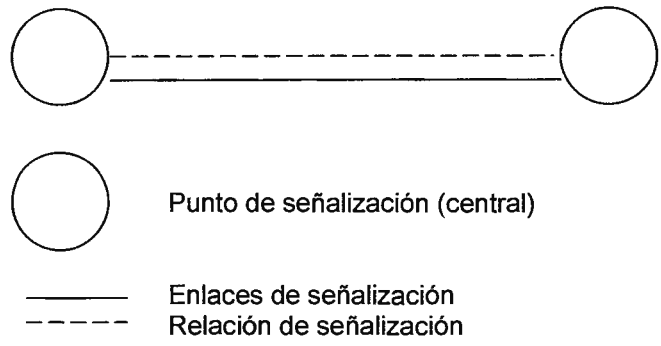


Figura 1.3 Esquema de señalización por canal común en modo asociado.

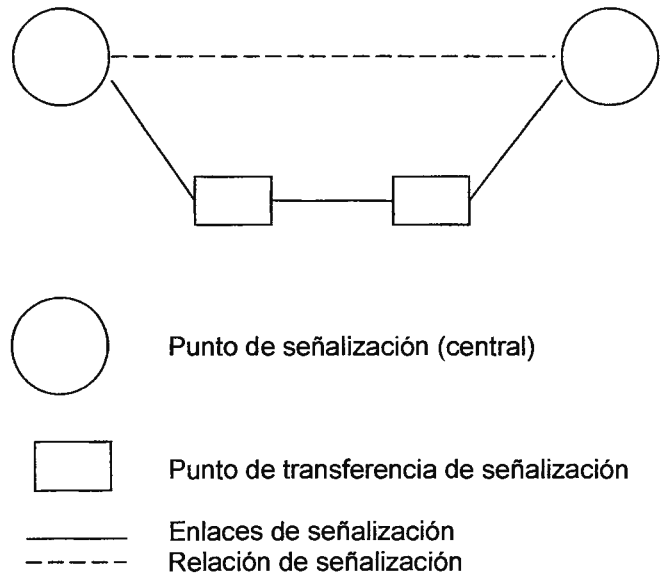


Figura 1.4 Esquema de señalización por canal común en modo cuasi-asociado.

1.4.2.1 Sistema de Señalización Número 7

El sistema de señalización número 7 o SS7 es un conjunto de protocolos que son utilizados por los proveedores de telefonía para soportar la señalización fuera de banda y las características avanzadas de llamadas. La SS7 es utilizada por la gran mayoría de operadores alrededor del mundo. El principal propósito de la SS7 es el establecimiento y la culminación de una llamada telefónica.

La SS7 es un estándar del Sector de Estandarización de la Unión Internacional de Telecomunicaciones (International Telecommunications Union Telecommunication Standardization Sector, abreviada ITU-T) desde 1981 en las recomendaciones de la serie Q.7xx. Una de las mayores diferencias con respecto a los estándares de señalización anteriores a SS7 es que SS7 es una señalización fuera de banda, mientras que las demás eran en banda.

Las redes SS7 son llamadas también Redes Inteligentes o Redes Avanzadas Inteligentes (Advanced Intelligent Networks, AIN) porque la información de señalización transportada por SS7 es en forma de mensajes, y esto permite no sólo ajustar la llamada, sino también otras funciones de servicios de valor agregado como por ejemplo el correo de voz.

1.4.2.1.1 Nodos de Señalización de la SS7

Físicamente una red SS7 tiene tres tipos de nodos de señalización:

- Puntos de servicios de conmutación (Service Switching Point, SSP). Normalmente los clientes del proveedor de servicio se conectan cerca de un SSP, el cual puede ser una computadora conectada a un switch de voz. Los SSP son puntos de origen y fin de la señalización. Si un cliente tiene su propia red, la señalización de su PBX es convertida por el SSP para que pueda ser transmitida por la red SS7. Un SSP también inicia consultas en la base de datos del proveedor de servicio.

- Puntos de transferencia de señal (Signal Transfer Point, STP). Dirigen el tráfico entre los nodos de señalización, hacen traducciones de las variantes de los protocolos de SS7 para que se puedan comunicar entre sí, y miden el tráfico y el uso de la red para facturar y supervisar. Los STP pueden ser nacionales, internacionales, o gateways. Estos últimos sirven para que las distintas redes de los proveedores de servicio puedan comunicarse entre sí.
- Puntos de control del servicio (Service Control Point, SCP). Conectan a la red SS7 bases de datos y otras aplicaciones entre las cuales se puede mencionar la detección inteligente del número telefónico (por ejemplo cuando un abonado llama al 911).

1.4.2.1.2 Topologías de Red de la SS7

En todos los puntos de una red SS7 hay redundancia tanto de equipos como de enlaces, compartiendo el tráfico y operando al 40%[5] de la capacidad máxima de manera que si falla un enlace, el otro enlace puede hacerse cargo de todo el tráfico sin congestionarse.

Físicamente el enrutamiento del tráfico de señalización no necesariamente sigue una ruta paralela a la que lleva la señal de voz. Como se mencionó anteriormente, la SS7 es una señalización por canal común.

Los enlaces de señalización se clasifican de la siguiente manera:

- Enlaces A o de acceso. Conectan el SSP o SCP con el STP más cercano. Es decir los enlaces A sirven para que las bases de datos y los switches telefónicos se conecten con los routers SS7.
- Enlaces B o de puente. Mediante estos enlaces se conectan pares de STP al mismo nivel jerárquico.
- Enlaces C o cruzados. Sirven para conectar un par de STP. Dan conectividad de respaldo por si hay congestión o fallan los enlaces B o D. Generalmente no se usan para tráfico relacionado a llamadas telefónicas.

---

[5] Scott Keagy, "Integración de redes de voz y datos".

- Enlaces D o diagonales. Conectan parejas de STP a diferentes niveles jerárquicos.
- Enlaces E o extendidos. Sirven para conectar switches telefónicos con STP remotos. Si los routers SS7 locales están congestionados estos enlaces son usados como enlaces de respaldo.
- Enlaces F o totalmente asociados. Sirven para dar conectividad para señalización directa entre SSP. En caso de que haya gran tráfico de señalización entre oficinas finales, este tráfico puede pasar por los mismos dispositivos que el tráfico de voz.

En la figura 1.5 se muestra la topología de una red SS7.

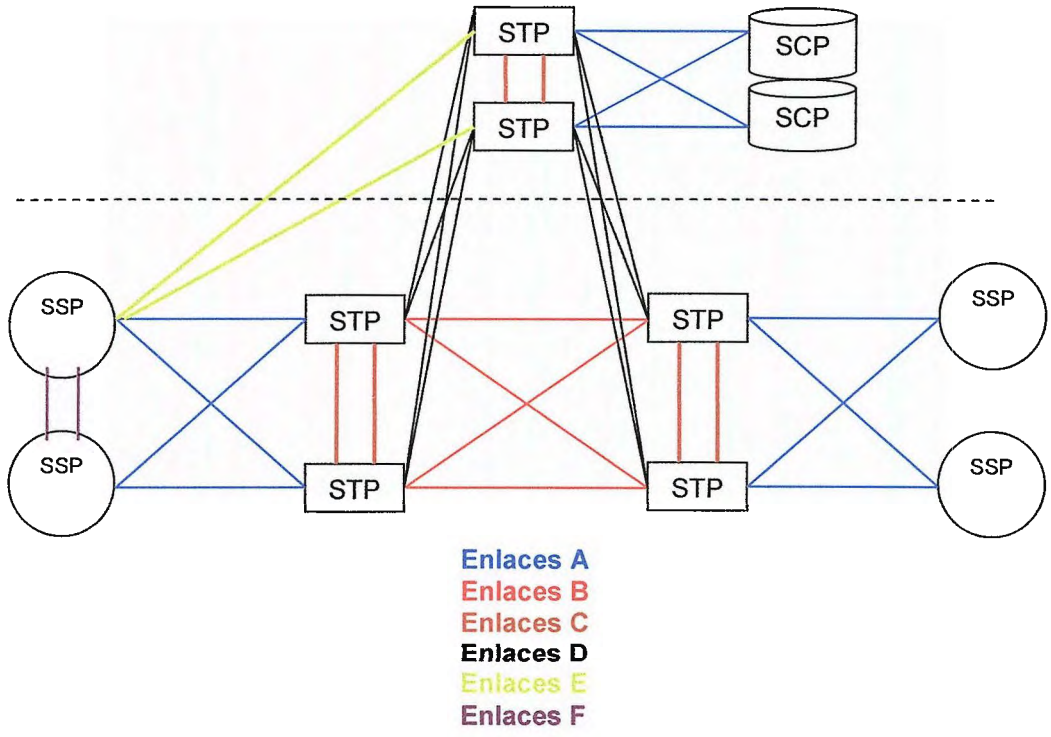


Figura 1.5 Topología de una red SS7

1.4.2.1.3 Direccionamiento de la SS7

En las redes SS7 las direcciones son necesarias para identificar a las entidades de red y los servicios disponibles dentro de cada una de las entidades. Los tipos de direccionamiento de la SS7 son:

- Números telefónicos.
- Códigos de punto.
- Números de subsistema.
- Códigos de señalización de enlace.
- Códigos de identificación de circuitos.

1.4.2.1.3.1 Números Telefónicos

Los números telefónicos identifican a los abonados, sin embargo no identifican a las entidades dentro de la red SS7 del proveedor de servicio. La tabla de enrutamiento de llamadas en un SSP se organiza de acuerdo a los dígitos del número telefónico marcado, luego un STP los traduce a un código de punto de destino. Esto tiene las ventajas de que se simplifica el enrutamiento de las llamadas para los SSP y concentra la inteligencia de red en los STP, y también se aumenta la seguridad de la red SS7 porque el STP se puede usar como firewall para ocultar el código del punto actual y las direcciones del número de subsistema a otras redes y regular el acceso a ciertos códigos y direcciones de subsistema.

Los dígitos del título global son los dígitos marcados del número de teléfono (destino). La traducción del título global es el proceso de resolver el código del punto de destino y el número de subsistema a partir del número marcado.

1.4.2.1.3.2 Códigos de Punto

Estos códigos proporcionan las únicas identidades para los SSP, STP y SCP, y su función equivale a la dirección IP en las redes TCP/IP. El formato de los códigos de punto es variable. La red global SS7 usa un espacio de dirección diferente en

los niveles nacional e internacional. Los gateways STP traducen entre la estructura del código de punto internacional y una estructura de código de punto nacional. Cada país puede definir cómo organizar la versión nacional de la estructura de código de punto.

1.4.2.1.3.3 Números de Subsistema

El número de subsistema identifica a un servicio dentro del nodo. Los números de subsistema son de 8 bits, pero pueden extenderse para albergar a más servicios después. La mayoría de números de subsistema pueden ser utilizados en aplicaciones de bases de datos.

1.4.2.1.3.4 Códigos de Señalización de Enlace

Estos códigos son direcciones lógicas que identifican las interfaces físicas, en otras palabras todos los enlaces que conectan con un destino determinado forman un conjunto de enlaces, y los enlaces dentro del conjunto se identifican mediante códigos de señalización de enlaces, por eso éstos sólo son importantes dentro de un punto de señalización dado. Las redes SS7 asignan un solo código de punto al nodo de manera global. Un nodo puede tener varios códigos de punto, pero no están relacionados a una interfaz física.

1.4.2.1.3.5 Códigos de Identificación de Circuitos

Un código de identificación de circuitos brinda la información necesaria para que un mensaje de señalización identifique a un troncal de voz determinado o el canal portador asociado con una llamada. El formato de los códigos identificadores de circuitos no está estandarizado.

1.4.2.1.4 Protocolos SS7

Los protocolos SS7 son:

- Componente de transferencia de mensajes (Message Transfer Part, MTP).
- Componente del usuario de teléfono (Telephone User Part, TUP).
- Componente del usuario de datos (Data User Part, DUP)
- Componente del usuario de la red digital de servicios integrados (Integrated Services Digital Network User Part, ISUP).
- Componente de control de la señalización de conexión (Signaling Connection and Control Part, SCCP).
- Componente de servicio de aplicación (Application Service Part, ASP).
- Componente de capacidades de transacción de la aplicación (Transaction Capabilities Application Part, TCAP).

La figura 1.6 muestra la arquitectura de protocolos SS7.

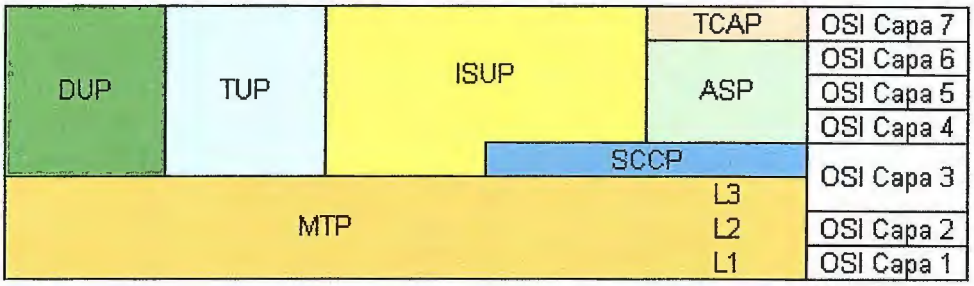


Figura 1.6 Arquitectura de los protocolos SS7.

En los protocolos SS7 los mensajes contienen los datos como una parte integral de la estructura del mensaje. En un mensaje SS7 no hay una clara diferencia entre un título y una carga útil debido a que las funciones se entremezclan. Se dice que una red SS7 es inteligente porque contiene la inteligencia del usuario final dentro de la definición del protocolo, ocultando de esta manera la complejidad de la red al abonado. Los datos de SS7 están estructurados en tipos de mensaje y parámetros predefinidos, así que toda la información está en la cabecera.

1.4.2.1.4.1.1 Componente de transferencia de mensajes (Message Transfer Part Layer 1, MTP L1) de capa 1

El MTP en general se encarga del transporte de mensajes SS7 de manera confiable, secuencial y no duplicada en la red SS7. El MTP de capa 1 se refiere a la interfaz física como su nombre lo indica, incluyendo características de la transmisión óptica o eléctrica. La recomendación ITU-T Q.702 habla al respecto. En El Salvador y en Europa se utilizan DS-0 de 64 kbps, el cual es la base de la señalización digital que corresponde a la capacidad de un canal de voz.

1.4.2.1.4.1.2 Componente de transferencia de mensajes (Message Transfer Part Layer 2, MTP L2) de capa 2

Como su nombre lo indica, se encarga del enlace de datos, e incluye la detección y corrección de errores del enlace y la secuenciación de los mensajes entre nodos. Los mensajes SS7 son encapsulados en una cabecera que brinda todas estas funciones. En caso de que no haya mensajes que enviar entonces es enviado un mensaje en blanco con el encapsulamiento MTP L2, y se le llama unidad de señal de relleno (Fill-In Signal Unit, FISU), que se muestra en la figura 1-7. Esto es abordado en la recomendación ITU-T Q.703.

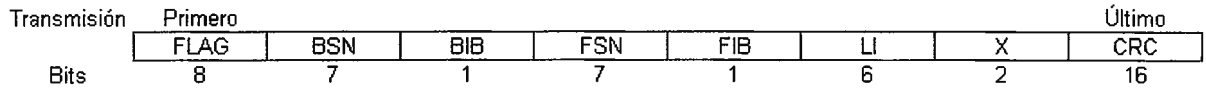


Figura 1.7 Unidad de señal de relleno (FISU).

A continuación se detallan los campos del FISU:

- El flag es un patrón de unos y ceros que identifica el inicio del mensaje.
- El número de secuencia negativa (backward sequence number, BSN) sirve para identificar el número de la última secuencia recibida válida.
- El bit indicador regresivo (backward indicator bit, BIB) indica un error de MTP L2.

- El número de secuencia de envío (forward sequence number, FSN) identifica el número de la última secuencia válida transmitida.
- El bit indicador de envío (forward indicator bit, FIB) sirve como reconocimiento de una transmisión, y se coloca al revés que el BIB. Se usan juntos para detectar errores, tienen el mismo valor en funcionamiento normal.
- El indicador de longitud (length indicator, LI) identifica el FISU (LI = 0), enlaza la unidad de señal de estado (link status signal unit, LSSU) (LI = 1 ó 2) o unidad de señal de mensaje (message signal unit, MSU) (LI = 3 ó más).
- La suma de comprobación de trama (frame check sum, FCS) utiliza el código de detección de errores CRC-16 para detectar errores de transmisión.

Aunque no se detecten errores en la transmisión un enlace puede ser inservible, tal como podría suceder en el caso en que en un nodo siga funcionando el MTP L2 a pesar de que en las capas más altas se presente una falla. Si esto sucede el nodo envía una unidad de señal de estado del enlace (LSSU) a través del enlace. La figura 1-8 muestra la unidad de señal de estado del enlace.

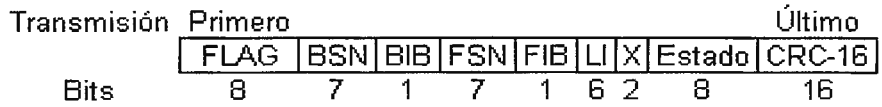


Figura 1.8 Unidad de señal de estado del enlace (LSSU).

En el campo de estado de la LSSU los últimos cinco bits se fijan a 0. Los tres primeros bits más significativos son llamados notificación del enlace. Los tipos de notificación del enlace son:

- Indicador de estado ocupado (status indicator busy, SIB), que se da cuando el nodo transmisor del mensaje está congestionado.
- Indicador de estado de procesador ocupado (status indicator processor outage, SIPO), que se da cuando no hay comunicación con MTP L3.

- Indicador de estado fuera de alineación (status indicator out of alignment, SIO), que es cuando se recibe el mensaje de violación de longitud o densidad de 1.
- Indicador de estado fuera de servicio (status indicator out of service, SIOS).
- Indicador de estado normal (status indicator normal, SIN), cuando inicia la autocomprobación no es posible enviar MTP L3.
- Indicador de estado de emergencia (status indicator emergency, SIE), cuando inicia la autocomprobación rápida no es posible enviar MTP L3.

1.4.2.1.4.1.3 Componente de transferencia de mensajes (Message Transfer Part Layer 2, MTP L3) de capa 3

La recomendación ITU-T Q.704 dice que en casos de fallas completas de enlaces el MTP L3 brinda notificación y re-enrutamiento. MTP L3 también identifica la capa más alta de la parte de usuario al que se le debe pasar el mensaje cuando llegue al punto de código del destino.

Los mensajes MTP L3 se pasan como unidades de señal de mensaje (MSU). La figura 1.9 muestra la MSU.

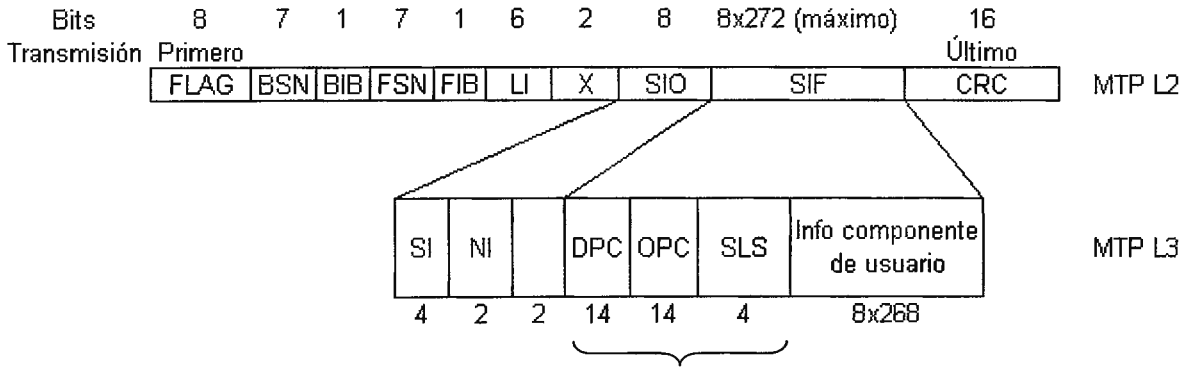


Figura 1.9 Unidad de señal de mensaje.

Los elementos más importantes de la MSU son el octeto de información de servicio (service information octect, SIO) y la etiqueta de enrutamiento dentro del campo de información de servicio (service information field, SIF). Dicha etiqueta contiene códigos de punto de la fuente y del destino, además de la selección de enlace de señalización (signaling link selection, SLS). La SLS identifica un enlace lógico dentro de un grupo de enlaces, lo cual conecta al siguiente punto de señalización de salto.

El SIO brinda la siguiente información:
Indicador de servicio (service indicator, SI), que es la parte del abonado al que se destina el MSU.
El MSU usa puntos de código y protocolos SS7 nacionales o internacionales. La versión del protocolo SS7 nacional usa el proceso de decisión para cada MSU, enrutando el mensaje si éste no es código de punto destino y distribuyéndolo en caso de que sí lo sea.

El enrutamiento de mensajes SS7 se hace de la siguiente forma. Cuando la función de asignación de ruta de MTP L3 recibe una MSU, el código de punto de destino es comparado con la tabla local de enrutamiento de código de punto. Después el SLS de la etiqueta de asignación de ruta identifica el enlace que debe usarse del conjunto de enlaces.

Desde el punto de vista de un SSP las distintas rutas para los tipos del enlace están administrativamente definidas así:

1. Enlace F: el destino está conectado directamente.
2. Enlace E: para llegar al destino local STP.
3. Enlace A: para llegar al STP local.

En cambio, desde el punto de vista de un STP:

1. Enlace A o E si el destino está conectado directamente.

2. Enlace B para llegar al STP local de destino.
3. Enlace D para llegar al STP local de grado superior asociado con el destino.
4. Enlace D para llegar al STP local de grado superior.

Cuando sucede congestión o falla en algún enlace, MTP L3 no puede enviar los mensajes a la parte de usuario de la capa superior, siendo entonces removidos los enlaces perjudicados de manera dinámica de las tablas de enrutamiento de los puntos de señalización afectados. Luego de la corrección las rutas vuelven a restablecerse dinámicamente. Todo esto se logra porque los puntos de señalización vecinos intercambian mensajes de administración. Son tres las funciones de administración:

- Enlace. Los enlaces son activados, desactivados y restablecidos basándose en la entrada de los mensajes MTP L2, así como también emite las funciones de administración de tráfico.
- Tráfico. Los puntos de señalización notifican la disponibilidad de enlaces para posibilitar los ajustes de enrutamiento. Los mensaje de cambio adelante y cambio atrás son administrados para que los puntos de señalización modifiquen sus tablas de enrutamiento para descartar o incorporar enlaces.
- Ruta. Si todos los enlaces de una señalización adyacente están indispensables entonces la administración de ruta se encarga de enviar el tráfico hacia otro punto de señalización, basándose en cambios de la topología en condiciones de congestión. En caso de que un punto se señalización pierda todos sus enlaces hacia un destino entonces envía un mensaje de transferencia prohibida a sus vecinos para que éstos encuentren un camino alternativo hacia el destino deseado.

1.4.2.1.4.2 Componente del Usuario de Teléfono (Telephone User Part, TUP)

TUP sirve para establecer y cancelar básicamente las llamadas telefónicas, con limitadas características adicionales. Actualmente se encuentra en desuso y es

utilizado en pocos países. TUP se define en las recomendaciones ITU-T Q.721 a Q.725.

1.4.2.1.4.3 Componente del Usuario de Datos (Data User Part, DUP)

La recomendación ITU-T Q.741 hace referencia a DUP. Este protocolo define los elementos necesarios para el control de llamadas, y el registro y cancelación de de facilidades. Los mensajes de señalización se dividen en dos categorías:

- Mensajes relacionados a llamadas y circuitos. Son usados para establecer y terminar una llamada o para supervisar el estado del circuito.
- Mensajes relacionados al registro y cancelación de facilidades. Se usan para intercambiar información entre puntos de señalización para registrar y cancelar información relacionada con facilidades del abonado.

1.4.2.1.4.4 Componente del usuario de la red digital de servicios integrados (Integrated Services Digital Network User Part, ISUP)

Este protocolo define el proceso para establecer, administrar y liberar circuitos troncales que llevan voz y datos sobre la red telefónica pública conmutada. ISUP ha remplazado a TUP y DUP en muchos países, y por ello ISUP se utiliza para llamadas de la red digital de servicios integrados como por llamadas que no son de dicha red. Las llamadas que se originan y finalizan en el mismo switch no usan ISUP. Las recomendaciones ITU-T Q.761 a Q.764 y Q.766 definen a ISUP.

1.4.2.1.4.5 Componente de control de la señalización de conexión (Signaling Connection and Control Part, SCCP)

SCCP se define en las recomendaciones ITU-T Q.711 a Q.716. Este protocolo ofrece mejoras a MTP L3 para brindar servicios a redes orientadas y no orientadas a conexión, así como traducción. En la figura 1.10 se muestra el SCCP.

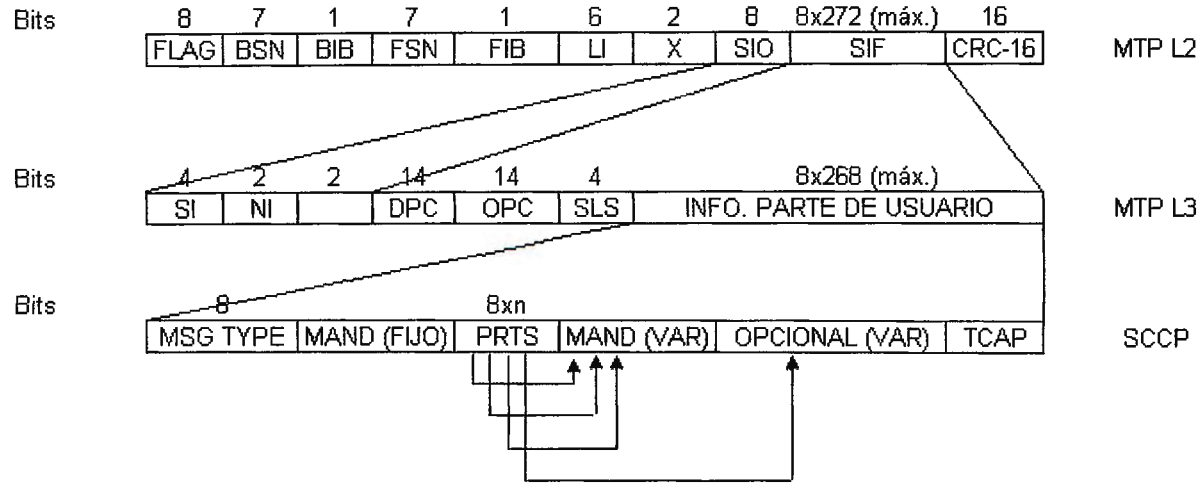


Figura 1.10 Componente de control de la señalización de conexión.

El tipo de mensaje (MSG TYPE) es un octeto obligatorio para todos los mensajes. El código de tipo de mensaje unívocamente define la función y el formato para cada mensaje SCCP.

SCCP proporciona funciones que están contenidas en MTP L3 tales como enrutamiento final a final y traducción global del título. MTP se encarga de la congestión en los enlaces y nodos, y SCCP se encarga del control del flujo en una capa más alta.

1.4.2.1.4.6 Componente de servicio de aplicación (Application Service Part, ASP)

Este protocolo prepara los servicios orientados a conexión de SCCP y da la sesión y servicios de presentación a TCAP y otras partes de la aplicación de usuario.

1.4.2.1.4.7 Componente de capacidades de transacción de la aplicación (Transaction Capabilities Application Part, TCAP)

Este protocolo se encarga de transportar los mensajes del abonado sobre los enlaces de señalización por medio de unidades de señalización. TCAP proporciona la comunicación entre las interfaces. La utilidad de los mensajes TCAP es acceder a bases de datos para pedir servicios en nodos remotos. Los mensajes TCAP pasan las órdenes al switch telefónico remoto para habilitar estos servicios. La recomendación Q.771 define TCAP.

1.4.2.1.5 Interfaz de abonado a redes SS7

La interfaz de acceso básico de la red digital de servicios integrados o la interfaz de acceso principal proporcionan a los abonados el acceso a redes SS7. Esto es ventajoso porque así las redes SS7 están protegidas contra ataques ya que no se revela a los abonados la topología de las redes ni el direccionamiento de los circuitos del proveedor, evitando así que la red esté indisponible.

CAPÍTULO II

2. LA SEÑALIZACIÓN UTILIZADA EN TELEFONÍA

En este capítulo se revisan las conexiones troncales de telefonía existentes entre los CPE (equipo terminal del abonado) y las oficinas centrales (Central Office, CO) administradas por un proveedor de circuitos. En un entorno empresarial típico, el CPE es un PBX (*Private Branch Exchange,* Intercambio privado de ramas) que conecta a los usuarios con la Red pública de telefonía conmutada (PSTN) a través de la CO. El CPE también puede ser un sencillo teléfono, como el existente en el hogar, o un equipo un poco más complejo, como el existente en una pequeña oficina.

En algunos casos, la CO no está implicada en la señalización telefónica, excepto para *proporcionar* la infraestructura de transmisión (por ejemplo, T-1/E-1) entre los emplazamientos de los clientes. En estos casos, los *switches* telefónicos situados en cada uno de los emplazamientos del cliente se relacionan con los demás a través de los *switches* situados en la CO. La señalización troncal utilizada en estas líneas punto a punto o troncales también cae dentro del ámbito de este capítulo.
Este capítulo se divide en las siguientes secciones:

- Funciones de señalización
- Señalización troncal analógica
- Tipos de enlace troncal digital

En la primera sección se revisan las diversas funciones y tipos de señalización que *deben* ser proporcionados por las líneas troncales de telefonía, y en la siguiente sección se revisa el modo en que se suministran dichas funcionalidades a través de diversos medios de transmisión troncal y de diversos protocolos.

2.1 Funciones de señalización

Aparte de la transmisión de una ruta de audio en ambas direcciones, existe un conjunto de importantes funciones de señalización que deben ser proporcionadas por una línea o un troncal de telefonía. En este capítulo, un troncal se refiere a una conexión entre *switches* telefónicos (por ejemplo, las PBX, los sistemas de clave o las CO) y una línea se refiere a una conexión entre un teléfono y un *switch* telefónico.

En esta sección se revisarán los siguientes tipos de señalización:

- Detección de equipo colgado-descolgado *(on-hook/off-hook).*
- Supervisión de comienzo de marcación.
- Transmisión de dígitos.
- Identificación de número.
- Tonos de progresión de la llamada.
- Supervisión de respuesta y desconexión.

Además de las funciones de señalización fundamentales descritas aquí, son necesarios otros tipos de señalización para posibilitar las opciones de llamada avanzadas.

2.1.1 Detección de equipo colgado-descolgado

Como su nombre implica la señalización *on-hook/off-hook* (colgado-descolgado) requiere sólo dos estados: *on-hook* y *off-hook.* El teléfono está colgado cuando la parte del microteléfono (micrófono y auricular) está descansando sobre la base del teléfono. El teléfono está descolgado cuando el microteléfono se alza de la base del teléfono para realizar o recibir una llamada. Los puntos finales troncales del teléfono deben permitir la transmisión y recepción de señales que indican los estados *on-hook y off-hook.* El método de señalización telefónica de estos dos estados difiere entre los tipos de enlaces troncales.

La señalización *on-hook/off-hook* es obligatoria para todos los tipos de troncales excepto para las conexiones troncales permanentes (por ejemplo, conexiones *always-on* o *hoot-n-holler).* Los enlaces troncales normalmente se implementan configurando la interfaz a un estado *off-hook.*

2.1.2 Supervisión de comienzo de marcación

La supervisión de comienzo de marcación asegura que el *switch* telefónico receptor está preparado para interpretar los dígitos (es decir, el número del teléfono de destino) transmitidos por el *switch* telefónico emisor. Sin supervisión de comienzo de marcación, el *switch* telefónico receptor puede perder los primeros dígitos de una dirección destino; entonces se producirá un intento de llamada fallido. Por esta razón, debería utilizar un tipo de señalización de enlace troncal que incluya supervisión de comienzo de marcación siempre que sea posible. Para enlaces troncales analógicos, E&M *wink-start* es la mejor opción y además ampliamente disponible. Algunos enlaces troncales digitales que utilizan señalización de canal común, proporcionan la funcionalidad de supervisión de comienzo de marcación.

Aunque la supervisión de comienzo de marcación proporciona confirmación positiva de que el *switch* remoto está preparado para recibir dígitos, no proporciona confirmación de que los dígitos se hayan recibido correctamente.

2.1.3 Transmisión de dígitos

Los teléfonos y los *switches* telefónicos transmiten dígitos para representar las direcciones de destino y proporcionar entradas desde los usuarios a los sistemas automatizados como buzón de voz, autoasistidos, distribuidores automáticos de llamada (*Automatic Call Distribution,* ACD) y sistemas de respuesta interactiva de voz (*Interactive Voice Response,* IVR).

Los sistemas de telefonía normalmente utilizan dos clases de transmisión de dígitos:

- Marcación por pulsos.
- Marcación por tonos.

2.1.3.1 Marcación por pulsos

En los teléfonos y circuitos originales se necesitaba de una operadora para interconectar la llamada de la parte origen a la parte destino, luego se implementó la marcación por pulsos que consistía en una serie de pulsos generados por los teléfonos de marcación giratoria y así interconectar automáticamente la llamada.

Esta marcación posee dos estados en el tiempo denominados off-hook y on-hook (denominados también como make y brake respectivamente), el on-hook corresponde a un pulso y cada dígito está representado por una cantidad de pulsos, por ejemplo el cero está representado como diez pulsos, el uno por un pulso y así sucesivamente.

La Figura 2.1 ilustra el estado de la línea con el transcurso del tiempo cuando se marca 1234.

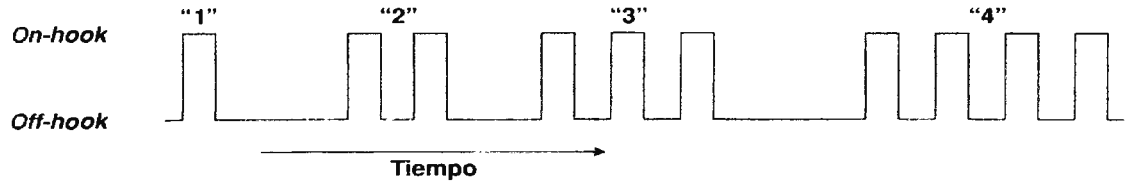


Figura 2.1 La marcación por pulsos transmite una serie de pulsos on-hook para representar los dígitos marcados.

Según la especificación EIA/TIA-470, los sistemas de marcación por pulsos norteamericanos operan a una velocidad nominal de 10 pulsos por segundo (PPS). Sin embargo, las tolerancias de la ingeniería permiten la operación desde 8 a 11 PPS. En otros países se aplican diferentes normas y velocidades de pulso, como el 20 PPS nominal utilizado en Japón.

2.1.3.2 Marcación por tonos

Hay varios estándares para transmitir dígitos mediante tonos audibles:

- Marcación multifrecuencia (DTMF).
- Marcación multifrecuencia por pulsos (*Multifrecuency pulse,* MF-P).
- Multifrecuencia obligada (MF-C).

Las normas de transmisión de dígitos multifrecuencia (MF) son generalmente un subconjunto de protocolos de señalización más amplios basados en los tonos MF audibles.

Estos protocolos de señalización más amplios describen la transmisión de dígitos, señales de supervisión, señales de línea y otros tipos de control de llamada y de información.

2.1.3.3 Marcación multifrecuencia

La marcación multifrecuencia (DTMF) es el *método* más utilizado para la *transmisión* de dígitos, tanto para la transmisión del número de destino como para el intercambio de información dentro de banda entre los que llaman y los sistemas automatizados de telefonía (buzón de voz, IVR, ACD, etc.).

La planificación entre dígitos y frecuencias audibles se basa en el diseño físico de un teclado numérico de teléfono estándar. Se asigna una frecuencia distinta a cada fila y columna del teclado numérico, de manera que cada tecla se corresponda con dos frecuencias, como se muestra en la Tabla 2.1.

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Tabla 2.1 Frecuencias DTMF asignadas a un teclado de teléfono estándar[6]

[6] S. Keagy , "Integración de redes de voz y datos".

La tecla * se utiliza normalmente para señalar solicitudes de funciones procedentes del usuario a la red pública. Muchos fabricantes de PBX también siguen este convenio para las redes privadas. La tecla # se utiliza normalmente como un dígito de terminación en redes públicas, de modo que la conexión de llamada puede empezar sin esperar a que termine la interrupción de dígitos.

2.1.3.4 Marcación multifrecuencia por pulsos

La asignación de frecuencias para señalización DTMF se diseñó para simplificar la construcción de los primeros teléfonos DTMF. Aunque el orden es intuitivo, no es el uso más eficaz de frecuencias de señalización. Los esquemas de asignación multifrecuencia (MF) utilizan generalmente un grupo de seis frecuencias, del que se seleccionan dos frecuencias distintas para representar cada valor. Aunque la planificación entre dígitos y frecuencia es menos intuitiva, los esquemas MF corrientes utilizan menos componentes de frecuencia que el esquema DTMF.

	700 Hz	900 Hz	1100 Hz	1300 Hz	1500 Hz	1700 Hz
700 Hz		1	2	4	7	ST3
900 Hz			3	5	8	ST1
1100 Hz				6	9	KP(*)
1300 Hz					0	ST2
1500 Hz						ST(#)
1700 Hz						

Tabla 2.2 Asignación MF para transmisión de dígitos CCITT SS#5 y R1 MF[7]

2.1.3.5 Multifrecuencia obligada

El protocolo más común de multifrecuencia obligada (MF-C) para la transmisión de dígitos es el protocolo de señalización CCITT R2, especificado en el CCII Blue Book de 1988, Fascículo VI, Parte 4 (ahora conocido como Recomendaciones

[7] S. Keagy, "Integración de redes de voz y datos".

Q.400 a Q.490 de la ITU-T). En realidad, los protocolos de señalización R2 se dividen en las partes de señalización de línea, que transmiten señales de supervisión, y las partes de señalización interregistro. La Tabla 2.4 ilustra la asignación de frecuencia para la transmisión de dígitos mediante señales interregistro R2 MF en la dirección de envío.

	1380 Hz	**1500 Hz**	**1620 Hz**	**1740 Hz**	**1860 Hz**	**1980 Hz**
1380 Hz		1	2	4	7	Código 11
1500 Hz			3	5	8	Código 12
1620 Hz				6	9	Código 13
1740 Hz					0	Código 14
1860 Hz						Código 15
1980 Hz						

Tabla 2.3 Transmisión de dígitos MF obligados utilizados en la señalización CCITT R2. [8]

La señalización se llama multifrecuencia obligada porque los tonos MF se ejecutan de continuo hasta que el lado remoto los reconoce con otro tono MF. Este hecho es importante porque significa que ambos extremos del enlace troncal deben transmitir tonos MF durante la misma conexión de llamada. También puede ver información sobre señalización semiobligada. Un enlace troncal semiobligado utiliza señalización obligada en un sentido y señalización de pulsos en el otro.

Debido a que ambos extremos de un enlace troncal MF-C necesitan transmitir tonos MF a través de la misma ruta de audio durante la misma conversación, hay un potencial de confusión. Un punto extremo recibe un reflejo de sus propias señales MF que es similar a las señales esperadas desde el extremo remoto. Para combatir el problema potencial, los sistemas MF-C (y, más generalmente, los sistemas bidireccionales MF) suelen utilizar diferentes rangos de frecuencia para los dos sentidos de la llamada. Esto significa que se deben usar muchas

[8] S. Keagy, "Integración de redes de voz y datos".

frecuencias dos veces mientras se está dentro de las frecuencias en la banda de voz.

Una nota final acerca de la señalización obligada para la transmisión de dígitos: para redes con largos retrasos de transmisión (por ejemplo, redes satélite o redes VoX), el reconocimiento individual de cada dígito puede llevarnos a un tiempo muy alto de conexión de llamada. Esta situación es análoga a la transferencia de datos TCP/IP (*Transmission Control Protocol/Internet Protocol*, Protocolo de Control de Transmision/Protocolo de internet) con un tamaño de ventana de 1 paquete. Considere, por ejemplo, una red con un retraso de 200+ ms en cada dirección. Un número de 11 dígitos necesita un mínimo de 4,4 segundos para la transmisión (400 ms para cada dígito con reconocimiento), suponiendo que los dígitos se detecten instantáneamente. Teniendo en cuenta los 50 a 60 ms que se necesitan para detectar cada dígito, el retraso global de la transmisión de dígitos MF obligados es bastante mayor que la transmisión de dígitos MF por pulsos.

2.1.4 Identificación de números

Hay una variedad de métodos para proporcionar el número de teléfono de la parte que efectúa una llamada (la parte que llama), y la parte que recibe una llamada (la parte llamada). Caller ID (*Caller Identification* o CID, o más propiamente dicho Identificación del Numero de Llamada –CNID) es un servicio telefónico que transmite los números que llaman a los teléfonos de la parte llamada durante la señal de ringing o cuando la llamada está siendo configurada pero antes de ser contestada.

2.1.4.1 Identificación de la parte que llama

La identificación de la parte que llama puede representar el origen técnico de la llamada (es decir, el número de teléfono exacto del enlace troncal/línea saliente desde el CPE), o puede representar el origen administrativo de la llamada (es decir, la parte que paga las facturas de teléfono). El ID del llamante (CID), a veces

llamado entrega de número llamado (*Called Number Delivery,* CND), proporciona el origen técnico de una llamada, y la Identificación automática de número (*Automatic Number Identification*, ANI) proporciona el origen administrativo de una llamada. ANI se menciona en la Ley de Telecomunicaciones de la SIGET en el art. 19 literal c), y en el Reglamento de la Ley de Telecomunicaciones en el art. 37 y art. 49.

Para los abonados particulares, los valores CID y ANI son generalmente iguales. Para abonados comerciales con múltiples enlaces troncales/líneas en una localización, CID y ANI probablemente serán distintos. La ANI es la misma para cada enlace troncal, suponiendo que el teléfono comercial esté consolidado bajo un solo número de cuenta (*bill-to number,* BTN). El CID representa el número de teléfono real del enlace troncal origen, que es diferente para cada enlace troncal.

a) ID del llamante

El ID del llamante consta de dos partes de información: solo el número y nombre+número. Cuando un switch telefónico origen envía un número de teléfono como el ID del llamante, la compañía telefónica recibe la llamada y es responsabilidad almacenar el nombre del abonado en una base de datos.

La información del ID del llamante es enviada al destino entre la primera y segunda señal de tono como se indica en la figura 2.2. Este formato es de dos segundos on y cuatro off y en países donde el período de silencio es mas corto debe ser modificada a este formato para ajustarse al tiempo obligado.

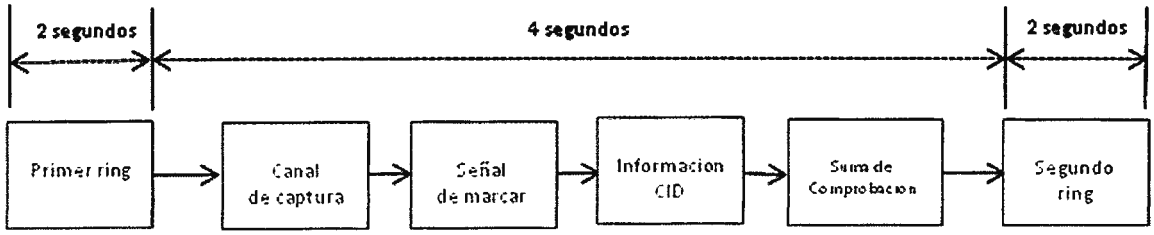


Figura 2.2 Visión en conjunto de señalización del ID del llamante

La información CID es una corriente síncrona binaria en serie que opera a 1.200 bits por segundo (bps). Un 0 lógico se representa por un tono de 2.200 Hz a -13,5 decibelios por miliwatio (dBm), y un 1 lógico se representa mediante un tono de 1.200 Hz a -13,5dBm. El nombre para este tipo de señalización es modulación continua de tecla de cambio de frecuencia binaria (*Frecuency Shift Keying*, FSK).
Al menos 500 ms después del primer *ring,* la CO envía una señal de captura de canal que es una serie alternante de ceros y unos para 250 ms (exactamente 300 bits, comenzando con un O y terminando con un 1). La captura de canal va seguida de una señal de marcar que dura 150 ms. Estos dos elementos de información son esencialmente un preámbulo de la verdadera información CID.
La información CID real puede tener dos formatos:

- Formato sencillo de mensaje de datos (*Simple Data Message Format*, SDMF).
- Formato múltiple de mensajes de datos (*Multiple Data Message Format*, MDMF).

Los formatos SDMF y MDMF incluyen los datos y el tiempo de la llamada. El formato MDMF incluye el número de teléfono y una cadena ASCII para el nombre de la parte que llama, mientras que el formato SDMF sólo identifica el número de teléfono de la parte que llama. A la información CID sigue una suma de comprobación.

b) Identificación automática del número

La identificación automática del número (ANI) se puede proporcionar en troncales analógicos y digitales desde el *switch* del proveedor al CPE. La ANI pasa a través de las redes de los proveedores SS7 mediante el mensaje de dirección inicial (*Initial Address Message,* IAM) del protocolo ISUP (Parte de Usuario de la Red Digital de Servicios Integrados, *ISDN User Part*), o a través de tonos multifrecuencia (MF) dentro de banda cuando la red no dé soporte a SS7. La red de señalización del proveedor, así como la conexión del enlace troncal desde la

CO al CPE, determinan cómo pasa la ANI desde el proveedor al CPE. Los dos métodos básicos de transmisión de la ANI desde la CO al CPE son:

- Mensaje RDSI Q.931 SETUP.
- Tonos CAS MF con Grupo de función D (FGD).

Si el proveedor tiene una red SS7 y el abonado tiene una conexión RDSI al proveedor, entonces la ANI pasa un mensaje Q.931 SETUP. La información del mensaje SETUP se asigna directamente desde el mensaje IAM en la red SS7. Cuando el abonado no tiene una conexión RDSI (Red Digital de Servicios Integrados) al proveedor, o ésta no tiene una red de señalización SS7, la ANI sólo se puede transmitir a través de enlaces troncales que se activan para la señalización del grupo de elemento D (FGD). Con FGD, la información actual de ANI se pasa dentro de banda usando tonos multifrecuencia (MF) analógicos, de acuerdo a la especificación R1 MF. FGD se proporciona generalmente en enlaces troncales E&M *wink-start* con un reconocimiento *wink* de la información ANI (explicada posteriormente en esta sección).

La señalización FGD se diseñó originalmente para los enlaces troncales entre un Proveedor de intercambio local (LEC) y un Proveedor de intercambio (IXC), para proporcionar a los abonados conveniente acceso a proveedores de servicio de larga distancia. Sin FGD, un abonado que tuviera que marcar un número de teléfono a un proveedor de larga distancia (por ejemplo, 1-800-CALL-ATT), introduciría un número pin para especificar la cuenta de pago y después marcaría el número destino. Con FGD, un abonado marca un código de prefijo de 1OXXXXX junto al número de teléfono destino. No hay tonos de marcación intermedios y no es necesario proporcionar información sobre la cuenta de pago, porque la señalización en el enlace troncal FGD proporciona automáticamente esta información. FGD también permite que los abonados seleccionen un IXC predeterminado a través del que efectuar llamadas de larga distancia cuando no se marca código de prefijo.

2.1.4.2 Identificación de la parte llamada

La identificación de la parte llamada proporciona el número que el llamante marcó al destino de la llamada. Este servicio es en realidad muy útil para abonados que tengan varios números de teléfono asociados con el mismo enlace troncal(s)/línea(s). En una configuración particular, la aplicación puede ser de teléfonos separados para padres e hijos, con un *ring* distinto que identifique la parte a la que se llama. En una configuración comercial, las aplicaciones incluyen marcación directa (Direct Dial, DID) y centros de llamada.

Hay dos clases de identificación de parte llamada:

- Marcación directa (*Dialing Direct*, DID).
- Servicio de identificación de número marcado (*Dialed Number Identification Service*, DNIS).

La clase DID se utiliza generalmente para dirigir las llamadas a un abonado específico en una empresa que comparte líneas telefónicas. La marcación directa se puede proporcionar en enlaces troncales analógicos o digitales, utilizando cualquier mecanismo de transmisión de dígitos:

- Marcación por pulsos (DP).
- Marcación multifrecuencia (DTMF).
- R1 multifrecuencia (MF).
- R2 multifrecuencia obligada (MF-C).

La clase DNIS se utiliza generalmente para dirigir llamadas a través de un sistema distribuido automático de llamadas (ACD) a un grupo de agentes apropiado, o para proporcionar información mejorada *screen-pop* en entornos de centros de llamada. DNIS se suministra en un mensaje RDSI Q.931 SETUP (basado en el contenido de un mensaje ISUP IAM en la red SS7). DNIS también se puede proporcionar mediante tonos MF.

2.1.5 Tonos de progreso de llamada

Las redes de voz señalan una variedad de condiciones para los usuarios a través de tonos audibles de progreso de llamada. Los tonos de progreso de llamada son familiares a los usuarios dentro de un país; sin embargo, varían entre países en sus características de frecuencia y cadencia. Un tono de progreso de llamada consiste ya sea de un solo tono, una combinación de hasta tres tonos o una secuencia de tonos. Si un solo tono es utilizado entonces debe ser modulado por un tono de frecuencia menor. Las frecuencias y cadencias utilizadas en El Salvador se presentan en la sección 3.3.1 "Señales presentes en la línea".

Al integrar redes en una escala global, es importante que el equipo en cada país proporcione los tonos de progreso de llamada acostumbrados en el país. Esto asegura que los usuarios dentro de cada país tengan una experiencia familiar y consistente al usar el sistema telefónico.

2.1.6 Supervisión de respuesta y de desconexión

La supervisión de respuesta en el extremo lejano y la supervisión de desconexión son elementos que proporcionan confirmación positiva del inicio y del fin de sesiones de llamada. Estos elementos son importantes para los proveedores de servicios con propósitos de facturación y contabilidad. Las compañías telefónicas no son los únicos proveedores de servicios; otros ejemplos de proveedores de servicios que se benefician de la supervisión de respuesta y de desconexión incluyen compañías que aplican programas departamentales *charge-back* para los gastos de telecomunicación, y hoteles que cobran a los huéspedes las llamadas telefónicas.

Sin la supervisión de respuesta, los proveedores de servicios no tienen un punto fijo para empezar a facturar una conexión. En tales casos, el proveedor de servicios puede comenzar a facturar veinte o treinta segundos después de que se marque el número, con la suposición de que el destino habrá contestado en este

tiempo. En otros casos, el proveedor de servicios puede empezar a facturar inmediatamente después de que se haya marcado el número. El problema de estos planteamientos es que las llamadas incompletas también se facturan, muy a pesar de los abonados.

2.2 Enlaces troncales de voz analógicos

Los enlaces troncales de voz analógicos se utilizan cuando el *switch* telefónico no soporta conexiones digitales, o cuando se necesitan pocos canales de voz. Por ejemplo, las oficinas pequeñas con sistemas de teclas utilizan tradicionalmente enlaces troncales analógicos para líneas *tic* y conexiones PSTN. Las oficinas grandes con PBX utilizan líneas *tic* analógicas para conectar con las oficinas remotas pequeñas.

Hay tres tipos comunes de enlace troncal analógico:

- Inicio de bucle *(loop-start).*
- Inicio de tierra *(ground start).*
- E&M.

2.2.1 Inicio de bucle

Los sistemas de telefonía residencial de todo el mundo utilizan señalización *loop-start.* Debido a que los enlaces troncales conectan entre los *switches* telefónicos y las líneas conectan un *switch* telefónico a un teléfono, las facilidades particulares se llaman líneas *loop-start.* Un circuito entre una CO y un PBX se puede llamar enlace troncal desde la perspectiva del cliente, o línea desde la perspectiva del vendedor del circuito.

2.2.1.1 Circuito libre

La señalización analógica *loop-start* utiliza sólo un par de cables entre el *switch* telefónico en una CO y el teléfono o *switch* telefónico conocido como CPE. La Figura 2.3 ilustra un circuito libre *loop-start* sin llamadas activas.

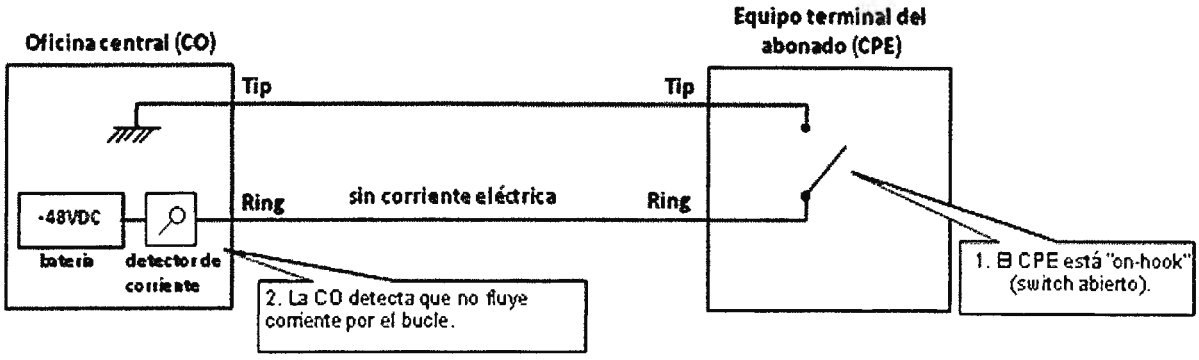


Figura 2.3 circuito loop-start en estado libre.

La CO proporciona una batería de corriente continua (DC) de -48 volt (V), que genera corriente eléctrica a través del bucle del circuito. Los teléfonos particulares no necesitan fuentes de potencia separadas por esta razón. Las CO también proporcionan un generador de tono de marcación y un generador de *ringing* de corriente alterna (AC) para señalar al CPE. La corriente eléctrica fluye desde la batería en la CO al CPE mediante el cable *ring,* y vuelve a través del CPE a la CO a tierra mediante el cable *tip*. La CO sabe cuándo la corriente fluye a través del bucle mediante un detector de corriente en el cable *tip*. Observe que el CPE no necesita una toma a tierra eléctrica.

2.2.1.2 Conexiones salientes desde el CPE

Cuando el CPE está en el estado *on-hook* (es decir, el teléfono está colgado), hay un *switch* eléctrico abierto que evita que la electricidad fluya a través del bucle del circuito. Cuando el CPE inicia una llamada cambiando al estado *off-hook* (es decir, el teléfono está descolgado), el *switch* eléctrico se cierra y la corriente fluye a través del bucle del circuito. Cuando la corriente fluye por el bucle, la CO detecta la condición *off-hook* del CPE. La CO responde transmitiendo un tono de

marcación en el bucle, el cual informa al CPE de que la CO está preparada para recibir dígitos del número de teléfono de destino. Esto es una forma de supervisión de comienzo de marcación. La detección *off-hook* y la supervisión de comienzo de marcación para enlaces troncales *loop-start* se ilustra en la Figura 2.4.

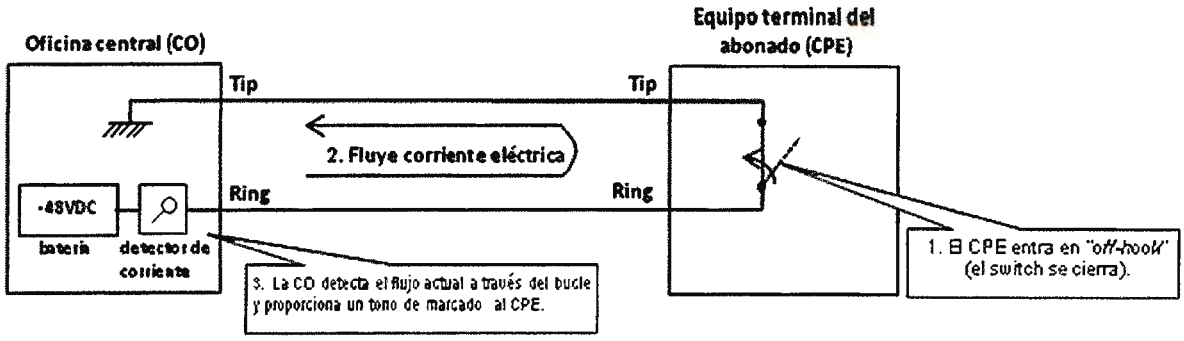


Figura 2.4 El CPE inicia una conexión en un enlace troncal loop-start

Siguiendo la transmisión de dígitos (mediante pulsos o DTMF) y los tonos de progreso, la señal de supervisión de respuesta se transmite cuando la parte distante responde a la llamada. La CO local transmite la señal de supervisión de respuesta como una inversión de polaridad en los cables tip y *ring*. En otras palabras, las conexiones tip y *ring* a tierra y a la batería se invierten en la CO durante la llamada, como se muestra en la Figura 2.5.

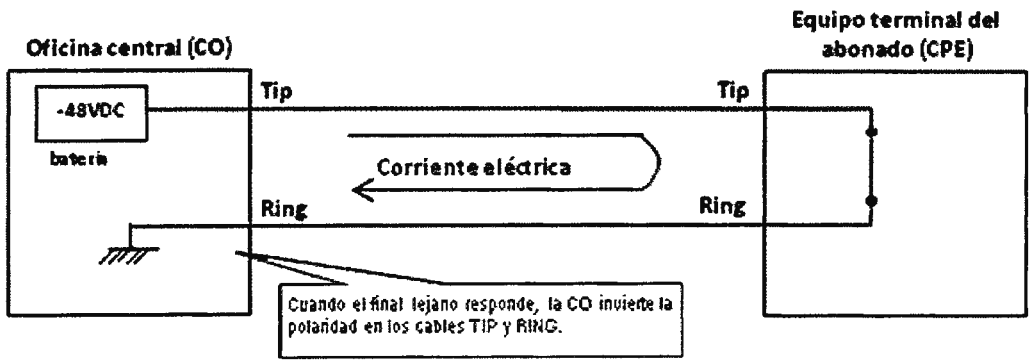


Figura 2.5 Supervisión de respuesta para enlaces troncales loop-start proporcionada como inversión de polaridad tip y ring

El CPE está diseñado para funcionar con cualquier polaridad de señal. Sin

embargo, los equipos convencionales no pueden transmitir señales DTMF 0cuando la polaridad está invertida. Debido a que la supervisión de respuesta loop-start invierte la polaridad de la señal durante la llamada, el equipo convencional no puede usar sistemas de buzón de voz, autoasistidos e IVR cuando la supervisión de respuesta esté presente.

2.2.1.3 Conexiones entrantes desde la CO

La CO señala una llamada entrante destinada al CPE enviando un voltaje de señal de marcación AC (entre 90 V y 140 V) en el bucle junto con la batería -48 VDC. Si el CPE un teléfono estándar, el voltaje AC se convierte en un *ring* audible que alerta al abonado. Si el CPE es un *switch* telefónico, la conexión a la CO se establece automáticamente después de un número predeterminado de *rings.* Cuando el CPE cambia a un estado *off-hook,* el *switch* eléctrico se cierra y la corriente fluye a través del bucle del circuito. La CO detecta que la corriente está fluyendo por el bucle y se establece la conexión de audio. La Fig. 2.6 ilustra este proceso.

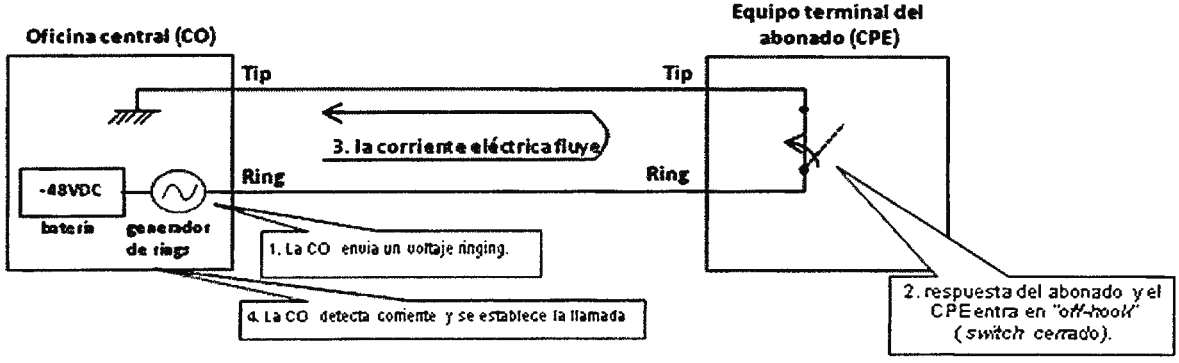


Figura 2.6 La CO inicia una conexión en un enlace troncal loop-start

2.2.1.4 Supervisión de desconexión

Hay varios métodos para que una CO proporcione supervisión de desconexión al CPE en enlaces troncales *loop-start:*

- Inversión de batería.
- Negación de batería.
- Desconexión de tono supervisor (*Supervisor Tone Desconnection*, STD).

La inversión de batería para la supervisión de desconexión es similar a la inversión de batería para la supervisión de respuesta *loop-start.* Los cables *tip* y *ring* se invierten en la CO, lo que ofrece confirmación positiva al CPE de que la parte final lejana ha vuelto a un estado *on-hook.*

La negación de batería se produce cuando la CO local elimina la batería -48VDC desde el bucle durante al menos 350 ms cuando la parte remota se desconecta. La negación de batería debe ser mayor de 350 ms para evitar la confusión con la transmisión de dígitos marcados por pulsos.

La desconexión de tono supervisor (STD) es un tono audible de progreso de llamada que indica que la parte remota se ha desconectado. El STD más comúnmente utilizado es una señal de ocupado; la señal real varía de país a país. Sin embargo, generalmente es un tono de 600 Hz o menos con una cadencia periódica *on-off.*

2.2.2 Ground-Start

El siguiente proceso traza una toma de enlace troncal *ground-start* por el CPE:

1. El CPE señala la CO que quiere tomar el enlace troncal.
2. La CO reconoce la petición del CPE para tomar el enlace troncal.
3. El CPE completa el bucle de circuito, o toma el enlace troncal.

Cuando la CO debe entregar una llamada entrante, se necesita al menos uno de estos pasos:

1. La CO señala al CPE en el que hay una llamada entrante.
2. El CPE reconoce la petición para tomar el enlace troncal completando el bucle de circuito.

En ambos casos, la CO detecta la corriente tan pronto como el CPE completa el bucle de circuito, y la llamada continúa como en el caso de un enlace troncal *loop-start*. Las siguientes secciones ofrecen una descripción más detallada del proceso de señalización *ground start*.

2.2.2.1 Circuito desocupado

Los enlaces troncales *ground-start* analógicos utilizan dos cables entre la CO y el CPE. Los enlaces troncales *ground-start* no funcionan correctamente a menos que los cables tip y *ring* se conecten con la polaridad correcta (es decir, tip a tierra, *ring* a -48VDC). La Figura 2.7 ilustra un circuito *ground-start* en estado desocupado.

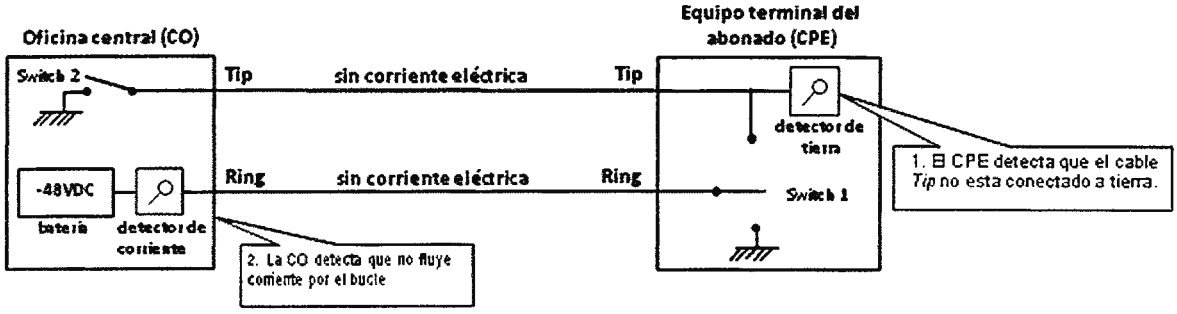


Figura 2.7 Enlace troncal ground-start en un estado desocupado

2.2.2.2 Conexiones salientes desde el CPE

Cuando el PBX o el sistema de teclas deben tomar el enlace troncal para una llamada saliente, mueve el *switch* 1 mostrado en la figura 2.8 a la posición de tierra, lo que proporciona una ruta de corriente eléctrica desde la batería en la CO a tierra en el CPE. La CO detecta el flujo de corriente y reconoce la petición del CPE para la toma del enlace troncal cerrando el *switch* 2. Este facilita una ruta a la corriente eléctrica desde el detector de tierra CPE a la tierra CO. La figura 2.8 ilustra este proceso.

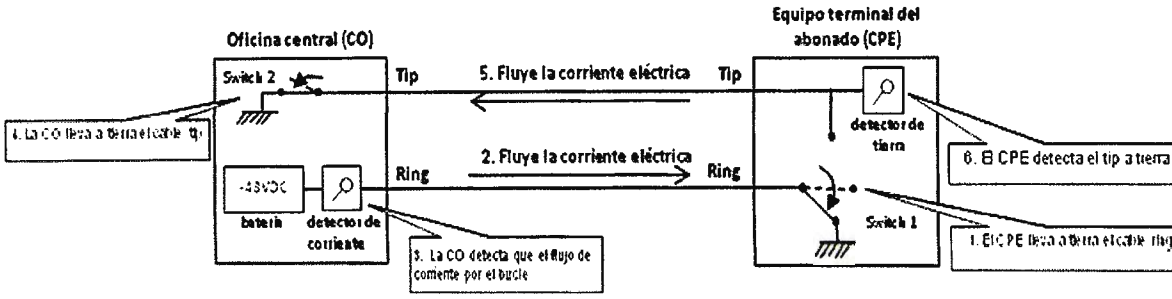


Figura 2.8 El CPE lleva a tierra el cable ring para pedir un enlace troncal; la CO contesta llevando a tierra el cable tip.

La Figura 2.9 ilustra los pasos restantes de la captura del enlace troncal. Cuando El CPE detecta que la CO ha llevado a tierra el cable *tip*, el CPE completa el bucle del circuito (como en un *loop-start),* y elimina la señal local a tierra. En este estado, el enlace troncal captura y aparece exactamente como un enlace troncal *loop-start.*

Hay que recordar que el motivo de esta señalización de acá para allá es minimizar la ventana de tiempo durante el cual ambos extremos del enlace troncal pueden capturar simultáneamente el circuito (una condición no conveniente llamada *glare).*

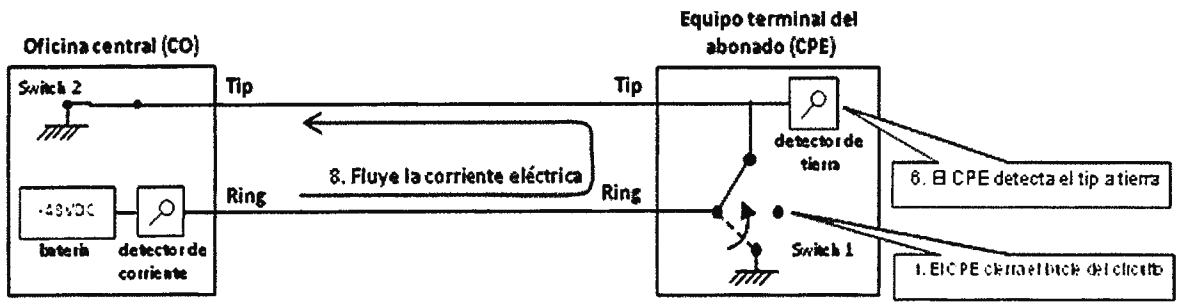


Figura 2.9 El CPE completa el bucle del circuito, que ahora aparece igual que un enlace troncal loop-start.

2.2.2.3 Conexiones entrantes desde la CO

La Figura 2.10 ilustra los primeros pasos que ocurren cuando una CO entrega una llamada entrante al CPE. La CO cierra el *switch* 2, que lleva a tierra el cable *tip*, y envía un voltaje de señal de marcación AC por el cable *ring*. El CPE percibe el cable *tip* llevado a tierra y el voltaje de señal de marcación AC en el cable *ring*, y en respuesta completa el bucle del circuito. En este punto, la llamada continúa como si lo hiciera en un enlace troncal *loopstart*.

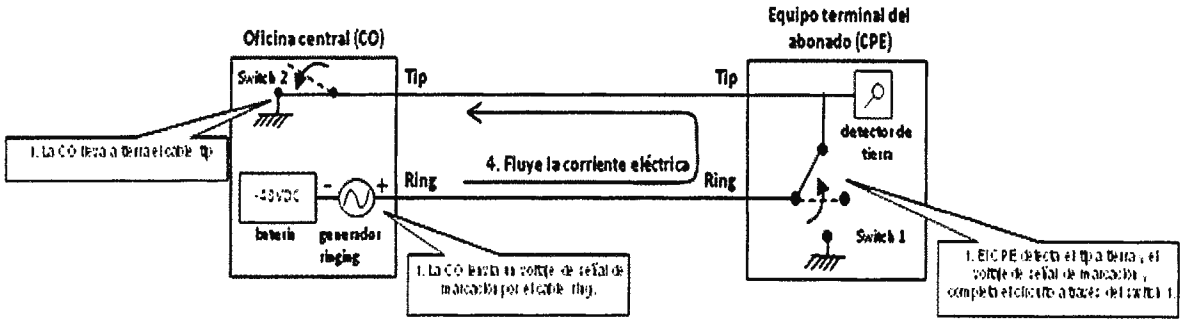


Figura 2.10 La CO señaliza la llamada al CPE llevando a tierra el tip y enviando una señal ringing AC por el cable ring.

2.2.3 E&M

E&M son las siglas de *Ear and Mouth, Earth and Magneto*. Los diagramas de circuito E&M tienen cables que se nombran con E y M (así como otras cuantas letras), de modo que el nombre es relevante en cualquier caso.

Los enlaces troncales *loop-start y ground-start* utilizan el mismo par de cables tanto para la ruta de audio como para las funciones de señalización. Los enlaces troncales E&M aíslan estas funciones en pares de cables distintos.

Dependiendo de la configuración del circuito E&M, las señales de enlace troncal y la ruta de audio pueden necesitar uno o dos pares de cables, para un total de cuatro a ocho cables. Dado que la operación del enlace troncal depende de la

disposición correcta de todos los cables, los enlaces troncales E&M cambian más al implementarse en la capa física que otros tipos de enlace troncal. La sección que sigue explorará varias facetas de los enlaces troncales E&M analógicos, incluyendo:

- FXO y FXS
- Cableado de interfaz.
- Ruta de audio.
- Tipos de circuito.
- Supervisión de comienzo de marcación.

2.2.3.1 FXO y FXS

Foreign Exchange Suscriber (FXS) y Foreign Exchange Office (FXO), son los nombres de las dos mas comunes interfaces (puertos o conexiones) encontradas en un ambiente telefónico análogo.

La interface FXS (la conexión sobre la pared del abonado) es un servicio de la CO y debe ser conectada al equipo del abonado (teléfonos, módems, y FAX). En otras palabras una interface FXS esta en la parte del abonado. Una interface FXS provee los siguientes servicios para un equipo del abonado:

- Tono de marcar
- Corriente de batería
- Voltaje de ring

La interface FXO (en el teléfono) recibe servicios POTS, típicamente de una CO de la PSTN. En otras palabras una interface FXO esta orientada a la CO. Una interface FXO provee los siguientes servicios de la red telefónica: Indicación on-hook/off-hook.

Las características descritas anteriormente, es que una línea de telecomunicaciones de un puerto FXO debe ser conectada para un puerto FXS en

orden a la conexión de trabajo. Cuando se realiza la conexión entre FXO y FXS se recibe el servicio telefónico por parte de la compañía escuchando el tono de marcar cuando se levanta el auricular.

2.2.3.2 Procedimiento de llamada FXS-FXO

- Usualmente el local loop está en modo Loop start.
- La FXO se conecta a una PBX o a una CO.
- La FXS se conecta a un aparato telefónico o a una línea y genera el timbre.
- La FXO detecta el timbre, cierra el lazo cuando se levanta el auricular y lo abre cuando el teléfono está colgado.
- La FXO se comporta como la red telefónica y se conecta a una línea de dos hilos mediante la marcación de dígitos DMTF, que identifican el destino a ser llamado.
- Una FXO recibe una llamada detectando un voltaje de ring proporcionado por un dispositivo FXS (VoIP, PBX, etc.)
- Ya en el estado off-hook se responde la llamada.

2.2.3.3 Cableado de interfaz

La Tabla 2.4 identifica el cableado que se utiliza en las interfaces de enlace troncal E&M analógico.

Cable	Nombre	Descripción
E	Oído (Ear) o Tierra (Earth)	El oído del PBX oye las señales desde la CO en este cable. El estado de la señal es: 1) flujo de corriente; o 2) no hay flujo de corriente.
M	Boca (Mouth) o Magneto	La boca del PBX habla a la CO por este cable. El estado de la señal es: 1) flujo de corriente; o 2) no hay flujo de corriente.

Cable	Nombre	Descripción
SG	Señal a tierra (Signal Ground)	Forma un bucle con el cable E a través del que la corriente puede fluir en configuraciones de aislamiento a tierra.
SB	Señal de batería (Signal Battery)	Forma un bucle con el cable M a través del que corriente puede fluir en configuraciones de aislamiento a tierra.
T1/R1	Tip-1/Ring-1	Proporciona audio entrante al PBX en circuitos E&M de cuatro cables, o audio en dos sentidos de circuitos E&M de dos cables.
T/R	Tip/Ring	Proporciona audio saliente procedente del PBX sobre circuitos E&M de cuatro cables; no se utiliza para circuitos de dos cables.

Tabla 2.4 Cables eléctricos utilizados para los enlaces troncales E&M analógicos. [9]

Debe conectar el cableado a un bloque *punch-down* para interoperar con algunos equipos de vendedor, mientras que otros vendedores necesitan que los cables estén aplicados directamente en ranuras en el chasis. La Figura 2.11 ilustra el pin de cableado RJ-45.

[9] S. Keagy, "Integración de redes de voz y datos".

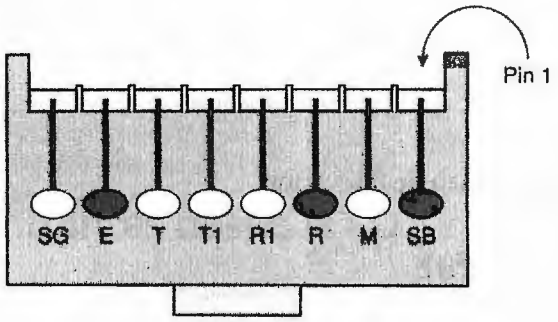


Figura 2.11 Disposición de los pines de un cableado E&M analógico.

2.2.3.4 Ruta de audio

Un enlace troncal E&M analógico puede operar en el modo E&M de dos cables o en el modo E&M de cuatro cables. Esta distinción sólo se refiere al número de cables empleados en la parte de audio del circuito. Los circuitos E&M de dos cables alojan las direcciones de transmisión y recepción de la ruta de audio sobre un solo par de cables. Los circuitos E&M de cuatro cables utilizan un par de cables para la dirección de transmisión y otro par de cables para la dirección de recepción de la ruta de audio.

NOTA:
Tanto la versión de dos cables como la de cuatro cables de un enlace troncal E&M analógico sólo soportan una única conexión de teléfono. No puede utilizar circuitos E&M de cuatro cables para efectuar dos llamadas simultáneas.

2.2.3.5 Tipos de circuito

Hay cinco configuraciones primarias de circuitos E&M, las cuales se distinguen por el grado de aislamiento eléctrico a tierra y la capacidad de operar en una configuración *back-to-back.*

La operación *back-to-back* de las interfaces de enlace troncal es conveniente para fines de prueba en laboratorio, y para cuando las conexiones E&M analógicas se

deban establecer entre PBX contiguos. Las interfaces de enlace troncal pueden operar en un diseño *back-to-back* cuando los sistemas de señalización sean simétricos. Esto significa que el bucle de señalización transmisor debe enviar las mismas señales que se esperan en el bucle de señalización receptor.

Para evitar los problemas asociados con los potenciales a tierra diferentes, las interfaces de aislamiento a tierra de enlace troncal E&M están diseñadas para que los sistemas eléctricos se conecten sólo a una toma de tierra en el mismo extremo del enlace troncal.

Los circuitos E&M sin aislamiento a tierra necesitan dos cables de señalización, y los circuitos con aislamiento a tierra necesitan dos pares de cables de señalización.

2.2.3.6 Supervisión de comienzo de marcación

Se ha tratado anteriormente la forma en que se representan los estados *on-hook* y *off-hook* los diferentes tipos de circuitos E&M. Esta sección explora los métodos de un PBX o una CO para tomar un enlace troncal E&M y comenzar la transmisión de dígitos, cada uno de los cuales implica una señalización *on-hook* y *off-hook*:

- *Immediate start*
- *Delay start*
- *Wink start*
- *Tone start*

a) Immediate start

E&M immediate start es la forma mas sencilla de supervisión de comienzo de marcación, y es casi lo mismo que la no supervisión. El lado origen del enlace troncal envía la señal *off-hook* y después de 150 ms comienza a transmitir los dígitos de la dirección de destino (en otras palabras, el número llamado). La idea es que la pausa de 150 ms es suficiente para que el receptor esté preparado. Sin

embargo, no hay ningún método para que el lado receptor indique que está realmente preparado para recibir dígitos. Después de la pausa el lado origen del enlace troncal transmite los dígitos tanto si el receptor está preparado como si no lo está.

b) Delay start

En señalización *delay-start,* el lado origen del enlace troncal E&M va *off-hook*, y comprueba el estado del lado receptor después de 75 a 300 ms. Si el receptor está *off-hook* durante la comprobación, el lado origen espera hasta que el receptor cambia a un estado *on-hook*. En sistemas sin integridad *delay-dial*, el lado origen transmite los dígitos después de un tiempo predefinido, incluso si el receptor no está preparado (similar a *immediate start*). Los sistemas con integridad *delay-dial* funcionan de forma similar a los sistemas *wink-start*. Es decir, el lado origen del enlace troncal espera a la transición del lado remoto de *off-hook* a *on-hook* antes de transmitir los dígitos.

c) Wink start

La señalización *wink-start* es el método común de supervisión de comienzo de marcación. Debería ser su primera elección al suministrar una interfaz E&M (analógica o digital). La figura 2.12 ilustra el proceso de captura del enlace troncal y la preparación para la transmisión de dígitos.

El lado origen del enlace troncal E&M envía la señal *off-hook* y espera el reconocimiento desde el receptor.

El receptor detecta la señal *off-hook* y contesta con un *wink* (un pulso *off-hook* que dura desde 140 a 290 ms, seguido del retorno a un estado *on-hook*).

El lado origen detecta el reconocimiento *wink* desde el receptor, hace una pausa de al menos 210 ms y después transmite los dígitos.

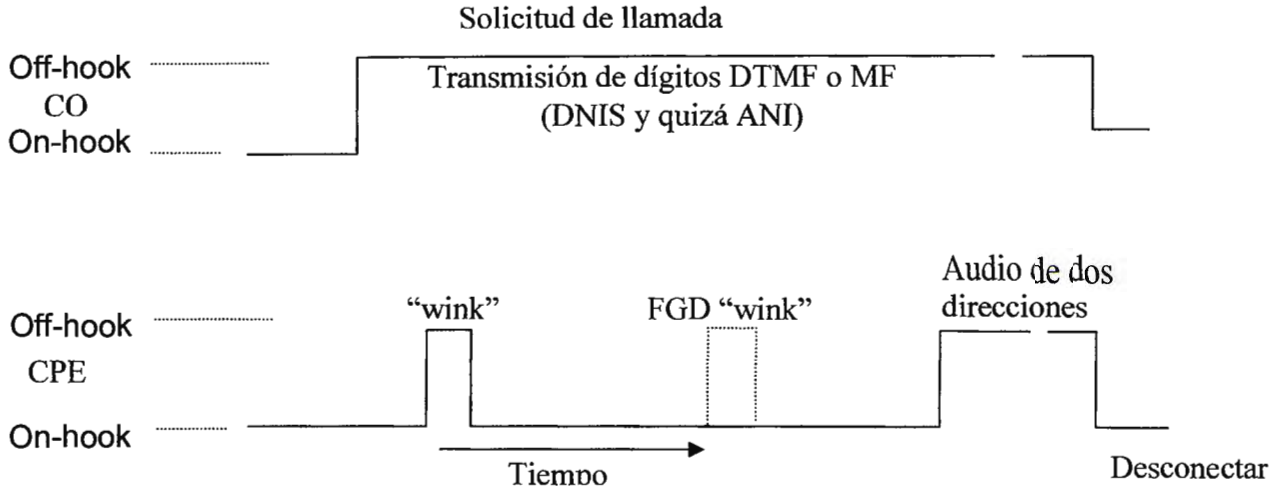


Figura 2.12 La señalización wink-start proporciona confirmación de que el receptor está preparado para recibir dígitos.

Si está activado Grupo de función D (FGD) para el enlace troncal, el lado receptor reconoce los dígitos con otro *wink.* Este tipo de señalización se llama E&M *wink start* con reconocimiento wink. El grupo de función B (FGB) no utiliza el reconocimiento *wink.*

El receptor cambia a un estado *off-hook* cuando la parte llamada contesta la conexión y se construye la ruta de audio.

d) Tone start

La señalización *tone-start* E&M se usa a veces entre PBX en redes privadas. Es parecida a la señalización *loop-start* y *ground-start*, en las que el lado receptor responde a una petición de captura de enlace troncal con un tono de marcación. El lado origen puede transmitir automáticamente los dígitos cuando detecte el tono de marcación. Si el buffer de dígitos en el lado origen del enlace troncal está vacío, la parte que llama oye el tono de marcación y marca dígitos adicionales para completar la llamada.

2.3 Tipos de enlace troncal digital

Cuando hay suficiente volumen de llamadas para justificar el costo de hardware y los cargos periódicos por el circuito, los enlaces troncales digitales son las opciones a elegir. Los circuitos T-1 y E-1 son las opciones digitales estándar que interaccionan con los PBX y los sistemas híbridos de teclas.

Para circuitos T-1, esto incluye codificación de líneas de inversión alternada de marcas (AMI) con formato de supertramas (SF) D4, así como codificación de líneas de sustitución bipolar 8-cero (B8ZS) con formato ampliado de supertramas (ESF). Para circuitos E-1, esto incluye codificación de líneas AMI o bipolares de alta densidad 3 (HDB3), y trama 0 timeslot con o sin multitramas CRC-4.

El resto de ésta sección explora los tipos de telefonía que se utilizan con circuitos T-1 y E-1, divididos en dos amplias clases:

- Señalización de canal asociado (CAS).
- Señalización de canal común (CCS).

2.3.1 Señalización de canal asociado

En la señalización de canal asociado (CAS), hay una relación determinante entre las señales de control de llamada y los canales de audio TDM que éstas controlan. La estructura básica de tramas de las funcionalidades T-1 y E-1 incluyen posiciones binarias específicas que representan la información de señalización para canales de audio específicos. Para las funcionalidades T-1, los bits de señalización se colocan en el mismo timeslot que la ruta de audio que controlan. Para las funcionalidades E-1, los bits de señalización se colocan en el timeslot 16 con una estructura de multitrama, de modo que los bits específicos en tramas específicas de la multitrama representan las señales de un canal TDM dado. Un sencillo algoritmo puede interpretar el estado de señalización de cada canal de audio TDM leyendo las posiciones de bits específicos en una multitrama T-1/E-1.

Las posiciones de bits específicos son constantes para cada supertrama T-1 o estructura de multitrama E-1.

Las siguientes secciones explorarán cómo los bits de señalización de control de llamada se insertan en tramas T-1/E-1, y cómo los valores de estos bits indican estados de señalización de control de llamadas:

- T-1: Señalización *robbed-bit.*
- E-1: Multitramas de 16 *timeslots.*
- Esquemas de señalización ABCD.

2.3.1.1 T-1: Señalización robbed-bit

La señalización robbed-bit (RBS) T-1, se llama así porque algunos de los bits de la carga de audio en cada *timeslot* TDM se quitan y se vuelven a utilizar para dar información de señalización. Cada seis tramas (0,75 ms), los bits que representan el estado de señalización para cada timeslot sobrescriben la información de audio codificada en el bit menos significativo de sus respectivos *timeslots.*

Ya que sólo el bit menos significativo de algunas tramas se sobrescribe, y ya que la codificación de modulación de código por pulsos (PCM) de la PSTN estándar es flexible a errores binarios, RBS no tiene efectos notables en la calidad de voz. Sin embargo, RBS tiene un impacto significativo en los timeslots que se usan para la transmisión de datos (en oposición a la voz). Si no se puede desactivar RBS para un canal, el ancho de banda efectivo del canal se reduce de 64 a 56 Kbps. Esto es por lo que algunas llamadas de datos RDSI se deben configurar para usar sólo 56 Kbps para cada canal B.

La Figura 2.13 ilustra cómo los bits de audio se quitan y se vuelven a usar para la señalización en unión con supertramas D4 en funcionalidades T-1.

Numero de trama	Bits en el *timeslot* 1								Bits en el *timeslot* 2								,,,	Bits en el *timeslot* 24							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
1																	,,,								
2																	,,,								
3																	,,,								
4																	,,,								
5																	,,,								
6								A1								A2	,,,							A24
7																	,,,								
8																	,,,								
9																	,,,								
10																	,,,								
11																	,,,								
12								B1								B2	,,,							B24

Figura 2.13 Señalización robbed-bit utilizadas con supertramas (SF) D4 en circuitos T-1.

Voz PCM

Bits ABCD

2.3.1.2 E-1: Multitramas de 16 timeslot

La señalización de canal asociado parece algo más intuitiva para los contadores de los circuitos E-1, ya que toda la información de señalización está contenida en el timeslot 16. Aunque todas las señales están en un solo timeslot (lo que parecería indicar un esquema de señalización de canal común), las posiciones de bits en ese timeslot tienen un significado fijo que es una parte inherente a la estructura de la trama. En otras palabras, todos los estados de señalización se definen en el mismo nivel de trama, y son continuamente codificados tanto si se reciben como si no se reciben llamadas activas o peticiones de señalización. Contraste esto con los métodos CCS, en los que un timeslot específico se utiliza como una corriente de bits en serie para mensajes de transmisión de eventos que necesitan un protocolo de capa más alto para ser interpretados.

La Figura 2.14 ilustra una sola trama E-1, en la que el timeslot cero (TS-0) proporciona las funciones de facilidad de trama, el timeslot 16 (TS-16) proporciona las funciones CAS y los restantes timeslots contienen señales de voz moduladas por código de pulsos (PCM).

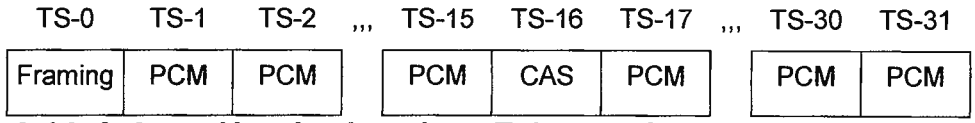


Figura 2.14 Asignación de timeslots E-1 para formato de tramas, datos y señalización.

Hay dos tipos de estructuras de multitramas para circuitos E-1. La primera estructura se basa en los bits CRC4 en TS-O. Esta estructura ofrece la misma función que una supertrama o una supertrama ampliada sobre facilidades T-1: para mantener la integridad de temporización del circuito síncrono y para proporcionar facilidades de prueba de capa de enlace (por ejemplo, señales de prueba de bucle, alarmas, etc.). La segunda estructura de multitrama se basa en TS-16 y es independiente de las multitramas en TS-0. Es decir, no es necesario que la primera trama de una multitrama TS-16 sea la primera trama de la multitrama TS0. En la práctica, muchos vendedores alinean estas estructuras de multitrama para simplificar. La Figura 2.21 ilustra cómo se colocan los bits CAS en el timeslot 16 de la multitrama TS-16.

La primera trama de la multitrama TS-16 proporciona sincronización para la multitrama y señalización de facilidad limitada. El bit Y indica la presencia o la ausencia de una alarma amarilla (que es una alarma roja del lado remoto), y los bits X no se utilizan y se reservan para el futuro o el uso de un país específico. La segunda trama de la multitrama TS-16 comienza la codificación CAS de bits. Los cuatro primeros bits codifican los bits ABCD para TS-1, y los cuatro últimos codifican los bits ABCD para TS-17. En la siguiente trama, los bits ABCD para TS-2 y TS-18 son codificados. TS-16 de cada trama sucesiva codifica los bits ABCD

de los restantes timeslots. En la Figura 2.21, los subscripts de los bits ABCD indican qué timeslots gobiernan.

Numero de trama	Bits en el *timeslot*-16							
	1	2	3	4	5	6	7	8
1	0	0	0	0	X	Y	X	X
2	A_1	B_1	C_1	D_1	A_{17}	B_{17}	C_{17}	D_{17}
3	A_2	B_2	C_2	D_2	A_{18}	B_{18}	C_{18}	D_{18}
4	A_3	B_3	C_3	D_3	A_{19}	B_{19}	C_{19}	D_{19}
5	A_4	B_4	C_4	D_4	A_{20}	B_{20}	C_{20}	D_{20}
6	A_5	B_5	C_5	D_5	A_{21}	B_{21}	C_{21}	D_{21}
7	A_6	B_6	C_6	D_6	A_{22}	B_{22}	C_{22}	D_{22}
8	A_7	B_7	C_7	D_7	A_{23}	B_{23}	C_{23}	D_{23}
9	A_8	B_8	C_8	D_8	A_{24}	B_{24}	C_{24}	D_{24}
10	A_9	B_9	C_9	D_9	A_{25}	B_{25}	C_{25}	D_{25}
11	A_{10}	B_{10}	C_{10}	D_{10}	A_{26}	B_{26}	C_{26}	D_{26}
12	A_{11}	B_{11}	C_{11}	D_{11}	A_{27}	B_{27}	C_{27}	D_{27}
13	A_{12}	B_{12}	C_{12}	D_{12}	A_{28}	B_{28}	C_{28}	D_{28}
14	A_{13}	B_{13}	C_{13}	D_{13}	A_{29}	B_{29}	C_{29}	D_{29}
15	A_{14}	B_{14}	C_{14}	D_{14}	A_{30}	B_{30}	C_{30}	D_{30}
16	A_{15}	B_{15}	C_{15}	D_{15}	A_{31}	B_{31}	C_{31}	D_{31}

Figura 2.15 Asignación ABCD de bits en una multitrama de 16 timeslots

2.3.1.3 Esquemas de señalización ABCD

Muchas variaciones de CAS utilizan sólo los bits A y B para imitar los tipos comunes de señalización analógica:

- *Loop start.*
- *Ground start.*
- E&M *immediate start*
- E&M *delay dial.*
- E&M *wink start.*

a) Loop start digital

La señalización loop-start digital necesita los bits de señalización A y B. El CPE controla el bit A, que representa el estado del bucle de circuito (A = O abierto, A =

1 cerrado). La CO controla el bit B, que representa el estado del tono de señal de marcación (B = O señal de marcación, B = 1 no señal de marcación). La CO y el CPE repiten los valores de los bits que reciben en los bits que no controlan, de modo que los valores del bit AB son iguales en cada dirección.

La CO inicia una captura del enlace troncal loop-start enviando un tono de señal de marcación sobre el cable *tip*. Cuando el abonado levanta el auricular (o el switch telefónico contesta automáticamente la llamada entrante), el bucle se cierra y la ruta de audio se activa.

b) Ground start digital

La señalización ground-start digital también necesita los bits de señalización A y B. En la señalización loop-start digital, la CO y el CPE controlan uno de los dos bits de señalización, los cuales tienen el mismo significado en ambos sentidos de la transmisión. Ground start digital se diferencia porque los bits tienen significados distintos y se configuran independientemente en cada sentido. La CO y el CPE tienen control completo de ambos bits de señalización que ellos transmiten.

c) E&M immediate-start digital

La señalización E&M digital sólo necesita 1 bit de señalización en cada sentido para representar dos estados de señalización: A = 0 es on-hook y A = 1 es off-hook. El bit B, que no se necesita, está generalmente configurado para seguir al bit A, incluso para sistemas que sigan de otra forma las recomendaciones de la ITU-T (por ejemplo, configuración C = O y D = 1). La complejidad física de los circuitos E&M analógicos desaparece en la señalización E&M digital, ya que las variaciones de cableado de circuito analógico y los niveles de voltaje se resumen en el valor de un único bit de señalización. Algunas variaciones de la señalización E&M digital permiten varios métodos de supervisión de comienzo de marcación.

d) E&M delay start digital

El CPE envía una señal off-hook a la CO para indicar que está preparado para efectuar una llamada. La CO responde con una señal off-hook para reconocer la petición. Al CPE no se le permite transmitir dígitos hasta que la CO tenga un colector de dígitos preparado para recibirlos. La CO indica su disposición a recibir dígitos volviendo a un estado on-hook. Cuando la CO recibe los dígitos y la parte remota contesta, la CO señaliza el estado off-hook, y la llamada se activa.

e) E&M winkstart digital

El CPE envía una señal off-hook a la CO para indicar que está preparado para efectuar una llamada. La CO reconoce la petición de captura de enlace central con un wink. Una vez que el CPE transmite los dígitos, la CO puede proporcionar otro reconocimiento wink de que los dígitos se han recibido correctamente. El segundo wink se asocia con la señalización elemento de grupo D (FGD), que se utiliza normalmente en enlaces troncales entre un LEC y un IXC. Cuando la parte remota contesta, la CO señala un estado off-hook y se activa la llamada.

2.3.2 Señalización de canal común

En la señalización de canal común (CCS), un timeslot dedicado de un circuito T-1/E-1 transporta una corriente arbitraria de bits en serie entre los puntos extremos del enlace troncal. Este timeslot se llama con frecuencia canal de datos (o canal-D en el caso de RDSI). Los puntos extremos CCS del enlace troncal utilizan el canal para transmitir señalización de telefonía e información de control, mediante un protocolo orientado a un mensaje con encapsulación como HDLC. Utilizando un protocolo orientado a mensaje, los esquemas CCS ofrecen una tabla casi ilimitada de posibilidades para la señalización y el control de telefonía. Ejemplos de estos elementos que CCS proporciona incluyen lámparas indicadoras de mensaje en espera en los teléfonos equipados apropiadamente, identificación de parte que llama y parte llamada e incluso intercambio de información de enrutamiento de llamadas entre PBX.

Los mensajes sólo se transmiten a través del canal de datos CCS cuando hay información que enviar, como un cambio de estado en una interfaz de enlace troncal (por ejemplo, on-hook a off-hook), o un reconocimiento de información anterior. Contraste este comportamiento con CAS, que proporciona una corriente constante de información sobre cada canal de audio mediante los bits de señalización construidos en la estructura de la trama.

Hay tres categorías de protocolos CCS la telefonía:

- Patente de vendedor
- RDSI
- Sistema de señalización privado 1 (PSS1, *Private Signaling System*)

2.4 Tasación

Según el Reglamento de la Ley de Telecomunicaciones de la SIGET, el Plan de Tasación se define como el plan técnico fundamental adoptado por un operador, que determina la modalidad técnica que servirá de base para el cobro a los usuarios finales por el uso de una red de telecomunicaciones, en función de parámetros como la duración de las comunicaciones, la distancia asociada, el tipo de servicio, y otros, según corresponda.

Según la recomendación ITU-T Q.722 se debe tasar una llamada telefónica a partir de cuando su establecimiento se ha completado, es decir cuando el abonado al que se llama contesta, y el cobro debe hacerce al abonado llamante (a menos que se llame a un número de servicio especial como emergencias u otro número sin cobro para el abonado llamante). Como se mencionó en el capítulo I (1.4.1), con la señalización de línea se conoce el estado de una conexión, de manera que es la señalización de línea la que indica cuando el abonado al que se llama descuelga su teléfono y contesta.

En el plan de numeración los números están agrupados en series numéricas o lotes según el tipo de servicio que se requiera. Los diferentes tipos de servicios contemplados en el plan de numeración son los siguientes:

- Servicio de larga distancia internacional por operadora de cobro revertido.
- Servicio de llamadas a diferentes destinos en forma semiautomática a través de sistemas prepagados.
- Servicio de sistema multiportador.
- Servicio de telefonía móvil.
- Servicio especial.
- Servicio telefónico automático con terminacion internacional con sobrecuota para el abonado A.
- Servicio telefónico automático con terminacion nacional con sobrecuota para el abonado A.
- Servicio telefónico automático de cobro revertido internacional.
- Servicio telefónico automático de cobro revertido nacional.
- Servicio telefónico fijo (zona central).
- Servicio telefónico fijo (zona metropolitana).
- Servicio telefónico fijo (zona occidental).
- Servicio telefónico fijo (zona oriental).

Los cobros de cada uno de estos servicios están regulados por la SIGET, que publica en su sitio web las tarifas máximas que puede aplicar cada operador. Queda a discreción del operador si aplica tarifas planas o diferenciadas según la distancia (llamada local o internacional), hora en que se hace la llamada (horario de tarifa reducida o de tarifa plena), tipo de red a la que se llama (fija o móvil), etc.

A manera de ejemplo, siendo CTE S.A de C.V. el operador de telefonía fija con la mayor cantidad de abonados[10], se muestra en la siguiente tabla las tarifas máximas aprobadas por SIGET en el segundo trimestre de 2006[11]:

Tipo de llamada	Costo por minuto (con IVA)
Local plena	$0.0265
Local Reducida	$0.0195
Nacional plena	$0.0446
Nacional Reducida	$0.0321
A celular	$0.3010
Internacional a Estados Unidos	$0.42

Tabla 2.1 Tarifas máximas autorizadas por SIGET a CTE S.A. de C.V. en 2006

Tarifa plena	De lunes a sábado de 7 a.m. a 6:59 p.m.
Tarifa reducida	De lunes a sábado de 7 p.m. a 6:59 a.m. Domingo y festivos nacionales todo el día

Tabla 2.2 Horarios de tarifas de CTE S.A. de C.V. desde 2006

Nota: en el software del Tasador de Telefonía Fija no se consideró en el horario de tarifa reducida el día domingo completo ni los festivos nacionales para ningún operador.

[10] El Diario de Hoy, 26 de enero de 2010, página 9

[11] http://www.siget.gob.sv/images/documentos/telecomunicaciones/tarifas/cte_iisem___20061555.pdf

CAPÍTULO III

3. DESCRIPCIÓN DE LAS ETAPAS DEL HARDWARE DEL TASADOR DE TELEFONÍA FIJA

El presente capítulo trata acerca de cada una de las etapas del hardware de la aplicación en cuanto a funcionamiento, características y ventajas para tener una idea en grosso modo de cómo se ha atacado el problema. El hardware tiene como función almacenar en un archivo de texto los números telefónicos llamados, el inicio y el fin de las llamadas. Luego estos datos se transfieren mediante puerto serie a la PC para ser procesados por el software creado en lenguaje MATLAB.

La figura 3.1 muestra el diagrama de bloques del hardware.

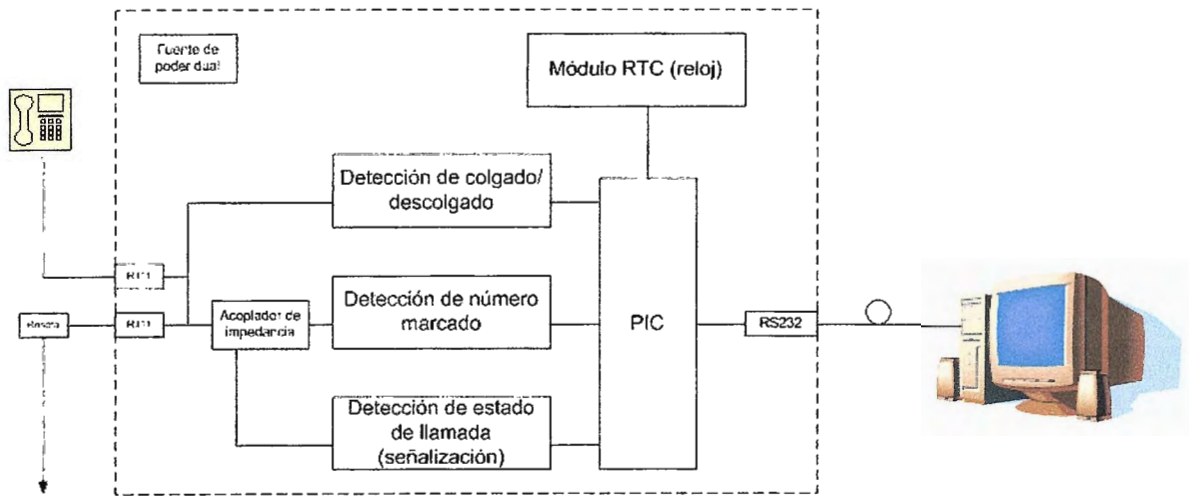


Figura 3.1 Diagrama de bloques del hardware del Tasador de Telefonía Fija

3.1 Detección de colgado y descolgado

La detección de colgado y descolgado es en principio la parte más fundamental antes de iniciar una comunicación telefónica debido a que se necesita saber el estado inicial y final del abonado que llama (abonado A) para fines del procesamiento completo de una comunicación telefónica. Después de determinar

cuándo una comunicación telefónica ha sido iniciada y finalizada se puede determinar la variable tiempo de duración de la llamada telefónica para después tasarla. En el bucle del abonado A se puede determinar cuándo se descuelga el auricular, en qué momento el abonado B contesta y cuándo el abonado A concluye la comunicación sin mayor problema ya que el circuito detector del estado de la llamada muestra el estado de la misma a su salida, pero no así cuando el abonado B ha concluido primero la llamada en caso de haberse establecido una comunicación telefónica, es decir si el abonado B colgó primero el abonado A escuchará un silencio y posteriormente un tono de ocupado, existe entonces un tiempo de algunos segundos que ya no pertenecen a la comunicación telefónica que se estableció y por lo tanto se tienen que eliminar estos datos que estarían demás.

3.1.1 Funcionamiento

En la línea telefónica existen -48 V que permanecen mientras no se ha descolgado el auricular, pero cuando se descuelga existe un cambio de nivel de voltaje y se presenta el tono de invitación a marcar. Si se vuelve a colgar el auricular la línea telefónica vuelve al estado inicial de los -48 V. Es a partir de estas condiciones que se emplea esta etapa por medio de un circuito sencillo para la detección de estado de colgado y descolgado, la cual está constituida por un optoacoplador encargado de detectar el estado inicial y final de una comunicación telefónica.

La función que realiza esta etapa es que cuando en la línea telefónica estén presentes los -48 V proporcionará a su salida como respuesta un nivel de tensión eléctrica de cero voltios ó 0 lógico (ver figura 3.2) y cuando se descuelgue el auricular presentará una salida de 5 V ó 1 lógico debido al cambio de nivel de voltaje que presenta la línea al momento de descolgar (ver figura 3.3). Este instante de tiempo en que ocurre el cambio más la permanencia en este estado son llevados como datos de información a la etapa del microcontrolador que se encargará de almacenar dichas variables y procesarlas junto con las demás variables de llamada de las demás etapas y que de igual manera lleguen a los puertos de entrada y salida correspondientes del microcontrolador.

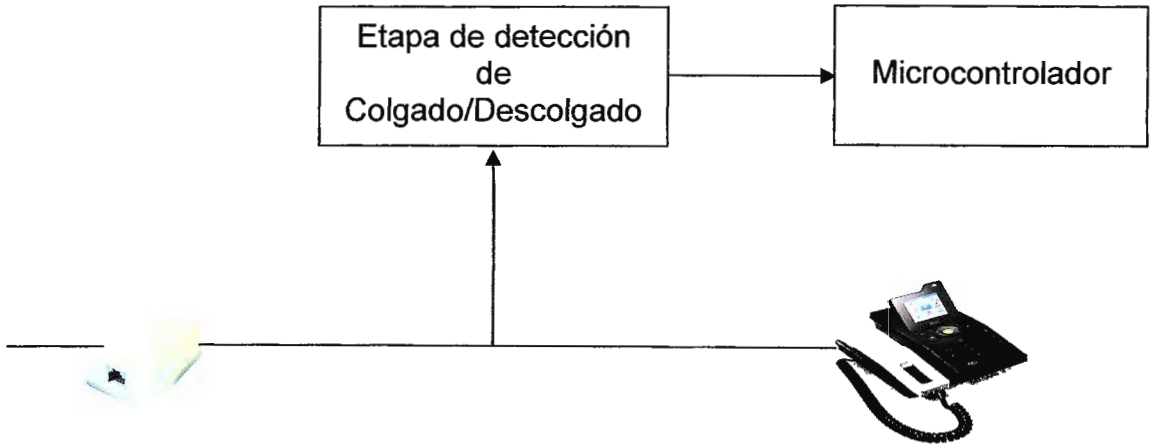


Figura 3.2 Proceso de detección de la línea telefónica en estado de colgado.

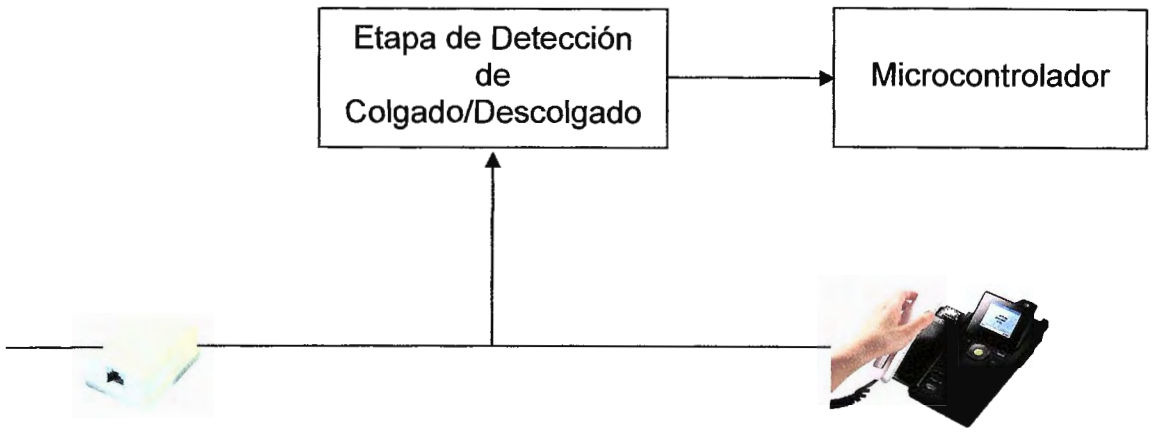


Figura 3.3 Proceso de detección de la línea telefónica en estado descolgado.

3.1.2 Características

Una característica de este detector es que puede ser una etapa independiente de las demás que constituyen todo el sistema ya que funciona solamente con la línea telefónica y está situada antes de las demás etapas y su resultado es llevado de una sola vez al microcontrolador, teniendo así un respaldo que garantice una buena captura del proceso de una llamada telefónica. Pero la característica principal de esta etapa es que da como resultado sólo dos posibles estados, está colgado el auricular o está descolgado, si se descuelga el teléfono cambia de estado lógico 0 a 1 y permanece así hasta que la línea vuelve al estado anterior cambiando también la respuesta a su salida. Es una etapa sencilla con pocos elementos, pero la información que arroja es trascendental para procesar todos los datos posibles en el desarrollo de una comunicación telefónica.

3.1.3 Ventajas

Se obtiene una salida de lógica binaria, lo cual resulta ideal para procesar en el microcontrolador. Su implementación es accesible porque sólo consta de un elemento optoelectrónico y un par de resistencias que proporcionan la corriente adecuada para alimentarlo para que funcione correctamente. Por ser una etapa optoelectrónica y sobretodo por ser un optoaislador se logra un aislamiento eléctrico entre los circuitos de entrada y salida, mediante el optoacoplador el único contacto entre ambos circuitos es un haz de luz.

3.2 Detección de número marcado

La detección del número marcado es esencial para determinar a qué número correspondiente al abonado B se marcó y así tener un información adicional que complementa a la información del sensado de la comunicación telefónica establecida o aún si no se estableció para efectos de tasar, ya que dependiendo de a qué número se ha marcado así son las diferentes tarifas, las cuales varían de acuerdo a los operadores de telefonía y el ente regulador. Además se necesita

saber si la llamada es local, larga distancia nacional, internacional, a celular, emergencias, etc. Por ello es importante detectar el número marcado adecuadamente, quedando registrado y posteriormente procesado por la aplicación en la computadora que debe contener toda la información necesaria en una base de datos para poder determinar la tarifa finalmente.

Existe un plan de numeración en cada país con el cual se enruta a los diferentes destinos tanto nacionales como internacionales. Es una etapa entonces que ayuda a comprobar de manera más acertada el cobro que se debe efectuar por parte del operador.

3.2.1. Funcionamiento

Esta etapa recibe los tonos que son enviados por el teléfono del abonado A, los cuales son codificados en una combinación de dos frecuencias DTMF. Se utiliza un circuito integrado decodificador de tonos que a cada combinación de dos tonos le reconoce un dígito, es decir que genera un patrón con los tonos que recibe y da como resultado un número en código binario de cuatro bits que está reflejado en su interfaz de cuatro salidas binarias que deben ser llevadas al microcontrolador para ser incluidas en el programa y después para su posterior utilización en la aplicación de tasación en la computadora.

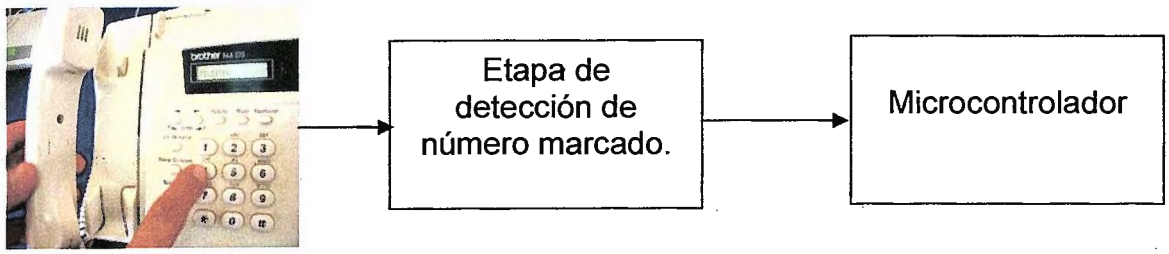


Figura 3.4 Marcación de dígitos y detección de dicha marcación.

3.2.2. Características

Al igual que la etapa anterior ésta también es independiente de las demás y solamente se toma en cuenta el número marcado si ha existido una comunicación telefónica como tal, de lo contrario se descarta el número marcado considerando también si la llamada es desviada al buzón de voz o si es máquina de anuncios la que responde. El resultado se toma en cuenta en el programa alojado en el microcontrolador para ser procesado juntamente con el tiempo de duración de llamada. Tiene la propiedad de procesar tantos dígitos como sean necesarios para establecer una llamada. Además otra característica es que si existe un dígito repetido al marcar es considerado válido ya que por su configuración se puede establecer que una vez marcado un dígito este pulso sea procesado internamente de una vez y casi al instante está preparada para recibir el siguiente dígito logrando un resultado del número exacto marcado a la salida de esta etapa.

3.2.3. Ventajas

En un principio esta etapa se realizaba con siete circuitos integrados, uno para cada frecuencia que genera cada dígito en las filas y columnas del teclado del teléfono, más sus arreglos para configurarlos de manera adecuada, así que se volvía una circuitería extensa. La principal ventaja de esta etapa es que consta de un solo circuito integrado específico para hacer esa actividad de detección de dígitos marcados, su configuración solamente está constituida por algunos elementos pasivos externos y económicos.

Las respuestas que se obtienen a la salida de esta etapa son fáciles de interpretar por el microcontrolador ya que son números binarios y son llevados directamente a éste sin necesidad de otras etapas adicionales.

3.3 Detección de estado de llamada

La detección de estado de llamada es la parte fundamental del analizador de tasación, pues acá se realiza el sensado de la respuesta del abonado B como tal, ésta detección indica entonces cuándo se ha iniciado una comunicación telefónica entre un abonado A y un abonado B. Si el abonado B ha contestado entonces la comunicación se ha establecido exitosamente, de manera que el circuito envía una señal de inicio de comunicación, pero si no se ha establecido la llamada el circuito también indicará que no se ha establecido dicha comunicación con el abonado B u otra situación se ha dado, ya que existen diferentes estados al intentar establecer una llamada.

Se ha utilizado un diseño que sea capaz de detectar el establecimiento de una llamada por sí mismo debido a que los operadores de telefonía fija en El Salvador no envían una señalización de línea al abonado A indicando que el abonado B ha contestado, pues la central solamente conecta a ambos abonados pero no envía una señal de retorno que indique el inicio de llamada, por lo tanto el circuito sensa la línea telefónica a partir de la información que está presente en la línea durante la comunicación telefónica. Las compañías de telefonía utilizan internamente señales digitales y éstas no pueden ser notadas por el abonado, pero las líneas telefónicas presentan una señal de audio que permiten identificar el proceso de estado de una comunicación telefónica y que sí pueden ser detectadas.

Las señales que se pueden escuchar al levantar el auricular permiten determinar el estado de la llamada telefónica. Dichas señales presentan patrones comunes a través de su frecuencia teniendo un orden preestablecido y presentándose de acuerdo a ciertas condiciones que ocurren durante la comunicación telefónica, y luego dejan de estar presentes. Esta señal de información es una frecuencia de 425 Hz aproximadamente. La duración de estas señales varía de acuerdo a cada país, por lo que es necesario referirse a las normas establecidas por el ente regulador en este país.

3.3.1. Señales presentes en la línea

Las compañías telefónicas informan sobre el estado de una llamada a través de señales de audio solamente, las cuales tienen características similares pero varían en su tiempo de duración. Inicialmente se consideró utilizar en el Tasador de Telefonía Fija la norma estadounidense para señalización de línea, es decir señales de doble tono (DTMF) que van desde 350 hasta 620 Hz, siendo la más utilizada las de 440 Hz. A continuación se definen estas señales para El Salvador, las cuales fueron tomadas como referencia (y de forma experimental) de la central telefónica de CTE S.A. de C.V. en Panchimalco en abril de 2010.

a. Tono de invitación a marcar

Frecuencia: 400 Hz
Cadencia: continua
Duración: de 10 a 20 segundos aproximadamente

Esta señal dura aproximadamente 20 segundos si no se marca ninguna tecla. Después de este periodo se presenta un tono de ocupado de 10 a 14 segundos y finalmente el teléfono se queda en silencio.

b. Tono de ocupado

Frecuencia: 444 Hz
Cadencia: emisión: 320 ms; silencio: 320 ms
Duración total: 15 segundos

Esta señal se presenta en la línea telefónica en diferentes estados de la llamada pero principalmente cuando al abonado B al que deseamos llamar está ocupado.

c. Tonos de marcado de dígitos

Estos tonos de marcados tienen una combinación de frecuencias DTMF por cada número en el teclado del teléfono y son enviados a la central para que ésta enrute hacia el destino con el que se desea comunicar. Estos tonos van del cero al

nueve incluyendo el asterisco y el numeral. A continuación están las características de dichos tonos.

Emisión: puede durar el tiempo que se tenga presionada la tecla, pero debe liberarse para poder ser enviado el dígito correspondiente.

Silencio: 16 segundos, después de esto hay un tono de ocupado y finalmente el teléfono se pone en silencio.

Frecuencias:

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

d. Tono de respuesta (ringback tone):

Frecuencia: 425 Hz

Cadencia: emisión: 1.2 segundos; silencio: 4.8 segundos

Duración total: 2 minutos

Este tono aparece cuando al abonado al que se desea llamar está disponible pero aún no ha contestado, y desaparece cuando se establece la comunicación o cuando el abonado B no contestó durando así 2 minutos. Puede suceder que conteste una máquina de anuncios en la que se envíe al buzón de voz. En este caso si el abonado A decide dejar un mensaje la llamada es procesada.

3.3.2. Funcionamiento

El detector de estado de llamada tiene como función principal detectar la respuesta de llamada, es decir el momento en que el abonado B ha levantado el auricular (pues es a partir de este momento que se da la comunicación telefónica) y contabilizar el tiempo de duración de la llamada. Además esta etapa indica el momento de finalización de la llamada si ha colgado la parte llamante (abonado A), aunque para esta parte es más factible basarse en la etapa de colgado y descolgado ya que es su función específica.

Entonces es posible conocer a la salida de esta etapa la respuesta de llamada por parte del abonado B, tiempo de duración de la llamada y finalización de ésta.

3.3.3. Características

La característica principal es que esta etapa usa sólo las frecuencias presentes en la línea para saber cuándo responde el abonado B. Muestra a su salida un cambio de estado lógico y lo mantiene estable mientras se da la comunicación telefónica y finalmente conmuta al estado lógico anterior cuando ésta finaliza, pudiendo tomar a su salida la variable tiempo de duración de la llamada que se ha realizado y la cual es la parte más fundamental para determinar los costos de la misma.

3.3.4. Ventajas

Es una etapa con solo un circuito integrado que trabaja con las frecuencias presentes en las líneas telefónicas, entregando a su salida niveles de voltajes de 0 y 5 V o estados de 0 lógico y 1 lógico que son factibles de procesar en la etapa del microcontrolador. Consta además de algunos componentes adicionales para configurarlo de manera que sea posible obtener dichos resultados como variables de salida del circuito, es decir a través de ellos se puede lograr calibrar esta etapa y obtener así los valores precisos a la salida.

3.3.5. Desventaja

Se sabe que no existe una señalización de línea de retorno en los operadores de telefonía fija, y que por lo tanto el circuito trabaja sólo con las frecuencias presentes en la línea telefónica. Debido a esa situación esta etapa genera un error de 4 segundos como máximo en detectar cuando el abonado B responde. En algunas ocasiones ese error se reduce a una cantidad menor en segundos o en otras ocasiones no habrá error, dependiendo si el abonado B conteste en el momento de silencio en los que el ringbacktone no esté presente.

3.4 Microcontrolador

Las etapas anteriores prácticamente sólo sirven para decodificar y digitalizar las señales eléctricas que se obtienen de la línea de telefonía fija, pues carecen de un elemento inteligente para poder procesarlas. Para ello se utiliza un tipo de microprocesador llamado microcontrolador.

3.4.1. Funcionamiento

Esta etapa se encarga de procesar las señales digitales obtenidas en la etapa de detección de auricular colgado/descolgado y la de detección de número telefónico marcado. Para ello el microcontrolador cuenta con un programa escrito en un lenguaje de programación de bajo nivel, el cual almacena el resultado que arroja dicho programa en un archivo de texto para su posterior análisis en otro programa elaborado en un lenguaje de programación de alto nivel que se ejecuta en una PC.

Además de procesar las señales obtenidas en las etapas ya mencionadas el microcontrolador obtiene el dato de la hora exacta en que inicia la llamada telefónica y la hora exacta en la que ésta termina. Para ello se auxilia de un módulo de reloj de tiempo real (real-time clock module en inglés, abreviado RTC module), el cual es un reloj de cuarzo cuya fuente de energía eléctrica es independiente a la alimentación del resto del circuito del tasador por lo que está

perennemente funcionando de manera que no sea necesario sincronizarlo cada vez que se deba llevar a cabo una captura de datos en una línea de telefonía fija.

3.4.2. Características

El microcontrolador utilizado es un PIC[12] (Programmable Intelligent Computer, que significa computadora inteligente programable), fabricado por la empresa Microchip Technology. El modelo del PIC es el PIC16F877[13], que cuenta con varios puertos de entrada y salida con los cuales puede comunicarse con las demás etapas del circuito del tasador, además de contar con memoria interna RAM y EEPROM y cuyo sencillo lenguaje de programación sólo tiene treinta y cinco palabras, además de que puede ser reprogramado según convenga a SIGET.

3.4.3. Ventajas

El hecho de que el hardware del tasador de telefonía fija cuente con un microcontrolador hace que no sea necesario utilizar una PC cuando se lleve a cabo una captura de datos en el domicilio u oficina de un abonado, lo cual resulta más cómodo para el técnico de SIGET. Además un microcontrolador tiene un consumo de energía eléctrica mucho menor que una PC, lo cual beneficia al abonado pues en caso de que se agotara la batería de la PC y sería necesario utilizar un adaptador AC/DC haciendo que el abonado incurra en un gasto extra de energía eléctrica.

Un microcontrolador ejecuta un programa escrito en lenguaje de programación de bajo nivel, ya que éste se ejecuta más rápido que un programa escrito en lenguaje de programación de alto nivel, por lo que se obtienen resultados de tasación de manera prácticamente instantánea.

---

[12] http://en.wikipedia.org/wiki/PIC_microcontroller
[13] Microchip technology Incorporated, "PIC16F87XA Data sheet",2003.

CAPÍTULO IV

4. ANALISIS Y DESARROLLO DEL CIRCUITO

Las líneas telefónicas presentan señales de audio para señalización, es decir con el objetivo de identificar el proceso de estado de una comunicación telefónica de manera que puedan ser escuchadas por el abonado.

Los operadores de telefonía internamente utilizan señales digitales para el direccionamiento y tratamiento de las llamadas telefónicas, pero esa información no la puede percibir el abonado. Anteriormente las señales eran analógicas y podían ser detectadas a través de los cambios de tensión en la línea telefónica, pero este sistema ya no es usado en El Salvador, de manera que se optó por estudiar el comportamiento de las únicas señales de información que están presentes en una línea de telefonía fija. Dichas señales se pueden escuchar con sólo levantar el auricular permitiendo así al abonado reconocer el estado de una llamada telefónica.

Estas señales son las que indican al abonado el estado de la llamada, y las cuales varían en su repetición y tienen un orden preestablecido presentándose sólo según si ciertas condiciones ocurren y en un período de tiempo específico, y cuando finalizan dejan de estar presentes en la línea. Una vez que son identificadas las señales es posible transformarlas a una secuencia de bits que estarán presentes sólo cuando aparece la frecuencia de 440 Hz para efectos de procesarlas como información en el microcontrolador. Los cálculos presentados en este capítulo se basan en la frecuencia de 440 Hz de la norma estadounidense, como se mencionó en 3.3.1, pero son también válidos para El Salvador debido a que se consideró un ancho de banda de 10% como se verá más adelante.

A continuación se presenta una serie de esquemas acerca de los diferentes estados de la comunicación telefónica representados en dos estados, 0 y 1 lógicos.

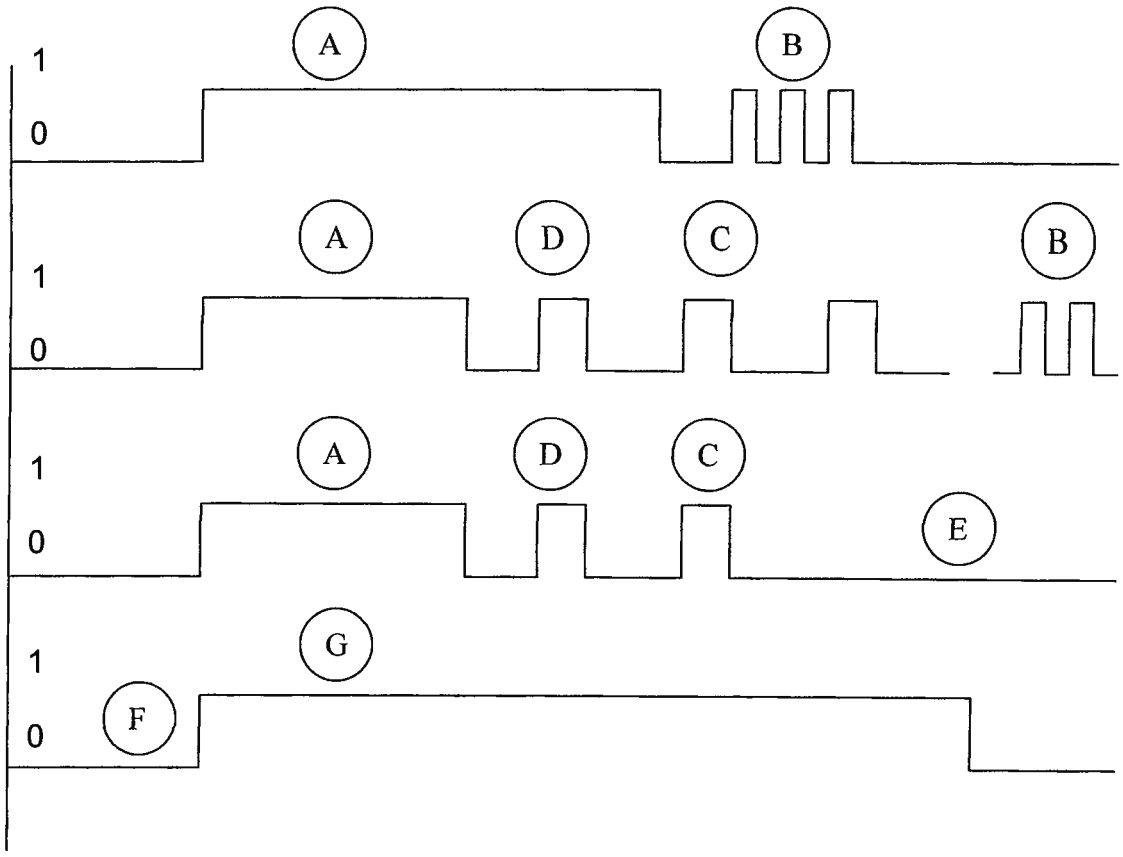


Figura 4.1 Establecimiento de una comunicación telefónica normal representada por bits, donde

A: Tono de invitación a marcar.

B: Tono de ocupado.

C: Ring back tone.

D: Marcación de dígitos.

E: Respuesta de llamada.

F: Estado de colgado

G: Estado de descolgado.

Cada pulso o 1 lógico representa la presencia de la señal telefónica de 440 Hz y que se presenta cada vez que se descuelga el teléfono. En la figura anterior se representa el establecimiento normal de una comunicación telefónica. En el primer esquema se levanta el auricular y se escucha el tono de invitación a marcar (A) durante 20 segundos aproximadamente, y si no se realiza una llamada se escucha

el tono de ocupado (B) durante 10 segundos y finalmente queda en silencio la comunicación hasta que se cuelgue el auricular.

En el segundo esquema se presenta el tono de invitación a marcar (A) más el lapso en el que se marcan los dígitos del número al que se desea llamar (D), posteriormente se presenta los ringbacktones (C) 10 veces y que duran 1.2 segundos cada uno por 4.8 segundos de silencio entre cada ringbacktone y si el abonado al que se llama no contesta se presenta tono de ocupado (B) y finalmente se queda en silencio.

Nuevamente en el tercer esquema se presenta el tono de invitación a marcar (A). En este caso también se marca el número de destino (D) y después de cierto numero de ringbacktones (C) el abonado al que se llama contesta (E), y la comunicación se mantiene hasta que uno de los dos cuelgue (referirse al último esquema).

El último esquema representa el estado de la línea de telefonía fija en cuanto a la tensión de la misma.

Luego de analizar este proceso normal de una comunicación telefónica se nota que el patrón de bits (señal de 440 Hz) está presente en la mayoría de los estados de la comunicación telefónica, excepto cuando contesta la persona a la que se llama o una máquina de mensajes (ya sea del abonado o de la operadora de telefonía).

De esta manera se establece una comunicación telefónica normal, en la que se centra el análisis y a partir del cual se obtendrá la información de las variables en cuestión (tiempo, número marcado, etc.) para su posterior tratamiento en el microcontrolador a nivel de código y después almacenar todo este análisis para ofrecer un reporte por medio de un software mostrando los resultados en texto para su fácil interpretación.

4.1 Funcionamiento del circuito electrónico y cálculo de valores de elementos pasivos

4.1.1. Acondicionamiento de la señal telefónica

En la figura 4.2 se muestra el diagrama del circuito implementado en el tasador de telefonía fija. J1A y J1B son los conectores hembras RJ11 en los que se conecta la línea de telefonía fija del abonado y el terminal telefónico, los cuales se pueden conectar en cualquier posición ya que éstos están en paralelo. El capacitor de 220 nF (C1) se utiliza para desacoplar la componente de DC de la línea telefónica, dejando pasar solamente las frecuencias de audio usadas por el circuito conectado a los conectores J1A y J1B. El transformador T1 se usa para aislar eléctricamente a la línea telefónica del circuito (esto es por norma) y debe tener una impedancia de 600 Ω a 400 Hz y una relación 1:1 (similar a los transformador es usados por los módems). Los diodos 1N4148 en antiparalelo (D1 y D2) se usan para limitar picos de ruido o audio altos.

4.1.2. Detección de colgado/descolgado

En las entradas de la línea telefónica también se conecta la etapa detectora de colgado/descolgado, pues como se ha mencionado antes su función es determinar el estado de la línea telefónica, si ésta ha sido descolgada o colgada para tener determinar con exactitud cuándo una llamada ha sido terminada (principalmente es el mayor enfoque en esta etapa), y para eso el optoacoplador 4N35 (OK1) está configurado con una resistencia de 47 KΩ a su entrada para limitar la corriente a un valor que funcione su led interno y éste proporcione una cantidad de luz suficiente para saturar el fototransistor interno produciendo a su salida un valor de 1 lógico, y si el led no se energiza pues el fototransistor no se satura y se obtiene un 0 lógico a su salida. La salida de esta etapa, que es el colector del fototransistor (pin 5 del 4N35), es llevada a uno de los pines del puerto C del microcontrolador, específicamente al pin RC5.

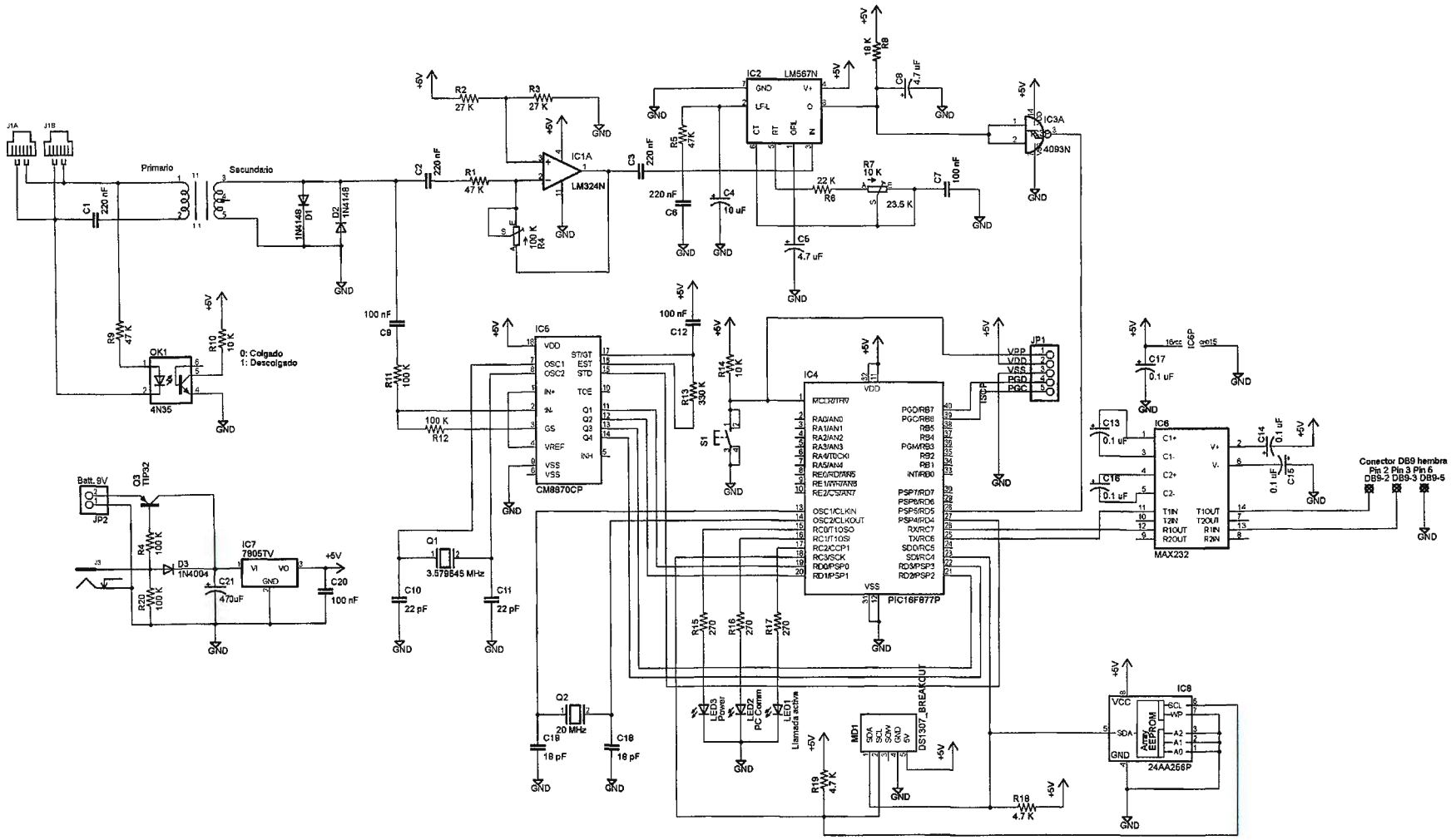


Figura 4.2 Diagrama del circuito del tasador de telefonía fija.

4.1.3. Detección de estados de llamada

El circuito integrado LM324 es un op-amp (amplificador operacional) configurado como inversor, para amplificar la entrada de audio telefónico (proveniente de los diodos D1 y D2 en antiparalelo), de unos 250 mV pico a unos 350 a 400 mV pico para enviarlo a la entrada del LM567, que es el decodificador de tonos. La ganancia A del op-amp es regulada a través del potenciómetro R4 con relación:

$$A = \frac{-R_4}{R_1}$$

Una de las características de este op-amp es que la alimentación puede ser tanto bipolar como simple[14], en este caso se ha utilizado con una alimentación simple, es decir con +5 V y GND solamente y su rango de voltaje diferencial es igual al voltaje de la fuente de poder. Esto favorece a la implementación del circuito ya que se utiliza una fuente de poder compacta.

El decodificador de tonos LM567 posee un VCO (Voltaje Controlled Oscillator, oscilador controlado por voltaje) el cual oscila a la frecuencia de la señal a detectar calculada con la formula:

$$f_O = \frac{1}{1.1RC_7}$$

Donde R es la resistencia serie de R6 más R7, y f_O es la frecuencia central del LM567 que debe ser lo más cercana posible a 440 Hz.

Suponiendo C7 como un valor comercial como 100 nF se despeja R, que resulta un valor cercano a 22 KΩ, por ello R6 es una resistencia fija.
R6 = 22 KΩ

Con estos valores teóricos de resistencia y capacitancia se obtienen 413 Hz. Luego se regula la frecuencia f_O con el potenciómetro R7 hasta alcanzar 440 Hz. Este tipo

[14] Fuente: Hoja técnica del LM324

de calibración se puede realizar con un osciloscopio en el pin 6 del LM567 para una mejor visualización de la variación de la f_O mientras se modifica R7.

Como la frecuencia de 440 Hz no se encuentra en el tono de ocupado en la norma estadounidense (formado por 480 y 680 Hz) se necesita detectar otra frecuencia, en este caso la de 480 Hz por ser la más cercana a 440 Hz. 480 Hz es un 10% mayor que 440 Hz, es decir que a partir de la frecuencia central f_o se necesita un ancho de banda de 10% para poder detectar 480 Hz. El Cálculo del ancho de banda (BW, bandwidth en inglés) del decodificador de tonos es el siguiente:

$$BW = 1070\sqrt{\frac{V_i}{f_o C_4}}$$

Donde:

BW es el ancho de banda expresado como un porcentaje de f_o

V_i es el voltaje de entrada rms

f_o es la frecuencia central del LM567, es decir 440 Hz en este caso

C4 se expresa en µF

Entonces:

$$BW = 1070\sqrt{\frac{V_i}{f_o C_4}}$$

$$10\% = 1070\sqrt{\frac{V_i}{440\times 10}}$$

$$V_i = \left(\frac{10}{1070}\right)^2 (440\times 10)$$

$$Vi = 0.384 V_{rms}$$

Ya conociendo el valor de V_i se ajusta la resistencia variable R4 hasta alcanzar 384 mV rms o 543 mV pico.

Nota: Como se mencionó en 3.3.1, para El Salvador las frecuencias presentes en la línea son 400, 425 y 440 Hz, siendo entonces f_o de 425 Hz. Con el ancho de banda de 10% se pueden detectar las frecuencias de 400 y 440 Hz.

El LM567 al detectar una señal de la frecuencia de f_o (440 Hz) presenta un cero lógico en su salida (pin 8), y alta impedancia al no detectar la señal. Para que al no detectar una señal similar a f_o presente un 1 lógico se utiliza la resistencia R8 conectada a entre el pin 8 y Vcc.

La función del capacitor C8 es que debido a que cuando el LM567 detecta una señal similar a f_o produce oscilaciones de unos y ceros lógicos, las cuales son eliminadas por este capacitor. Sin embargo esta capacitancia origina un retardo en la respuesta del circuito cuando la señal de 440 Hz desaparece. Este retardo se debe a la descarga del capacitor C8. Su periodo es:

$$T = C_8 \times R_8$$
$$T = 4.7 \times 10^{-6} \times 18000$$
$$T = 0.085\_segundos$$

Para poder invertir la salida del LM567 se emplea una compuerta lógica NAND del circuito integrado 4093 configurada como compuerta NOT obteniendo una secuencia de bits similar a la que se muestra en la figura 4.1. Para evitar errores a la salida de la compuerta por estados lógicos falsos a la entrada de la misma (voltajes en el rango de 2 a 3 V) se utiliza un disparador Schmitt, pues otras compuertas lógicas pueden oscilar en ese rango de voltajes.

La salida de esta compuerta lógica es llevada al puerto RD5 del microcontrolador (pin 28) como una variable central para poder determinar posteriormente mediante el programa almacenado en dicho microcontrolador el tiempo de duración de la comunicación telefónica.

El circuito integrado MT8870 es un receptor completo de DTMF que contiene un filtro de banda dividida y funciones de decodificador digital[15]. La sección de filtros usa técnicas de capacitores conmutados para grupos de filtros altos y bajos; el

[15] Fuente: Hoja técnica del MT8870

decodificador usa técnicas de conteo digital para detectar y decodificar los 16 pares de DTMF en códigos de 4 bits.
La configuración de este circuito integrado está dada por la hoja técnica del mismo y se puede implementar tanto para carga balanceada como para carga simple. En el caso del tasador de telefonía fija se ha implementado para carga simple. Dicha configuración tiene conectado un cristal de 3.579545 MHz en sus pines OSC1 y OSC2 (pines 7 y 8, respectivamente) para completar su circuito de oscilación interno.

El MT8870 proporciona 4 salidas en código binario correspondientes a los 16 posibles estados obtenidos al presionar un botón del teclado del teléfono (del 0 al 9, # y *; más A, B, C y D que no están disponibles en teléfonos comerciales). Estas salidas son Q1, Q2, Q3 y Q4 que son llevadas al puerto D del microcontrolador (de RD0 a RD3, respectivamente). Además se cuenta con el pin STD del MT8870 que indica cuándo ha sido marcado un dígito, es decir cuando ha sido enviado un par de tonos DTMF y éste ha sido registrado o actualizado. El microcontrolador detecta que ha sido marcado un dígito sin importar que éste haya sido marcado dos veces o más ya que dentro del plan de numeración existen números telefónicos que poseen una estructura con dígitos repetidos, pero de esta manera se garantiza que el número telefónico que ha sido marcado es en realidad un número válido dentro del plan de numeración.

De este modo se logra introducir al microcontrolador toda la información necesaria para tener un resultado más completo de las llamadas salientes que el abonado A efectúa y determinar con base a documentos oficiales de SIGET si la llamada debe ser cobrada o es de cobro revertido[16].

[16] Plan de numeración 2005, SIGET.

4.1.4. Microcontrolador y sus periféricos

4.1.4.1. Módulo de reloj de tiempo real

Es pues la etapa del microcontrolador con su programa el encargado de realizar los cálculos de las variables tiempo de duración de la comunicación telefónica, el número telefónico almacenado, hora, fecha, etc. Para ello se auxilia además de un modulo RTC (Real Time Clock, reloj de tiempo real) para obtener la hora y fecha exacta en que se realiza la llamada, además de la memoria EEPROM en la que se almacenan todos los resultados que procesa el microcontrolador, ya que esta memoria flash tiene más capacidad de almacenamiento que la memoria del microcontrolador.

El modulo RTC viene programado con la fecha y la hora, pero hay que actualizarla a la zona horaria de El Salvador. Posee una batería de litio CR1225 de 41 mA·h que mantendrá trabajando al módulo RTC durante 9 años sin una fuente de poder externa de 5 V. El acceso al módulo RTC es mediante el protocolo I^2C (Inter-Integrated Circuit, bus de comunicación en serie).

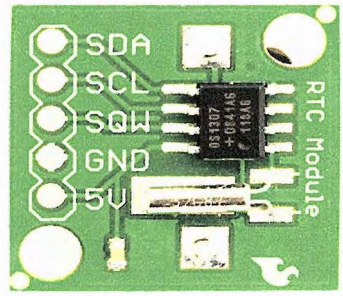


Figura 4.3 Circuito integrado DS1307, modulo RTC.

Algunas de las características del módulo RTC DS1307 son:

- Dos líneas para la interfaz I^2C.
- Hora en formato horas: minutos: segundos AM/PM
- Día, fecha y año.
- Compensación de año bisiesto.
- Calendario exacto superior al año 2100.

- Respaldo de batería incluído.
- 56 bytes de memoria no volátil disponible.

El acceso al RTC se obtiene implementando una condición de inicio (start) y proveyendo un código de identificación seguido por una dirección de registro, seguidamente los registros pueden ser accedidos secuencialmente hasta que una condición de parada (stop) sea ejecutada. Cuando el voltaje de alimentación Vcc del DS1307 cae por debajo de un umbral llamado VBAT el dispositivo conmuta internamente al respaldo de batería y bajo consumo de corriente, ver figura 4.4.

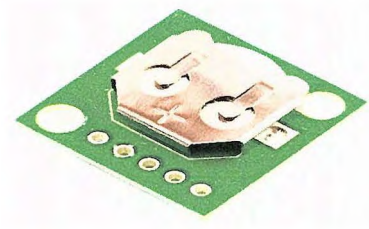


Figura 4.4 Batería de respaldo del DS1307.

Descripción de pines:
Vcc: Voltaje de alimentación principal
VBat +3V: Entrada de batería
GND: Tierra
SDA: Serial Data, puerto serie de datos
SCL: Serial Clock, puerto serie de señal de reloj
SQW/OUT: Onda cuandrada/Driver de salida

Vcc es la entrada de 5 V y cuando son aplicados dentro de los límites normales el dispositivo es completamente accesible y los datos pueden ser escritos y leídos. Cuando la batería de 3 V es conectada al dispositivo y Vcc está por debajo de 1.25 x VBAT, la lectura y escritura se inhiben, sin embargo la función de mantener el tiempo

no se ve afectada por la entrada de voltaje más baja. Cuando Vcc cae por debajo de VBAT la función de mantener el tiempo continúa en operación.

VBAT: es la entrada de voltaje de batería de litio standard de 3 V. Como se ha mencionado anteriormente, cuando el dispositivo RTC trabaja con este voltaje de entrada, no se tiene acceso a dicho dispositivo.

SCL (serial clock input): es usado para sincronizar los movimientos de datos sobre la interfaz serie SDA.

SDA (serial data input/output): es el pin de entrada o salida para las dos líneas de interfaz serial. Este pin es de drenaje abierto, el cual requiere una resistencia de pull-up externa.

SQW/OUT (Square Wave/Output driver): cuando se habilita el bit de SQWE a 1 lógico, el SQW/OUT da a su salida una de las siguientes cuatro ondas cuadradas de frecuencias 1 KHz, 4 KHz, 8 KHz, ó 32 KHz. Este pin también es de drenaje abierto así que se necesita una resistencia de pull-up.

4.1.4.2 Memoria EEPROM

El circuito integrado 24AA1025 es una memoria de sólo lectura eléctricamente programable y borrable (EEPROM, por sus siglas en inglés) de 128K x 8 bytes, capaz de operar en los rangos de voltajes de 1.7 V a 5.5 V y ha sido diseñado para desarrollar aplicaciones de bajo consumo de energía tales como comunicaciones o adquicisión de datos.

Descripción de pines:

A0 y A1: Las entradas de dirección A0 y A1 son usadas por el 24AA1025 para operaciones de dispositivo múltiple. Los niveles de estas entradas son comparadas

de con los bits correspondientes en la dirección de esclavo. La entrada A2 es un pin no configurable.

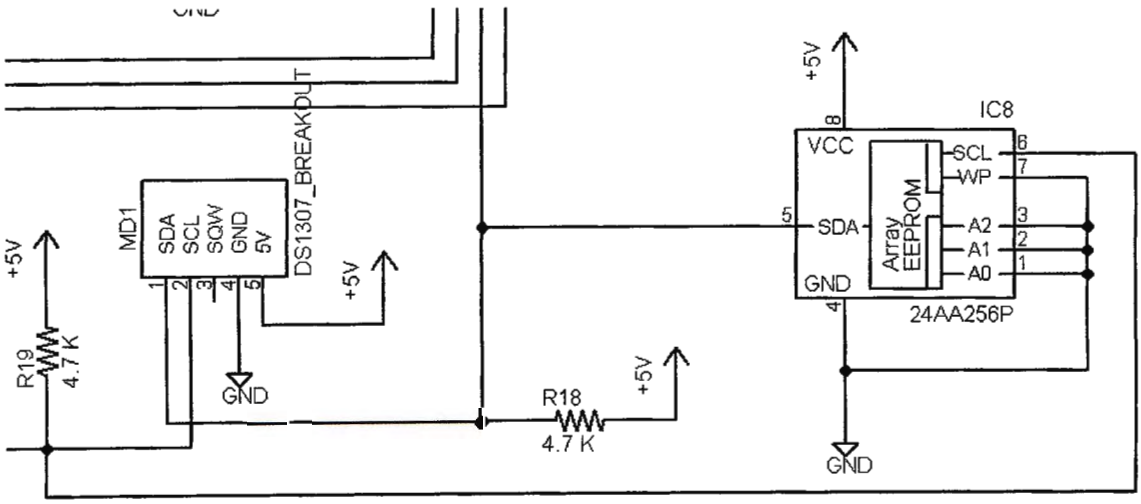
SCL: esta entrada es usada para sincronizar la transferencia de datos desde y hasta el 24AA1025.

SDA: esta es una entrada bidireccional usada para transferir direcciones de datos al interior del dispositivo y datos de salida. Requiere una resistencia de pull-up.

WP: este pin se conecta a Vcc o a Vss, si se conecta a Vss las operaciones de escritura se habilitan, si se conecta a Vcc las operaciones de escritura están inhibidas pero las operaciones de lectura no se ven afectadas.

4.1.4.3 Programación del microcontrolador

Para fijar la fecha y hora se creó un código en lenguaje C[17] para PIC auxiliándose de la hoja técnica de éste. El 16F877 posee 4 puertos, de los cuales el puerto destinado para estas etapas es el puerto C, que específicamente debe ser conectado entre sí con el RTC y la memoria flash 24AA1025 en los pines SDA y SCL ya que al utilizar estos dispositivos con un microcontrolador la configuración debe ir de la siguiente manera:



[17] Ver Código fuente en Anexos

Figura 4.5 Interconexión entre dispositivos, módulo RTC y memoria EEPROM.

El microcontrolador 16F877A posee 5 puertos (A, B, C, D y E) que pueden ser usados como entradas o salidas. En principio todo microcontrolador necesita un circuito externo que le indique la velocidad a la que debe trabajar, éste se conoce como oscilador o reloj y es muy simple pero vital para el buen funcionamiento del sistema.

El tipo de oscilador que ha sido utilizado es un cristal de 4 MHz porque garantiza mayor precisión y un buen arranque del microcontrolador. Internamente esta frecuencia es dividida por 4, lo que hace que la frecuencia efectiva de trabajo sea de 1 MHz por lo que cada instrucción se ejecuta en un microsegundo. El cristal de cuarzo debe ir acompañado de dos condensadores y se conecta de la siguiente forma:

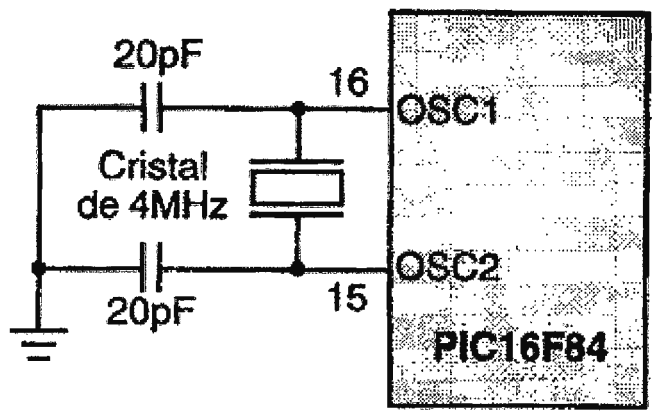


Figura 4.6 Conexión del oscilador al cristal. (Fuente: curso avanzado de microcontroladores)

En los microcontroladores se requiere un pin para reiniciar (reset) el funcionamiento del sistema cuando sea necesario ya sea por una falla o por diseño del sistema mismo. Este pin de reset en los PIC es llamado MCLR o Master Clear. El PIC 16F877A admite diferentes tipos de reset, pero para el tasador de telefonía fija se ha

optado por el reset de MCLR que se consigue llevando momentáneamente este pin a un estado lógico bajo, mientras que el watchdog WDT produce el reset cuando su temporizador sobrepasa la cuenta, es decir que pasa de 0FFh a 00h. De manera que si se quiere tener el control de reset del sistema se conecta de la siguiente forma:

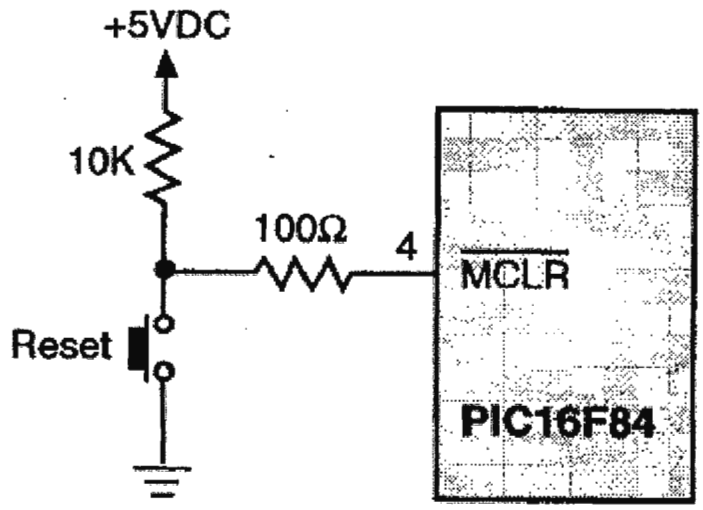


Figura 4.7 Conexión del botón de reset del microcontrolador. (Fuente: curso avanzado de microcontroladores)

Este microcontrolador posee dentro de su arquitectura una memoria de programa flash (14 bits de palabra) de 8K, memoria de datos de 368 bytes y memoria EEPROM de 256 bytes.

Para acceder al microcontrolador sin necesidad de sacarlo del circuito tasador y poder cargar el código de programa se ha creado un puerto que va conectado a los pines de PGC y PGD del puerto B del 16F877 y también al MCLR. En este puerto de comunicación se conecta el ISCP de Microchip para descargar el programa desde la PC.

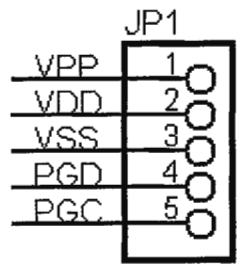


Figura 4.8 Conexión del puerto de comunicación ISCP al microcontrolador.

4.1.4.4. Comunicación serial entre el tasador y la PC

Finalmente debe existir la comunicación entre el tasador y la PC, ésta deberá contener la aplicación respectiva con una base de datos para realizar los cálculos de tasación mas detallada puesto que la información que sensa y los resultados que almacena el tasador solo podrán ser vistos y procesados en la PC como se planteó la solución previamente, de manera que el tasador enviará los resultados de las variables de interés a la PC a través de esta etapa.

Esta etapa es una interfaz de comunicación que consta de un circuito integrado MAX232 con su respectiva configuración dada por la hoja técnica (ver figura) y que posee solo algunos elementos discretos como capacitares externos de valores comunes en el mercado, dicha configuración es muy utilizada para comunicaciones seriales entre periféricos y la PC a través del puerto serie RS232. Como medio físico te transmisión se ha utilizado un cable que de un extremo posee el conector RS-232 y del otro posee un conector USB que es el que se conectará a la PC, así de esta manera se asegura de que la transferencia se pueda realizar en cualquier PC ya que hoy día todas las computadoras poseen puertos USB.

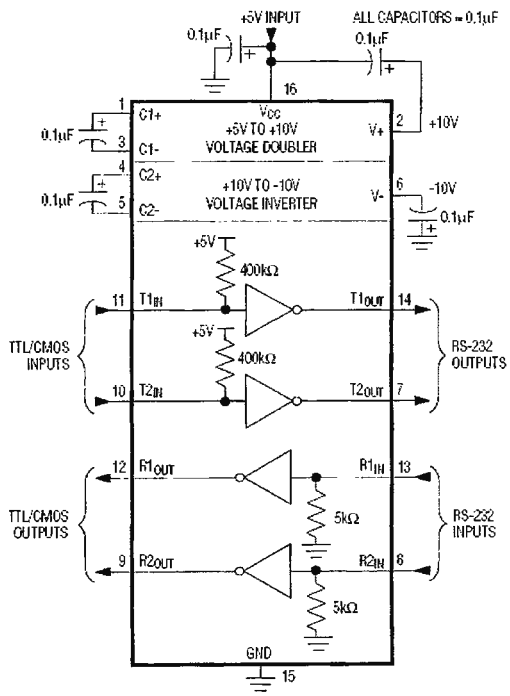


Figura 4.9 Configuración básica del MAX 232.

CAPÍTULO V

5. FUNCIONAMIENTO DEL TASADOR Y FASE DE PRUEBAS

Después de haber revisado el funcionamiento del circuito y de las diferentes etapas nos enfocamos ahora en la aplicación como tal del tasador.

Antes de hacer una medición se debe tener las condiciones adecuadas para dejar sensando al tasador en una línea fija residencial, como por ejemplo los conectores deben estar en buen estado así como las cajas de interconexión para evitar posibles falsos contactos. El tasador tiene la capacidad de almacenar 2048 registros de llamadas, es decir 2048 llamadas telefónicas establecidas, esto es debido a que la memoria EEPROM es de 256Kbit.
El cálculo es como sigue:

$$Num\_registros = \frac{[256 \times \frac{1024(bits)}{8\frac{bit}{Byte}}]}{[16\frac{Bytes}{\text{Re}gistro}]} = 2048\ registros$$

De manera que el tasador puede permanecer en una residencia tasando durante varios días consecutivos hasta utilizar los 2048 registros si se diera el caso.
Se debe tener en cuenta siempre que la batería de respaldo esté en buenas condiciones, ya que el tasador posee un dual power (ver figura 5.1.), debido a que muchas veces existen cortes de energía y la telefonía fija siempre funciona a pesar de ello. Por el contrario, la alimentación normalmente es a través de una fuente de poder de 12 V.

Antes de efectuar cualquier sensado en la línea del abonado A, se debe instalar el driver para el cable serial DB-9 a USB, efectuar la conexión del cable serial del tasador hacia la PC (ver figura 5.2.), almacenar el archivo ejecutable Tasador.exe en

la PC. Se corre el archivo ejecutable Tasador.exe, al correrlo debe parpadear el led amarillo, indicando que existe comunicación entre el tasador y la PC.

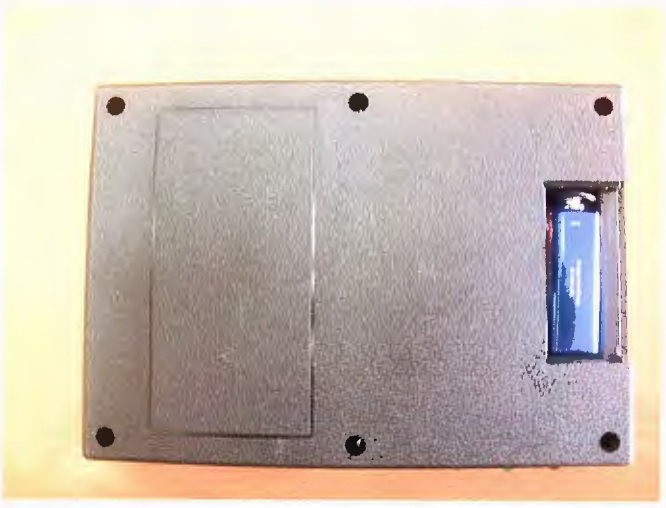


Figura 5.1 Vista anterior del tasador junto con la batería de 9V.

Figura 5.2 Conexión del tasador hacia la PC a través del cable serial.

El programa de la PC configura la hora del RTC del tasador con la hora del sistema de la PC, de esta forma se garantiza que el tasador tendrá una hora de referencia, dicha hora esta en formato militar.

Asegurarse que los RJ11 estén bien conectados y así garantizar que habrá un buen sensado, visualizar además que el led rojo de power del tasador esté encendido.

Después de sincronizado el tasador con la hora del sistema, se procede a la conexión de los cables físicos de la línea del Abonado A al tasador para comenzar ha realizar el sensado como se muestra en la figura 5.3.

Figura 5.3 Conexión de cables de la línea fija y teléfono.

El tasador posee en la vista frontal tres leds indicadores (ver figura), cuya función para cada uno es la siguiente:

- Led rojo: indica que ha sido conectada la alimentación y de igual forma si está con la batería de respaldo, el estado de este es permanente.
- Led verde: Parpadea al levantar el auricular y se queda estable al iniciar la comunicación telefónica, se apaga al finalizar dicha comunicación.
- Led amarillo: Es parpadeante cuando se corre el programa ejecutable de la PC y se hace una transferencia de datos.

Figura 5.4 Vista frontal del tasador.

Después que se han hecho las mediciones durante algún tiempo en la línea del Abonado, estas se puede transferir a la PC, para ello se debe abrir el archivo Tasador.exe, a continuación se detalla el proceso de descarga de información hacia la PC.

5.1 Aplicación de la PC

5.1.1. Procedimiento para sincronizar y verificar si existen datos en el tasador

- En principio se vuelve a conectar el cable serial a los equipos respectivos, como se explica anteriormente para realizar la descarga de información. Correr el archivo ejecutable Tasador.exe.

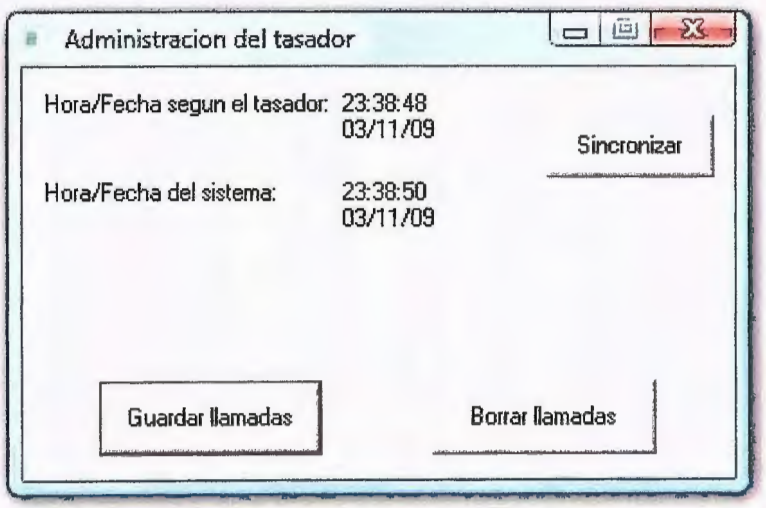


Figura 5.6 Archivo ejecutable para administrar al tasador desde la PC.

- Sincronización: Click sobre el botón de Sincronizar.

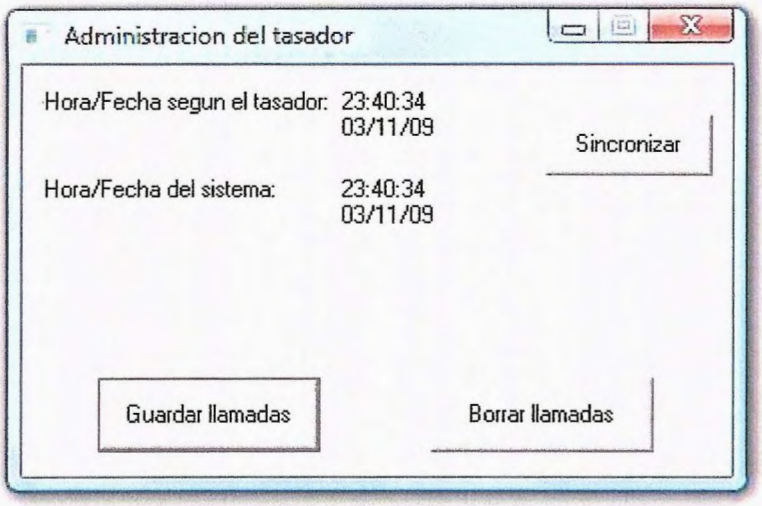


Figura 5.6 Inicio de proceso de sincronización.

- Verificamos si existen llamadas: click sobre el botón de Guardar llamadas.

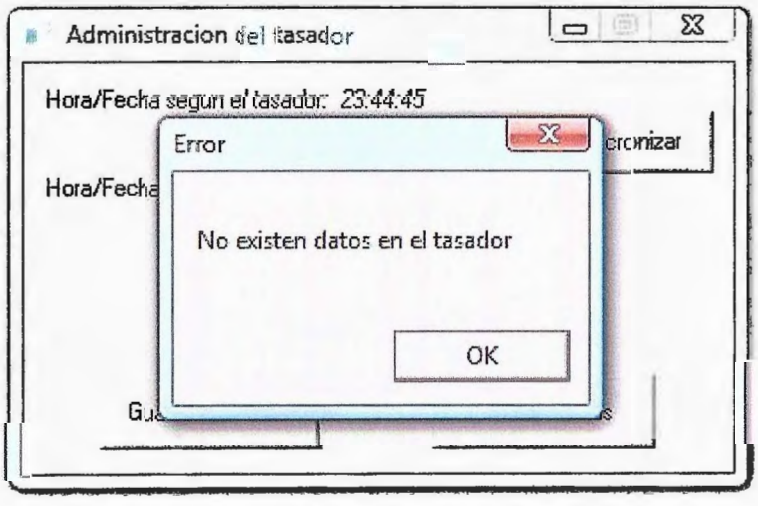


Figura 5.7 Verificación de datos en el tasador.

5.1.2 Procedimiento para la descarga del registro de llamadas

- Correr el archivo ejecutable Tasador.exe.

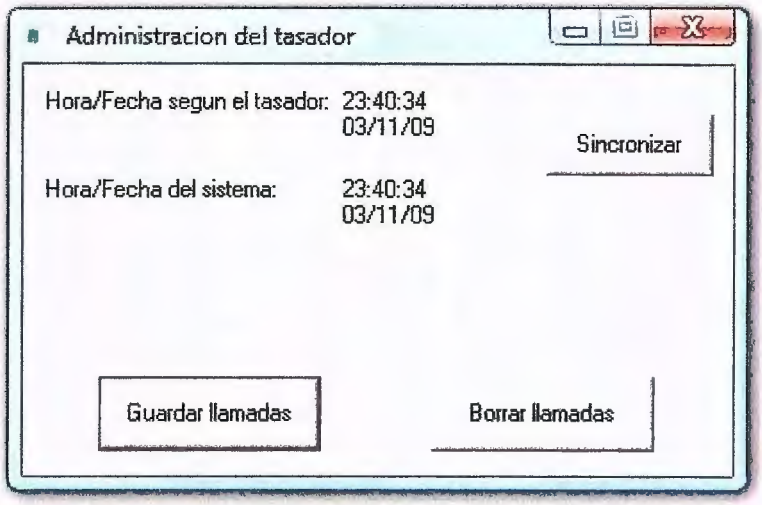


Figura 5.8 Archivo ejecutable Tasador.exe.

- Click sobre Guardar llamadas y especificar la ruta donde será almacenada la información del tasador.

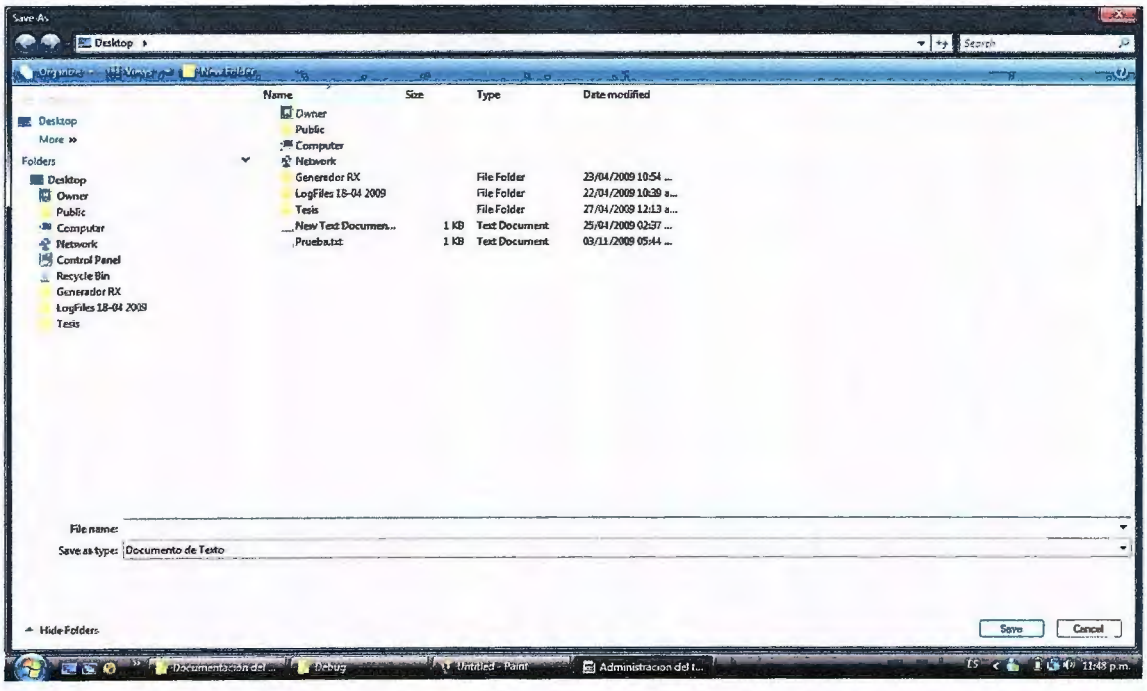


Figura 5.9 ventana de la ruta de almacenamiento de datos.

- Después de especificar la ruta, se confirma el éxito de almacenamiento de llamadas en la PC.

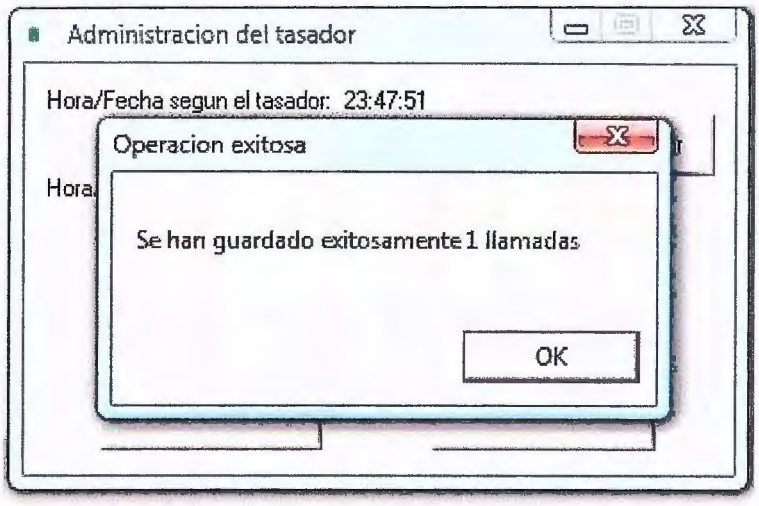


Figura 5.10 Confirmación de almacenamiento de llamadas.

- Después de almacenar la información, ésta se puede visualizar en formato TXT, cuyo contenido es: Numero del abonado B al que se ha llamado, Hora de inicio de la llamada, fecha de inicio de la llamada, hora de finalización de la llamada y fecha de finalización.

Figura 5.11 Resultados del tasador en archivo de texto.

- Si se desea seguir almacenando las llamadas, podemos almacenarlas con los pasos anteriores, y en el mismo archivo de texto si se prefiere.

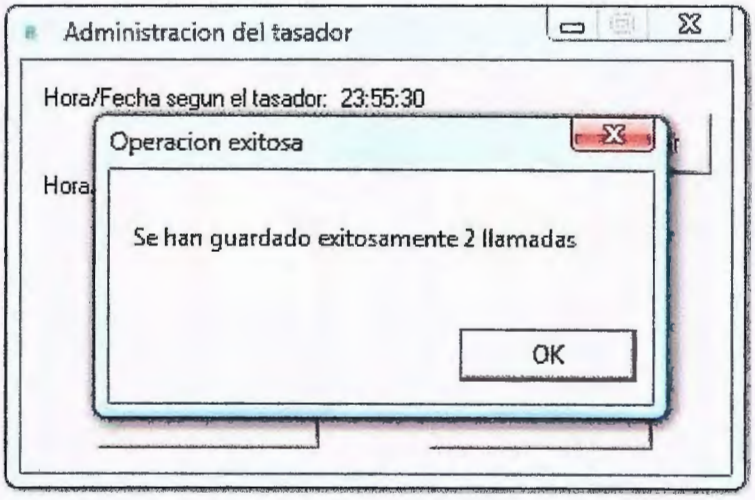


Figura 5.12 Confirmación de almacenamiento de datos en la PC.

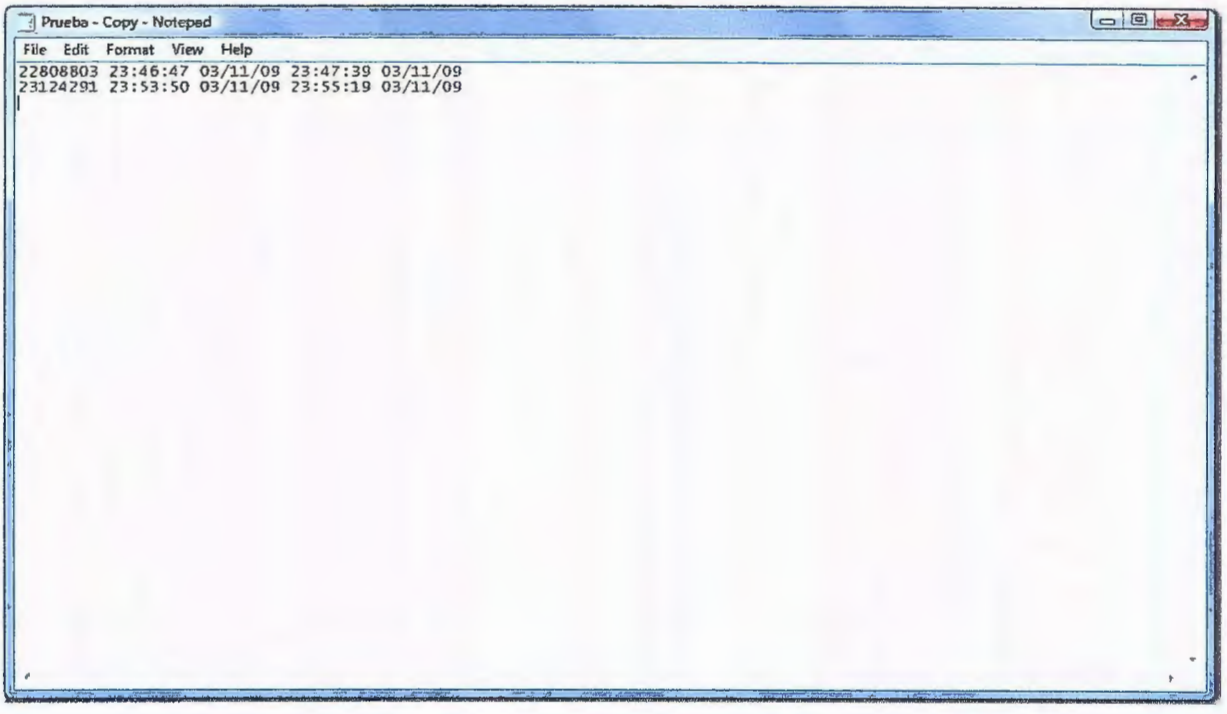


Figura 5.13 Continuación de almacenamiento de datos en la PC.

5.1.3. Procedimiento Para Borrar las llamadas

- En la ventana principal dar Click sobre el botón de borrar llamadas.

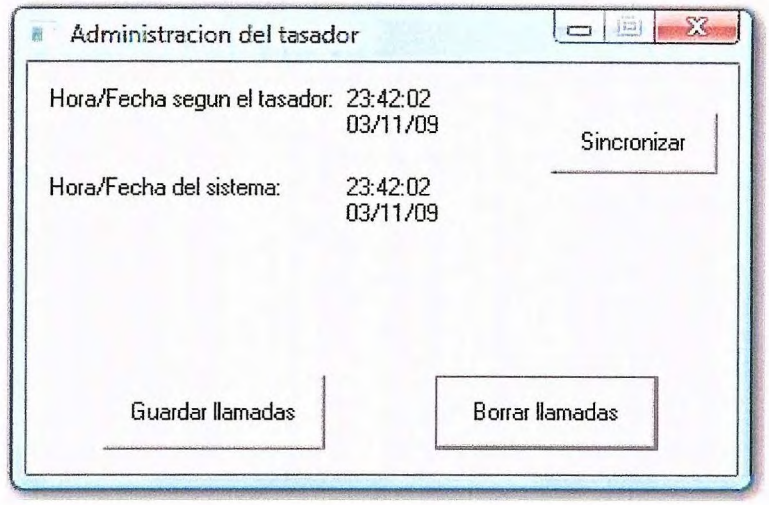


Figura 5.14 Archivo ejecutable Tasador.exe, y procedimiento para borrar.

- Confirmación de borrado de llamadas.

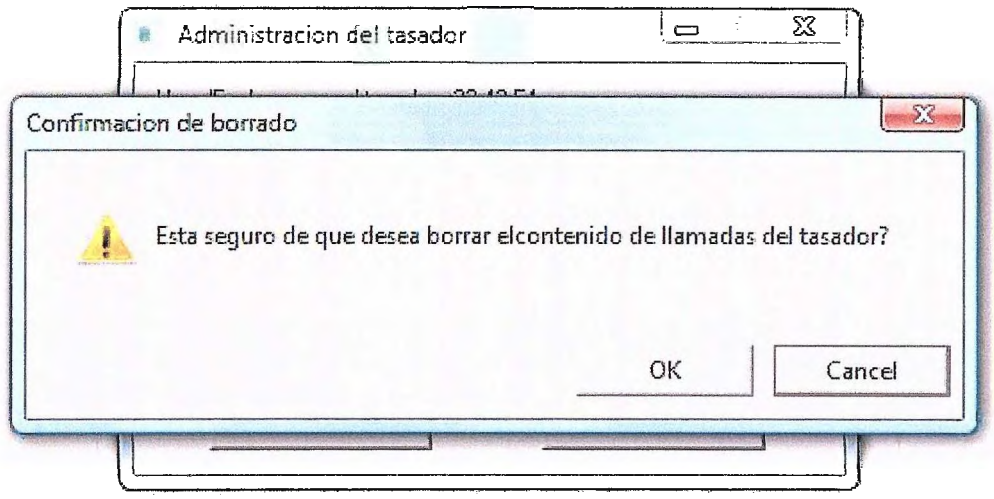


Figura 5.13 Interrogante para el borrado de de información en el tasador.

- Confirmación de la operación efectuada.

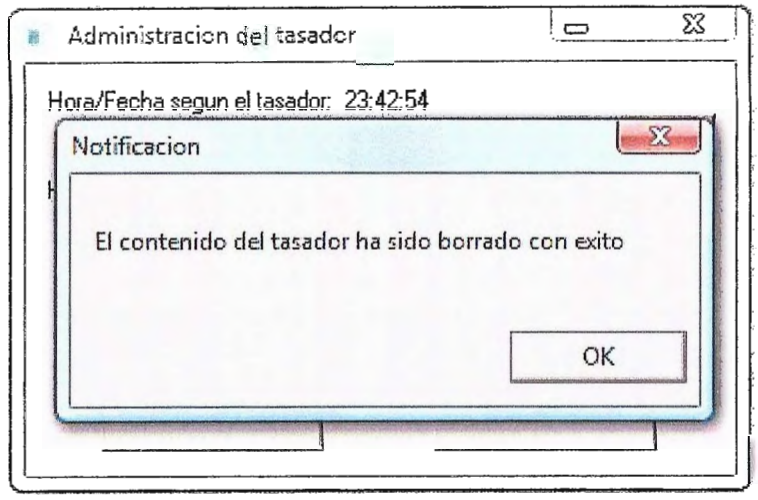


Figura 5.14 Confirmación de borrado.

5.1.4. Procedimiento Para el uso de la Aplicación en matlab menú_tasador

- La aplicación llamada menu_tasador es un archivo ejecutable elaborado en Matlab, para procesar la información anteriormente recopilada con el data logger. Al correr dicho ejecutable éste mostrará la siguiente ventana de menú:

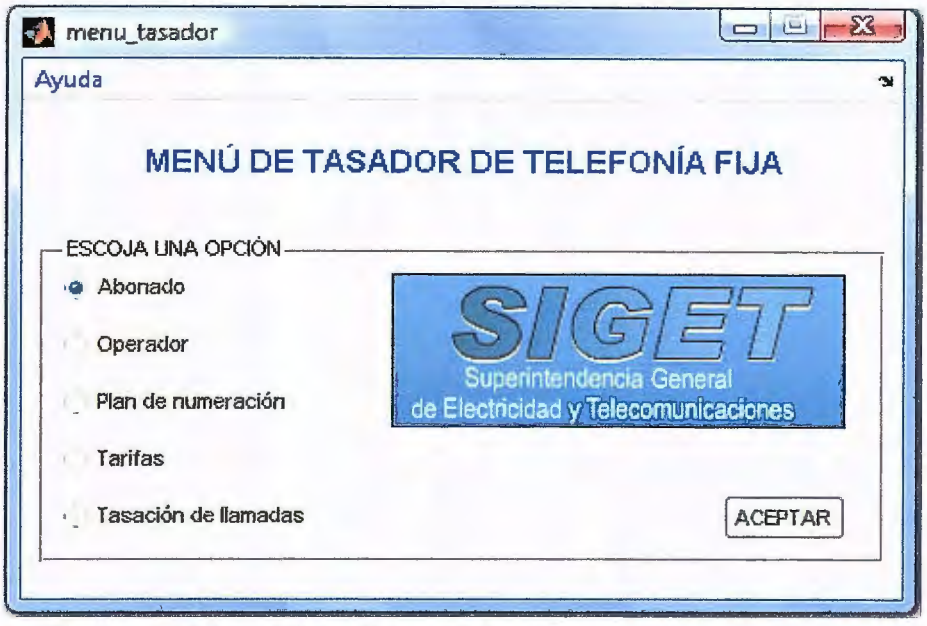


Figura 5.15 archivo ejecutable menu_tasador.

- Al seleccionar la primera opción del menú, se muestran las diferentes ventanas a continuación:

Figura 5.16 Creación de un abonado.

- Se procede a llenar los datos del abonado con el que se han realizado las mediciones.

Figura 5.17 Datos del abonado.

- También se puede modificar datos de los abonados antes creados, como el nombre, tipo de identificación, etc.

Figura 5.18 Modificación de datos del abonado.

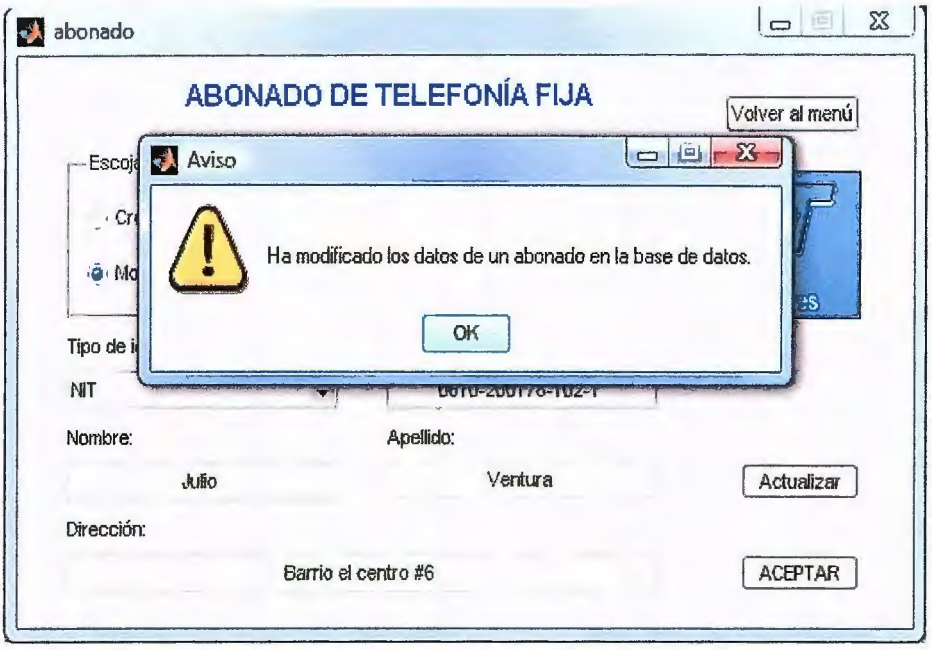


Figura 5.19 Confirmación de modificación de un abonado.

- Si se desea crear un nuevo operador, este deberá ser previamente autorizado por SIGET, con sus respectivos lotes numéricos, una vez hecho esto se pueden hacer modificaciones o eliminar si se requiere.

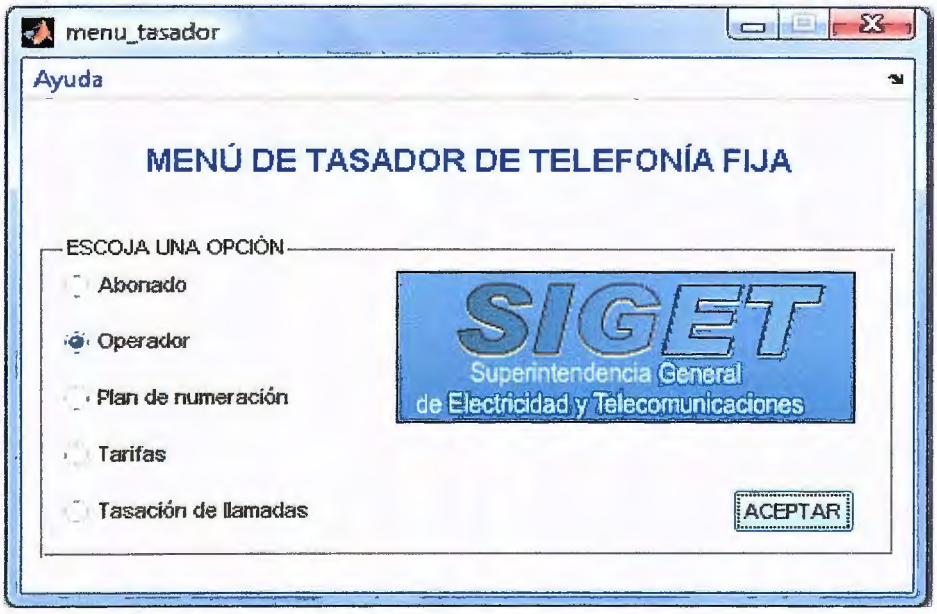


Figura 5.20 Opción para crear un operador.

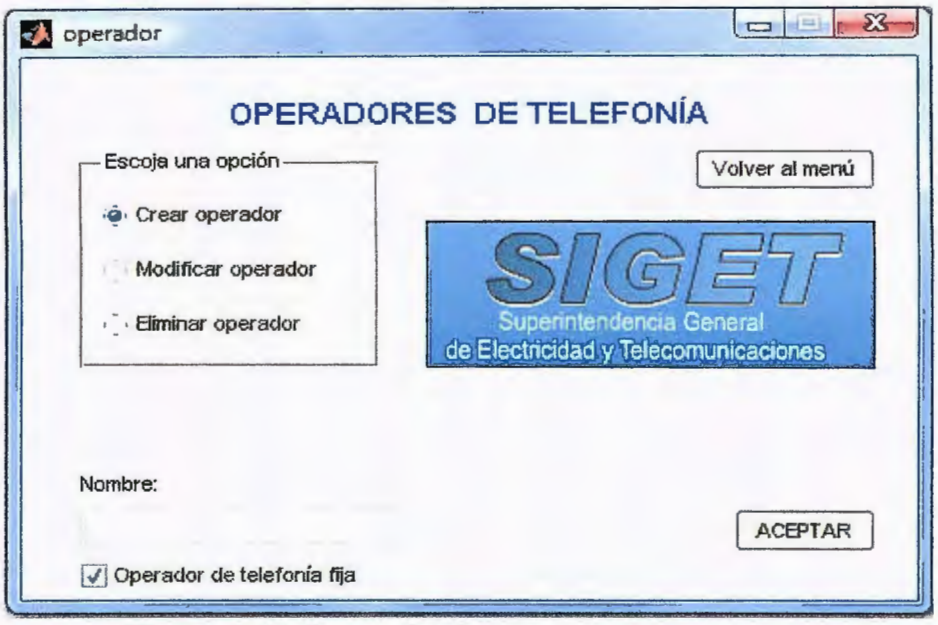


Figura 5.21 Creación de un operador.

- Si existe la aprobación de nuevos lotes numéricos, se puede crear en la base de datos un nuevo lote numérico, modificar o eliminar dicho lote, indicando el operador y el tipo de servicio en un menú desplegable:

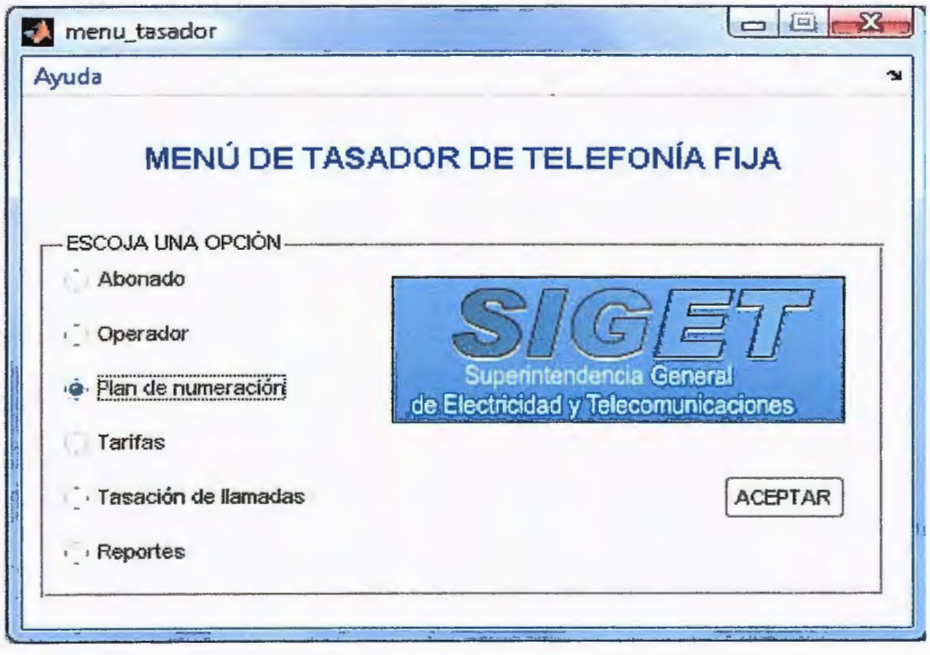


Figura 5.22 Opción de plan de numeración.

Figura 5.23 Opción para modificar el plan de numeración.

- La tarificación puede también ser modificada de acuerdo a las aprobaciones previas:

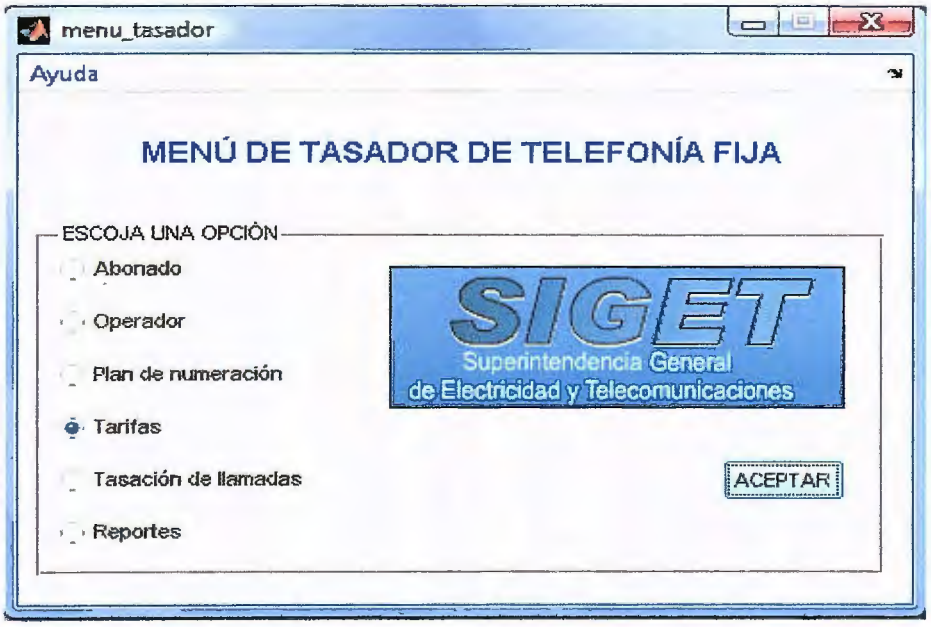


Figura 5.24 Opción para establecer tarifas de llamadas.

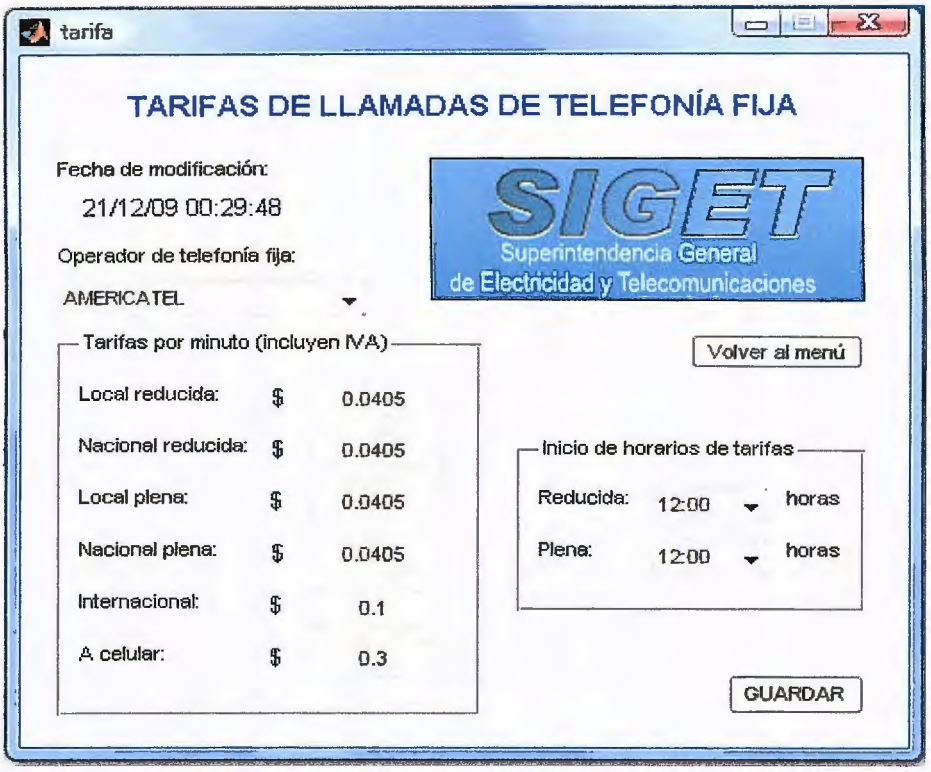


Figura 5.25 Campos relacionados a tarifas.

- Finalmente se procede a realizar una tasación de llamadas, por lo que se selecciona la opción de Tasación de llamadas.

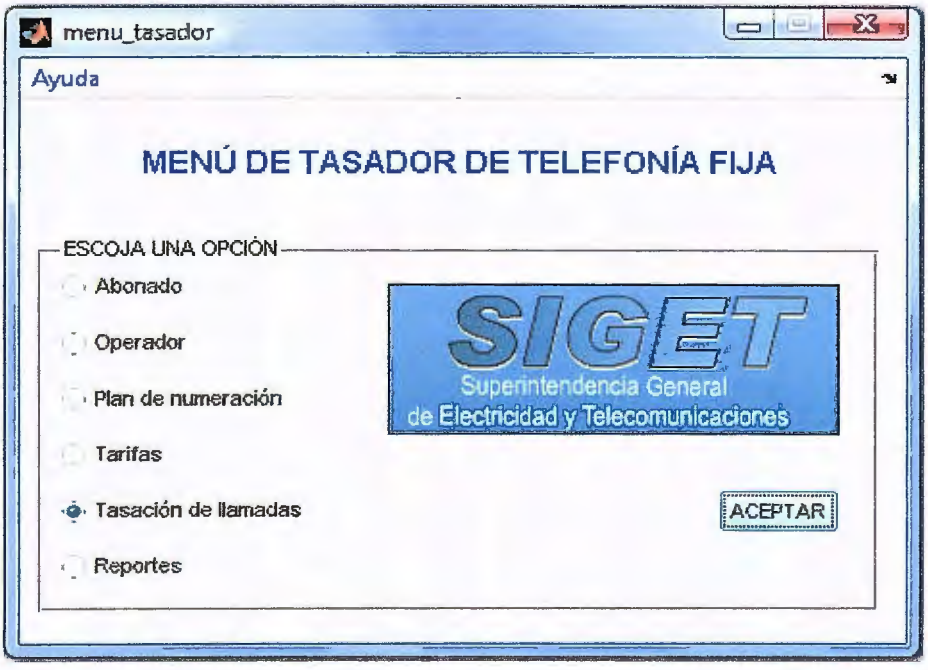


Figura 5.26 Opción de Tasación.

- Se completa el campo con el número de teléfono del abonado, y se da click en buscar operador para saber el operador que le presta el servicio, Click en buscar operador:

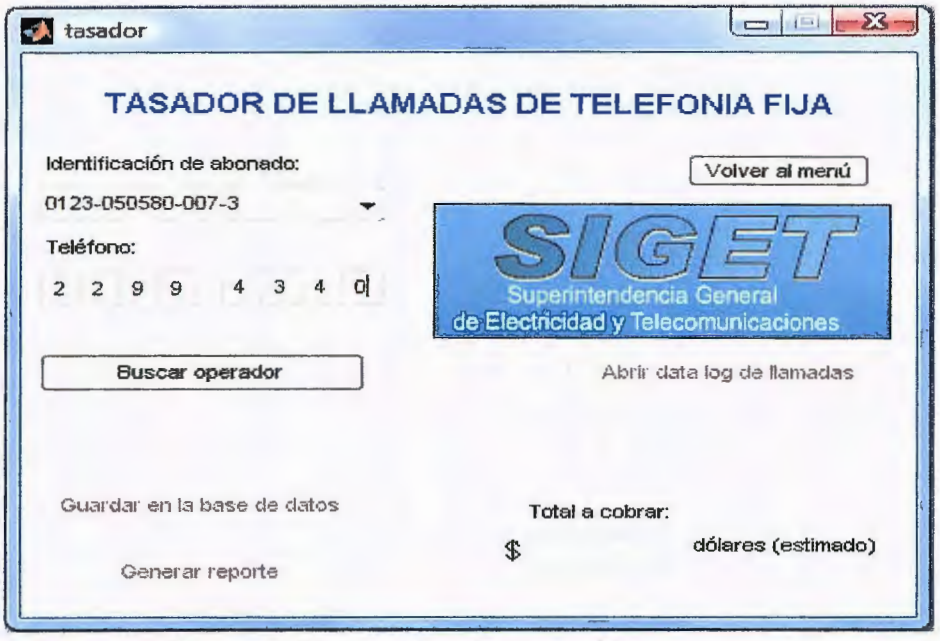


Figura 5.27 Opción de Tasación.

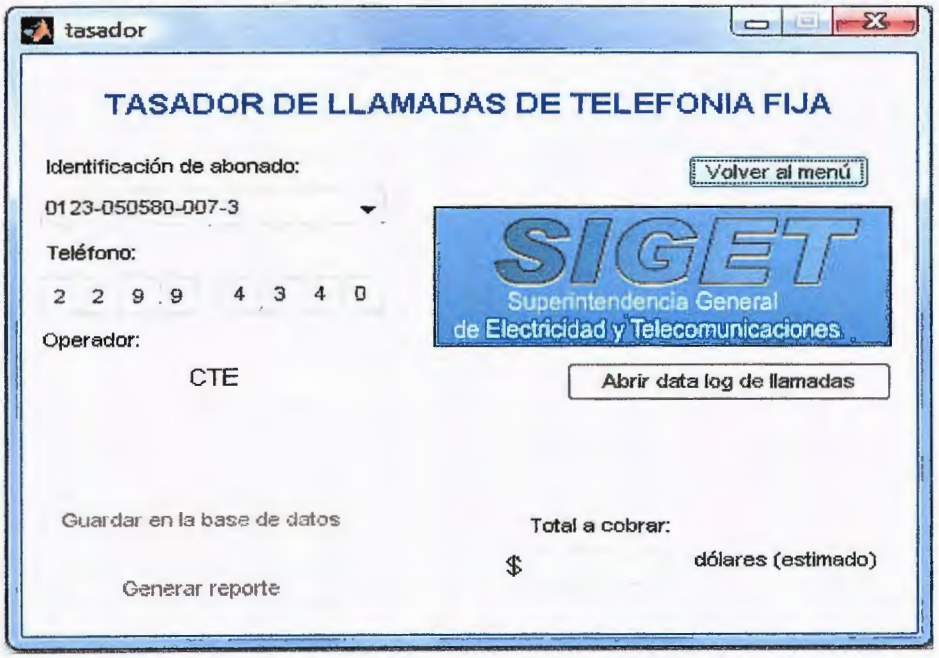


Figura 5.28 Identificación del operador a que pertenece el abonado.

- Se procede a abrir el data log de llamadas, es donde esta toda la información que se ha descargado previamente del data logger:

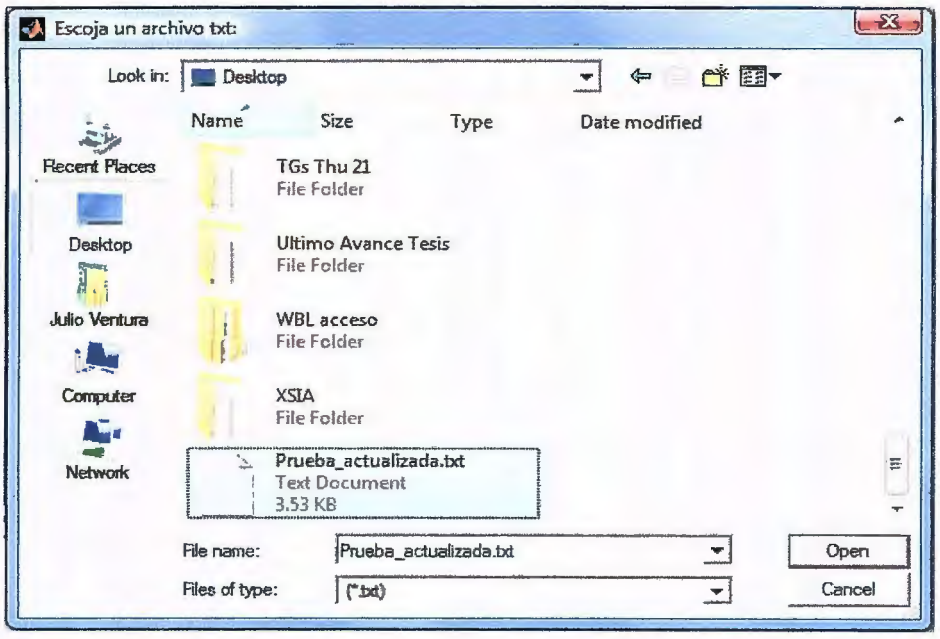


Figura 5.29 Data log de llamadas.

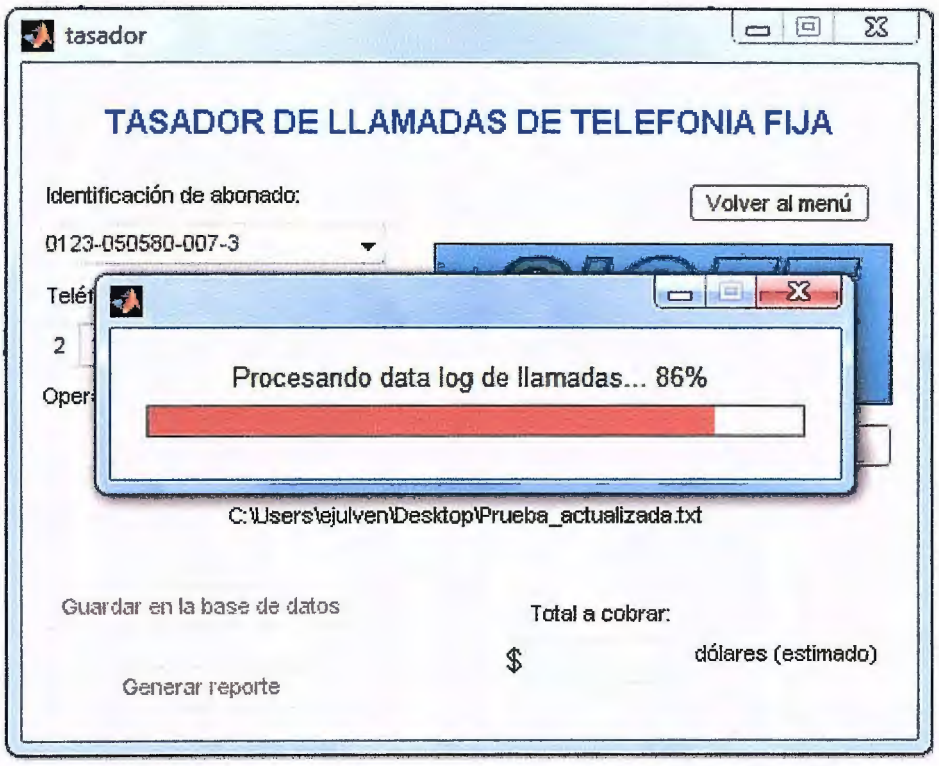


Figura 5.30 Procesamiento del data log.

- Los resultados se almacenan en la base de datos si se desea, además de generar un reporte de los resultados por categorías en formato HTML:

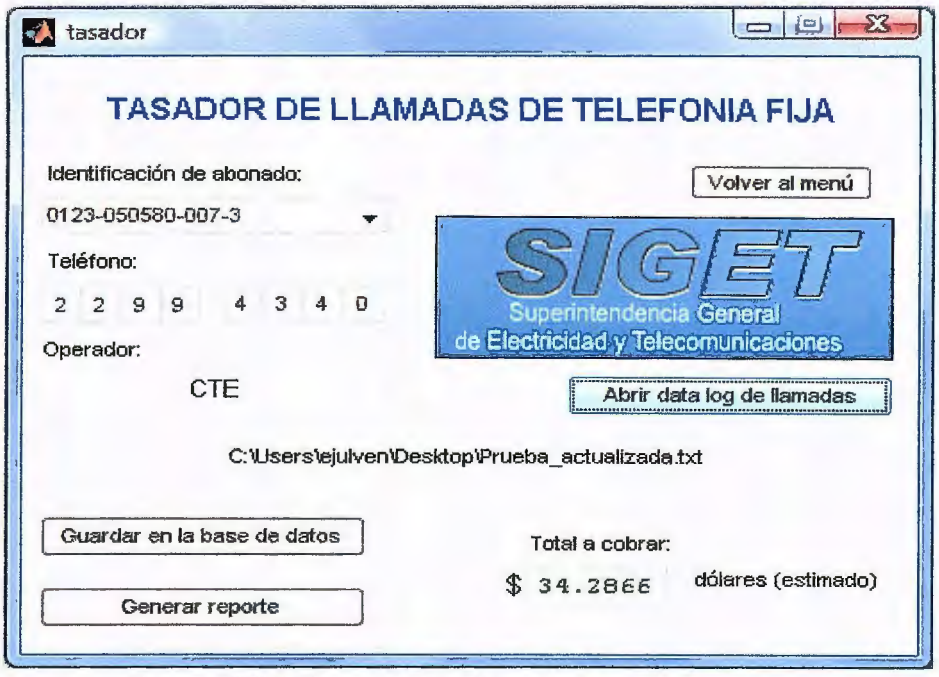


Figura 5.31 Total del cobro a efectuar.

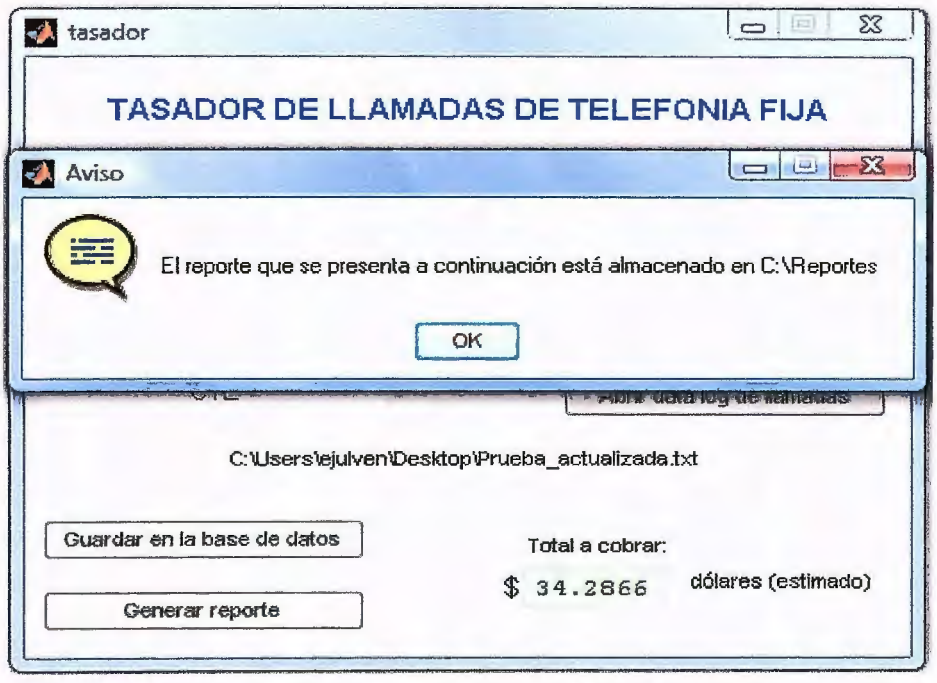


Figura 5.32 Confirmación de generación de reporte.

- Es necesaria la Generación de reportes para documentar las mediciones y para posteriores resoluciones, disponiendo de consolidados o resumenes.

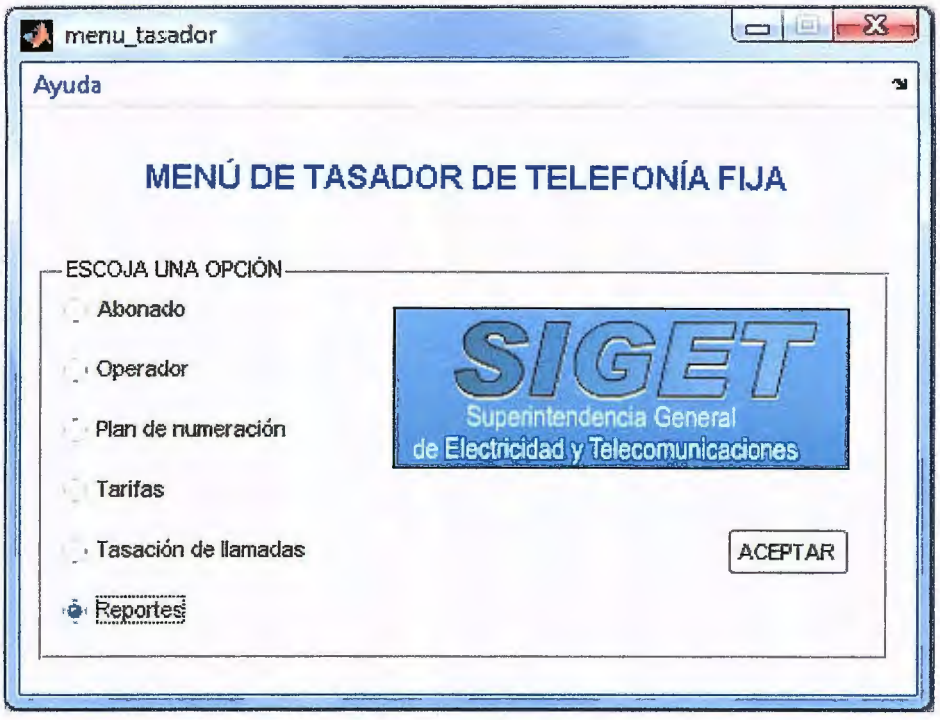


Figura 5.33 Opción de generación de Reporte.

Figura 5.34 Reporte.

CAPITULO VI

6 COSTO DEL PROYECTO

Elemento	Cantidad	Precio Unitario (USD)	Total (USD)
Componentes electrónicos	Diversos	125.01	125.01
Detector de tonos	5	2.89	14.45
Microcontrolador	1	5	5
Chasis	2	12.00	24.00
Circuito Impreso	1	5.00	5.00
Imprevistos	1	22.89	22.89
Total			**191.35**

Tabla 6.1 Costos del proyecto

CONCLUSIONES

- Con el desarrollo del analizador de tasación de telefonía fija, se beneficiaran muchos abonados que se han visto afectados por el cobro indebido del servicio de telefonía fija en todo el país, teniendo un mejor control sobre los operadores de telefonía fija en El Salvador.

- El Analizador de tasación de telefonía fija puede ser instalado en la residencia del abonado afectado durante un mes aproximadamente, teniendo este un respaldo de energía por afectaciones de cortes de energía, además la información puede ser descargada a la PC mediante la aplicación en cualquier momento, dicha información debe ser comparada con un informe detallado del servicio generado en las fechas correspondientes por parte del operador.

- El analizador está enfocado a los abonados que tienen el servicio de línea de telefonía fija residencial, no se garantiza la correcta funcionalidad para compañías que utilizan conmutador o PBX, debido a que existen diversos parámetros en estos tipos de servicios que el tasador no reconoce. Además en muchos casos las compañías contratan planes de servicios con tarifa fija y no por el consumo diario.

- Las operadores de telefonía fija, poseen un sistema de cobro (Charging system) muy bueno pero bastantes complejos, muchas veces los errores en la tasación es debido a problemas en los equipos o un error humano en la parametrización del sistema de cobro. Nuestro sistema es una herramienta externa para detectar dichos problemas en el lado del abonado A, pero se carece de información en la línea del abonado debido a que las centrales de telefonía no envían señales de retorno en dicha línea que indiquen que el abonado B ha contestado y con ello ha iniciado con éxito una comunicación telefónica. Por lo tanto existe un margen de error de al menos 5 segundos en

la duración del tiempo de llamada, puesto que solo se trabaja con las señales de audio presentes en la línea en ese momento.

- Con el desarrollo de este sistema se ha logrado facilitar la recopilación de datos al ente regulador, siendo mas eficientes en la resolución de casos abiertos por quejas que los abonados afectados presentan a diario en dicha institución. Pudiendo desarrollar más analizadores a futuro para atender diferentes zonas del país.

RECOMENDACIONES

- Las operadoras de telefonía fija de El Salvador, tienen dentro de sus propiedades de las centrales, la capacidad de activar el servicio de envío de información de respuesta de llamada al abonado A, por lo que se sugiere que a futuro se establezca una norma en la que obligue a los operadores a prestar dicho servicio, con esto se facilita el desarrollo de un sistema totalmente confiable y mas exacto. De esta manera se lograría hacer mucho más eficiente al ente regulador detectar anomalías en el cobro y aplicar la ley de telecomunicaciones.
- Se sugiere además seguir invirtiendo en el desarrollo de proyectos e investigaciones para obtener resultados que favorezcan al buen desarrollo del país dentro de los lineamientos correspondientes en el mundo de las telecomunicaciones.
- Reproducir el sistema desarrollado para dar mayor cobertura de servicio a los abonados afectados, y desarrollar otros nuevos productos puesto que el mundo de las telecomunicaciones es cambiante, y actualmente surgen nuevos planes y estrategias de mercado de los operadores que no constituyen el cobro de acuerdo al consumo, sino que son planes de cobro fijos con servicios ilimitados, pero no por eso dejarían de presentar anomalías en el cobro.

GLOSARIO

ACD: Automatic Call Distribution

ANI: Automatic Number Identification

ASP: Application Service Part

CAS: Señalización de Canal Asociado

CID: Caller Identification

CND: Called Number Delivery

CO: Central Office

CPE: Equipo Terminal del Abonado

CRC-16: Código de Detección de Errores

DID: Dialing Direct

DNIS: Dialed Number Identification Service

DTMF: Dual Tone Multifrecuency

DUP: Data User Part

E-1: Trama de transmisión digital de 1544 kbits en 24 time slots

EEPROM: Electrically-Erasable Programmable Read-Only Memory

FCS: Frame Check Sum

FGD: Grupo de Elemento D

FIB: Forward Indicator Bit

FSK: Frecuency Shift Keying

FSN: Forward Sequence Number

FXO: Foreign Exchange Office

FXS: Foreign Exchange Suscriber

GROUND START: Inicio de Tierra

I^2C: Inter-Integrated Circuit, bus de comunicación en serie

ISUP: Integrated Services Digital Network User Part

ITU-T: International Telecommunications Union Telecommunication Standardization Sector

IVR: Interactive Voice Response

LI: Length Indicator

LOOP-START: Inicio de Bucle

LSSU: Link Status Signal Unit

LSSU: Unidad de Señal de Estado del Enlace

MCLR: Master Clear

MDMF: Multiple Data Message Format

MF-C: Multifrecuencia Obligada

MF-P: Multifrecuency Pulse

MSU: Message Signal Unit

MSU: Unidades de Señal de Mensaje

MTP L1: Message Transfer Part Layer 1

MTP L2: Message Transfer Part Layer 2

MTP: Message Transfer Part

ON-HOOK/OFF-HOOK: Colgado/Descolgado

PBX: Private Branch Exchange

PIC: Peripheral Interface Controller

PIC: Peripheral Interface Controller

PPS: Pulsos por Segundo

PSS1: Private Signaling System

PSTN: Public Switched Telephone Network

RDSI: Red Digital de Servicios Integrados

RTC: Real Time Clock

SCCP: Signaling Connection and Control Part

SCL: Serial Clock Input

SCP: Service Control Point

SDA: Serial Data Input/Output

SDMF: Simple Data Message Format

SI: Service Indicator

SIB: Status Indicator Busy

SIE: Status Indicator Emergency

SIF: Service Information Field

SIN: Status Indicator Normal

SIO: Service Information Octect

SIO: Status Indicator Out of Alignment

SIOS: Status Indicator Out of Service

SIPO: Status Indicator Processor Outage

SLS: Signaling Link Selection

SQW/OUT: Square Wave/Output Driver

SS7: Sistema de Señalización #7

SSP: Service Switching Point

STP: Signal Transfer Point

T-1: Trama de transmisión digital de 2048 kbits en 32 time slots

TCAP: Transaction Capabilities Application Part

TCP/IP: Transmission Control Protocol/Internet Protocol

TUP: Telephone User Part

REFERENCIAS BIBLIOGRÁFICAS

[1] Superintendencia General de Electricidad y Telecomunicaciones, "Manual de indicadores I semestre 2009", 18 de agosto de 2009. Disponible en Internet en: http://www.siget.gob.sv/images/documentos/telecomunicaciones/estadisticas/manual_indicadores_2009_i_semestre0.pdf
Última visita 06042010

[2] M. Brain, "How Telephones Work", 31 de agosto de 2007. Disponible en Internet en:
http://communication.howstuffworks.com/telephone.htm
Última visita 06042010

[3] M. Juárez, "Glosario de términos de telecomunicaciones", cátedra de Mercado y Regulación de las Telecomunicaciones, Universidad Don Bosco, ciclo I-2005. No publicado.

[4] Anónimo, "Teléfono", 31 de agosto de 2007. Disponible en Internet en:
http://es.wikipedia.org/wiki/Tel%C3%A9fono
Última visita 20032010

[5] Anónimo, "Dual-tone multi-frecuency", 29 de agosto de 2007. Disponible en Internet en: http://en.wikipedia.org/wiki/DTMF
Última visita 20032010

[6] H. Intven, J. Oliver, E. Sepúlveda, "Manual de Reglamentación de las Telecomunicaciones", primera impresión, Washington D.C., Estados Unidos, noviembre de 2000, Módulo 6, Apéndice C. Disponible en Internet en:
http://www.infodev.org/en/Publication.132.html
Última visita 06042010

[7] V. Iniestra, "Telefonía", 31 de agosto de 2007. Disponible en Internet en:
http://www.geocities.com/v.iniestra/apuntes/telefonia/
Última visita 01022008

[8] S. Keagy, "Integración de redes de voz y datos", Pearson Education S.A., Madrid 2001.

[9] Anónimo, "Signaling System 7", 29 de agosto de 2007. Disponible en Internet en:
http://en.wikipedia.org/wiki/Signalling_System_7
Última visita 20032010

[10] Anónimo, "SS7 Protocol Suit", 1 de septiembre de 2007. Disponible en Internet en:
http://www.protocols.com/pbook/ss7.htm
Última visita 06042010

[11] Anónimo, "Microcontrolador", 31 de agosto de 2009. Disponible en Internet en
http://en.wikipedia.org/wiki/PIC_microcontroller
Última visita 06042010

[12] Microchip Technology Inc., "Hoja técnica del circuito integrado 24AA256, 2007"
http://ww1.microchip.com/downloads/en/DeviceDoc/21203P.pdf
Última visita 10112009

[13] Microchip Technology Inc., "Hoja técnica del microcontrolador PIC16F877A"
http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf
Última visita 12112009

[14] Sparkfun Electronics, "Hoja técnica del Real Time Clock"
http://www.sparkfun.com/datasheets/Components/DS1307.pdf
Última visita 10112009

[15] National Semiconductor, "Hoja técnica del circuito integrado LM567"
http://www.datasheetcatalog.com/datasheets_pdf/L/M/5/6/LM567.shtml
Última visita 12112009

[16] Texas Instruments, "Hoja técnica del circuito integrado MAX232"
http://www.datasheetcatalog.com/datasheets_pdf/M/A/X/2/MAX232.shtml
Última visita 12112009

[17] MITEL Semiconductor, "Hoja técnica del circuito integrado MT8870"
http://www.datasheetcatalog.com/datasheets_pdf/M/T/8/8/MT8870.shtml
Última visita 12112009

[18] Superintendencia General de Electricidad y Telecomunicaciones, "Plan de Numeración Nacional", 14 de mayo de 2005. Disponible en Internet en:
http://www.siget.gob.sv/images/documentos/telecomunicaciones/recursos_de_telecomunicaciones/plan_de_numeracion_14mayo05204.pdf
Última visita 05042010

[19] Unión Internacional de Telecomunicaciones, "Recomendaciones de la serie G". Disponibles en Internet en:
http://eu.sabotage.org/www/ITU/G/
Última visita 05042010

[20] Unión Internacional de Telecomunicaciones, "Recomendaciones de la serie Q". Disponibles en Internet en:
http://eu.sabotage.org/www/ITU/Q/
Última visita 05042010

[21] Symmetricom, "Soluciones de Sincronismo". Disponible en Internet en:
http://docs.google.com/viewer?a=v&q=cache:4BJ1PmlIU7EJ:poli.br/~pan/R%2520C%2520F%2520L/Apostilha%2520-%2520Solu%25E7oes%2520de%2520sincronismo.pdf+ss7+sincronismo&hl=es&gl=sv&pid=bl&srcid=ADGEESgXkDqp-GcNprKX9zrxsR-4WVVC9ru23A1G7a-4GxqeTTVENuWbg1Fq4M5cH3BZsTD0pHQZH0tuChBJtq2pknv_zyRqwyUTWVwbPt6MsYrcpyocbpLwP50QdowDashbEqOF1KRk&sig=AHIEtbRjJySvq13pcGenl0XyZ1SCCANIBw
Última visita 05042010

[22] Oscilloquartz, "Manual de Fuente de Referencia Primaria modelo 5585", 2008. No publicado.

[23] Superintendencia General de Electricidad y Telecomunicaciones, "Reglamento de la Ley de Telecomunicaciones", 5 de octubre de 2008. Disponible en internet en: http://www.siget.gob.sv/images/documentos/telecomunicaciones/legislacion/reglamento_de_la_ley_de_telecomunicaciones0.pdf
Última visita 05042010

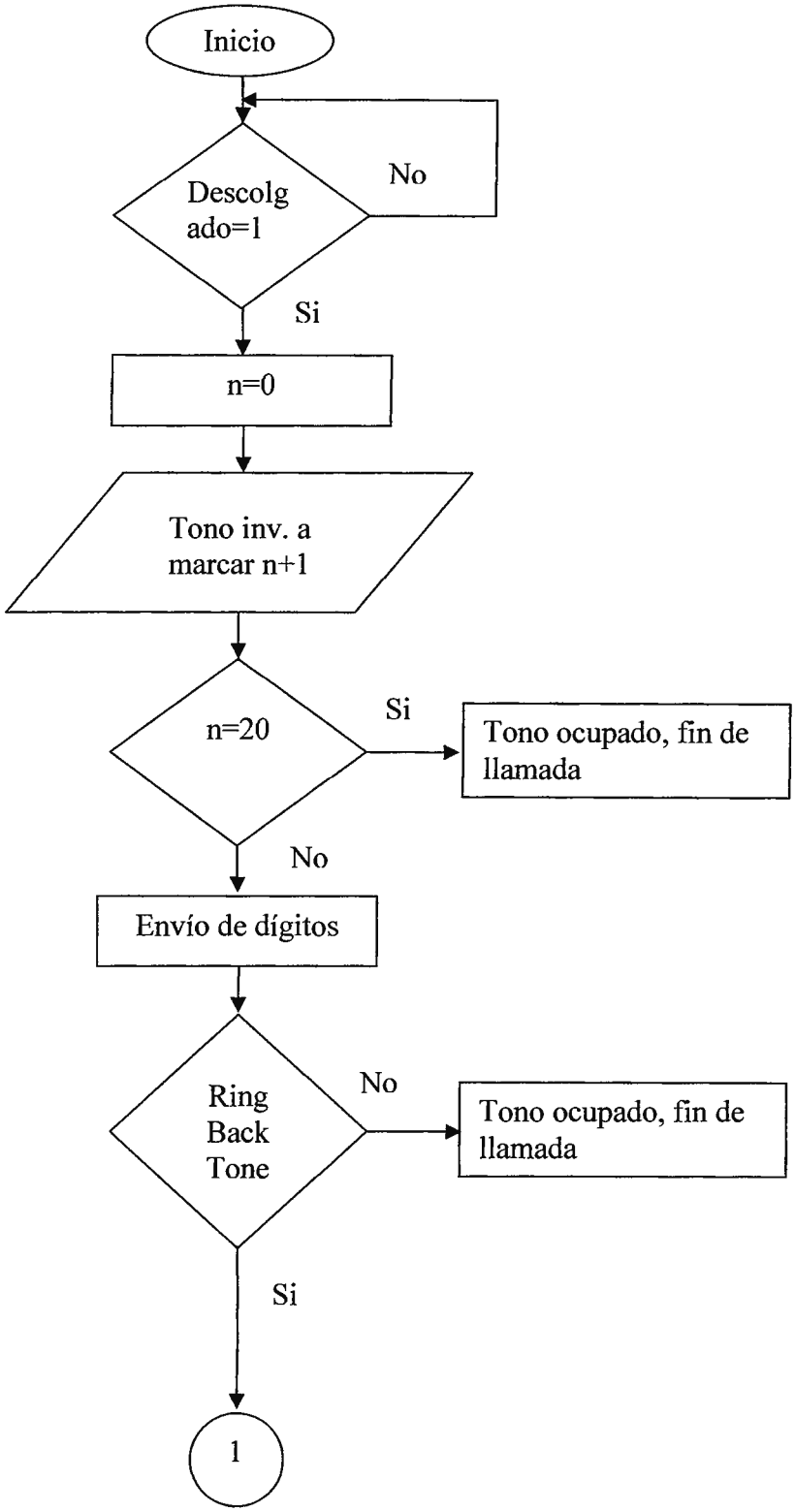
[24] Superintendencia General de Electricidad y Telecomunicaciones, "Tarifas aplicadas por CTE", 27 de julio de 2006. Disponible en internet en: http://www.siget.gob.sv/images/documentos/telecomunicaciones/tarifas/cte_iisem___20061555.pdf
Última visita 05042010

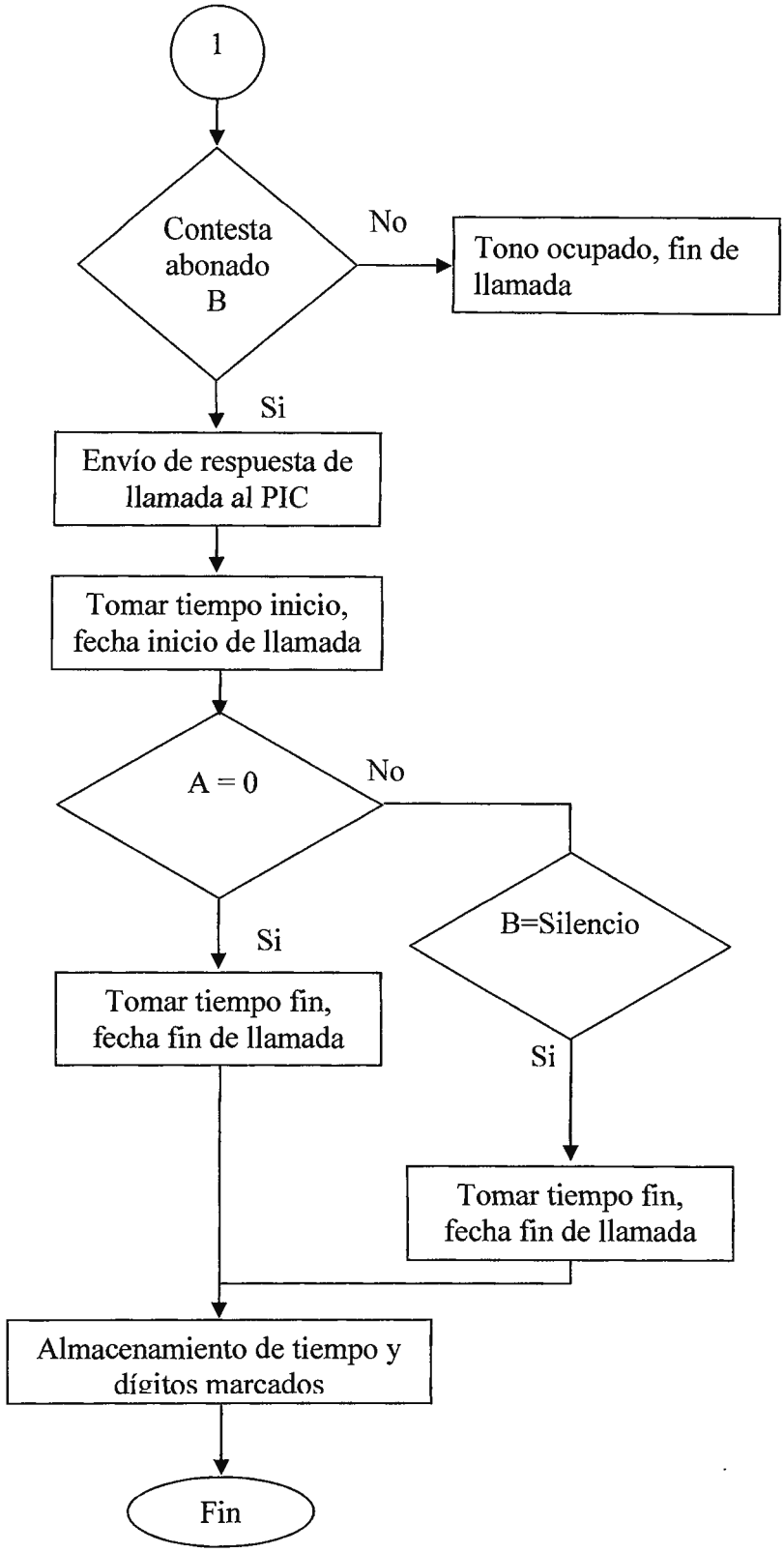
[25] Superintendencia General de Electricidad y Telecomunicaciones, "Lotes numéricos", 15 de septiembre de 2009. Disponible en internet en: http://www.siget.gob.sv/images/documentos/telecomunicaciones/recursos_de_telecomunicaciones/lotes_numericos_asignados_a_sep09090.pdf
Última visita 05042010

[26] El Diario de Hoy, 26 de enero de 2010, página 9.

ANEXOS

Anexo A: FLUJOGRAMA CONTROL DE TASACION




1
Contesta abonado B
No
Tono ocupado, fin de llamada
Si
Envío de respuesta de llamada al PIC
Tomar tiempo inicio, fecha inicio de llamada
A = 0
No
Si
B=Silencio
Si
Tomar tiempo fin, fecha fin de llamada
Tomar tiempo fin, fecha fin de llamada
Almacenamiento de tiempo y dígitos marcados
Fin

Anexo B: PROGRAMA DEL MICROCONTROLADOR

DATA LOGGER.c

```
#include "Data Logger.h"
#include "Comunicacion serie.h"
#include "Control Tasacion.h"

#use fast_io(a)
#use fast_io(b)
#use fast_io(c)
#use fast_io(d)
#use fast_io(e)

#define LED_POWER PIN_C0

void main()
{
   output_a(0x00);
   output_b(0x00);
   output_c(0x00);
   output_d(0x00);
   output_e(0x00);

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DIV_BY_4,249,5);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

   set_tris_a(0x00);       //Puerto no utilizado
   set_tris_b(0xC0);       //Puerto utilizado solamente para programacion
ICSP
   set_tris_c(0xF8);       //RC7 y RC6: RX y TX - USART (entradas)
                           //RC5: Deteccion de colgado (entrada)
                           //RC4 y RC3: SDA y SCL - I2C (entradas)
                           //RC2, RC1 y RC0: LEDs indicadores (salidas)
   set_tris_d(0x3F);       //RD7 y RD6: No utilizados
                           //RD5: Deteccion de tono de señalizacion
(entrada)
                           //RD4: Notificacion de tono numerico (entrada)
                           //RD3 a RD0: Codigo de tono numerico (entradas)
   set_tris_e(0x00);       //Puerto no utilizado

   enable_interrupts(INT_TIMER0);
   enable_interrupts(INT_TIMER2);
   enable_interrupts(GLOBAL);

   output_bit(LED_POWER, 1);     //Enciende el LED de indicador de encendido

   for (;;) {
      AtenderComunicacionSerie();
```

```
      AtenderTasacion();
   }
}
```

MANEJO DE RTC.c

```
#include "Data Logger.h"
#include "Almacenamiento de datos.h"

REGISTRO BufferRegistro;

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Error de transaccion I2C
//0x02 - Error de escritura
unsigned int8  LimpiarMemoria() {
   unsigned int8 i;
   i2c_start();

   //Inicia el ciclo de escritura a la memoria
   if (i2c_write(0xA0))
      //En caso de no haber reconocimiento, se regresa la respuesta de
error:
      return 0x01;

   //Envia la direccion interna a sobre escribir
   if (i2c_write(0x00)) return 0x01;
   if (i2c_write(0x00)) return 0x01;

   //Envia los datos a escribir en la localidad 0x0000 y 0x0001
   if (i2c_write(0x00)) return 0x01;
   if (i2c_write(0x00)) return 0x01;

   //Una vez hecho el borrado, se termina la transaccion
   i2c_stop();

   //Se inicia el proceso de espera a que la memoria termine su ciclo de
   //escritura (Se esperan 6ms como máximo)
   for (i=0; i<60; i++) {
      delay_us(100);   //Espera 0.1ms por iteracion

      i2c_start();   //Inicia una nueva transaccion

      //Verifica si se da un ACK iniciando una transaccion de escritura
      if (!i2c_write(0xA0)) {
         i2c_stop();    //Si se dio un ACK, termina la transferencia
         return 0x00;   //La funcion retorna con exito
      }

      //Si no se da un ACK, solo se termina la transaccion actual y se
intentara
      //nuevamente en otra iteracion
      i2c_stop();
   }
```

```
   //Si la transaccion tomo mas de lo esperado, se entiende que hubo un
error
   if (i >= 60) return 0x02;

   return 0x00;
}

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Numero de registro invalido
//0x02 - Error de transaccion I2C
unsigned int8 LeerRegistro(unsigned int16 NumReg) {
   BYTE_WORD Direccion;
   unsigned int8 i;

   //Valida la direccion recibida
   if (NumReg >= CapRegistros) return 0x01;

   //Determina la direccion exacta a acceder
   Direccion.Word = NumReg * sizeof(REGISTRO);

   //Inicia una nueva transaccion I2C
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x02;

   //Envia la direccion de la localidad a acceder
   if (i2c_write(Direccion.Byte1)) return 0x02;
   if (i2c_write(Direccion.Byte0)) return 0x02;

   //Inicia de nuevo la transaccion
   i2c_start();

   //Envia nuevamente la direccion de la memoria en el bus (transaccion
lectura)
   if (i2c_write(0xA1)) return 0x02;

   //Procede a leer la informacion del registro accedido
   for (i=0; i<sizeof(REGISTRO)-1; i++) {
      BufferRegistro.Datos[i] = i2c_read(1);  //Envia ACK en todos los bytes
menos el ultimo
   }
   BufferRegistro.Datos[i] = i2c_read(0);  //Para el ultimo byte, envia NAK

   //Tras haber leido el registro, se termina la transaccion
   i2c_stop();

   //Como todo salio bien, se termina con codigo de exito
   return 0x00;
}

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Error de transaccion I2C
//0x02 - Memoria llena
```

```
unsigned int8 GuardarRegistro() {
   unsigned int8 i;
   BYTE_WORD NumReg;
   BYTE_WORD DirMem;

   //Determina la posicion del siguiente registro a almacenar dependiendo de
la
   //capacidad utilizada
   i2c_start();         //Inicia la transaccion
   i2c_write(0xA0);     //Byte de control: transaccion de escritura
   i2c_write(0x00);     //Envia la direccion a acceder: 0x0000
   i2c_write(0x00);
   i2c_stop();          //Termina la escritura

   i2c_start();                  //Inicia otra transaccion
   i2c_write(0xA1);              //Byte de control: transaccion de lectura
   NumReg.Byte0 = i2c_read(1);   //Lee la localidad 0x0000
   NumReg.Byte1 = i2c_read(0);   //Lee la localidad 0x0001
   i2c_stop();                   //Fin de la transaccion

/*
   LeerRegistro(0);

   //Forma la capacidad utilizada tomando los 2 bytes
   NumReg.Byte0 = BufferRegistro.Datos[0];
   NumReg.Byte1 = BufferRegistro.Datos[1];
*/

   NumReg.Word++;    //Suma 1 para determinar la proxima localidad

   //Determina si todavia hay espacio en la memoria
   if (NumReg.Word >= CapRegistros) return 0x02;

   //Una vez determinado que hay capacidad, se calcula la direccion fisica
   //dentro de la EEPROM
   DirMem.Word = NumReg.Word * sizeof(REGISTRO);

   //Inicia una nueva transaccion I2C
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x01;

   //Envia la direccion de la localidad a acceder
   if (i2c_write(DirMem.Byte1)) return 0x01;
   if (i2c_write(DirMem.Byte0)) return 0x01;

   //Procede a enviar el contenido del registro
   for (i=0; i<sizeof(REGISTRO); i++) {
      if (i2c_write(BufferRegistro.Datos[i])) return 0x01;
   }

   //Tras haber escrito el registro, se termina la transaccion
   i2c_stop();

   //Dado que la memoria entra en un ciclo de escritura, se espera a que la
```

```
   //misma termine para poder escribir el resto de la informacion
   i = 0;
   while (!i) {
      i2c_start();                  //Genera condicion de inicio
      if (!i2c_write(0xA0)) i = 1;  //Envia byte de control, si hay ACK,
termino
      i2c_stop();                  //Genera condicion de parada
   }

   //Inicia una nueva transaccion I2C para atualizar el total de registros
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x01;

   //Se procedera a actualizar la localidad 0 (conteo de registros
almacenados)
   if (i2c_write(0)) return 0x01;
   if (i2c_write(0)) return 0x01;

   //Se envia el nuevo numero de registros almacenados
   if (i2c_write(NumReg.Byte0)) return 0x01;
   if (i2c_write(NumReg.Byte1)) return 0x01;

   //Tras haber actualizado el total de registros, termina la transaccion
i2c
   i2c_stop();

   return 0x00;      //Retorna con exito
}
```

ALMACENAMIENTO DE DATOS.c

```
#include "Data Logger.h"
#include "Almacenamiento de datos.h"

REGISTRO BufferRegistro;

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Error de transaccion I2C
//0x02 - Error de escritura
unsigned int8  LimpiarMemoria() {
   unsigned int8 i;
   i2c_start();

   //Inicia el ciclo de escritura a la memoria
   if (i2c_write(0xA0))
      //En caso de no haber reconocimiento, se regresa la respuesta de
error:
      return 0x01;

   //Envia la direccion interna a sobre escribir
   if (i2c_write(0x00)) return 0x01;
   if (i2c_write(0x00)) return 0x01;
```

```
   //Envia los datos a escribir en la localidad 0x0000 y 0x0001
   if (i2c_write(0x00)) return 0x01;
   if (i2c_write(0x00)) return 0x01;

   //Una vez hecho el borrado, se termina la transaccion
   i2c_stop();

   //Se inicia el proceso de espera a que la memoria termine su ciclo de
   //escritura (Se esperan 6ms como máximo)
   for (i=0; i<60; i++) {
      delay_us(100);   //Espera 0.1ms por iteracion

      i2c_start();   //Inicia una nueva transaccion

      //Verifica si se da un ACK iniciando una transaccion de escritura
      if (!i2c_write(0xA0)) {
         i2c_stop();    //Si se dio un ACK, termina la transferencia
         return 0x00;   //La funcion retorna con exito
      }

      //Si no se da un ACK, solo se termina la transaccion actual y se
intentara
      //nuevamente en otra iteracion
      i2c_stop();
   }

   //Si la transaccion tomo mas de lo esperado, se entiende que hubo un
error
   if (i >= 60) return 0x02;

   return 0x00;
}

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Numero de registro invalido
//0x02 - Error de transaccion I2C
unsigned int8 LeerRegistro(unsigned int16 NumReg) {
   BYTE_WORD Direccion;
   unsigned int8 i;

   //Valida la direccion recibida
   if (NumReg >= CapRegistros) return 0x01;

   //Determina la direccion exacta a acceder
   Direccion.Word = NumReg * sizeof(REGISTRO);

   //Inicia una nueva transaccion I2C
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x02;

   //Envia la direccion de la localidad a acceder
   if (i2c_write(Direccion.Byte1)) return 0x02;
   if (i2c_write(Direccion.Byte0)) return 0x02;
```

```
   //Envia los datos a escribir en la localidad 0x0000 y 0x0001
   if (i2c_write(0x00)) return 0x01;
   if (i2c_write(0x00)) return 0x01;

   //Una vez hecho el borrado, se termina la transaccion
   i2c_stop();

   //Se inicia el proceso de espera a que la memoria termine su ciclo de
   //escritura (Se esperan 6ms como máximo)
   for (i=0; i<60; i++) {
      delay_us(100);   //Espera 0.1ms por iteracion

      i2c_start();   //Inicia una nueva transaccion

      //Verifica si se da un ACK iniciando una transaccion de escritura
      if (!i2c_write(0xA0)) {
         i2c_stop();    //Si se dio un ACK, termina la transferencia
         return 0x00;   //La funcion retorna con exito
      }

      //Si no se da un ACK, solo se termina la transaccion actual y se
intentara
      //nuevamente en otra iteracion
      i2c_stop();
   }

   //Si la transaccion tomo mas de lo esperado, se entiende que hubo un
error
   if (i >= 60) return 0x02;

   return 0x00;
}

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Numero de registro invalido
//0x02 - Error de transaccion I2C
unsigned int8 LeerRegistro(unsigned int16 NumReg) {
   BYTE_WORD Direccion;
   unsigned int8 i;

   //Valida la direccion recibida
   if (NumReg >= CapRegistros) return 0x01;

   //Determina la direccion exacta a acceder
   Direccion.Word = NumReg * sizeof(REGISTRO);

   //Inicia una nueva transaccion I2C
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x02;

   //Envia la direccion de la localidad a acceder
   if (i2c_write(Direccion.Byte1)) return 0x02;
   if (i2c_write(Direccion.Byte0)) return 0x02;
```

```
   //Inicia de nuevo la transaccion
   i2c_start();

   //Envia nuevamente la direccion de la memoria en el bus (transaccion
lectura)
   if (i2c_write(0xA1)) return 0x02;

   //Procede a leer la informacion del registro accedido
   for (i=0; i<sizeof(REGISTRO)-1; i++) {
      BufferRegistro.Datos[i] = i2c_read(1);  //Envia ACK en todos los bytes
menos el ultimo
   }
   BufferRegistro.Datos[i] = i2c_read(0);  //Para el ultimo byte, envia NAK

   //Tras haber leido el registro, se termina la transaccion
   i2c_stop();

   //Como todo salio bien, se termina con codigo de exito
   return 0x00;
}

//Codigos de respuesta:
//0x00 - Exito
//0x01 - Error de transaccion I2C
//0x02 - Memoria llena
unsigned int8 GuardarRegistro() {
   unsigned int8 i;
   BYTE_WORD NumReg;
   BYTE_WORD DirMem;

   //Determina la posicion del siguiente registro a almacenar dependiendo de
la
   //capacidad utilizada
   i2c_start();         //Inicia la transaccion
   i2c_write(0xA0);     //Byte de control: transaccion de escritura
   i2c_write(0x00);     //Envia la direccion a acceder: 0x0000
   i2c_write(0x00);
   i2c_stop();          //Termina la escritura

   i2c_start();                  //Inicia otra transaccion
   i2c_write(0xA1);              //Byte de control: transaccion de lectura
   NumReg.Byte0 = i2c_read(1);   //Lee la localidad 0x0000
   NumReg.Byte1 = i2c_read(0);   //Lee la localidad 0x0001
   i2c_stop();                   //Fin de la transaccion

   NumReg.Word++;    //Suma 1 para determinar la proxima localidad

   //Determina si todavia hay espacio en la memoria
   if (NumReg.Word >= CapRegistros) return 0x02;

   //Una vez determinado que hay capacidad, se calcula la direccion fisica
   //dentro de la EEPROM
   DirMem.Word = NumReg.Word * sizeof(REGISTRO);

   //Inicia una nueva transaccion I2C
```

```
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x01;

   //Envia la direccion de la localidad a acceder
   if (i2c_write(DirMem.Byte1)) return 0x01;
   if (i2c_write(DirMem.Byte0)) return 0x01;

   //Procede a enviar el contenido del registro
   for (i=0; i<sizeof(REGISTRO); i++) {
      if (i2c_write(BufferRegistro.Datos[i])) return 0x01;
   }

   //Tras haber escrito el registro, se termina la transaccion
   i2c_stop();

   //Dado que la memoria entra en un ciclo de escritura, se espera a que la
   //misma termine para poder escribir el resto de la informacion
   i = 0;
   while (!i) {
      i2c_start();                  //Genera condicion de inicio
      if (!i2c_write(0xA0)) i = 1;  //Envia byte de control, si hay ACK,
termino
      i2c_stop();                  //Genera condicion de parada
   }

   //Inicia una nueva transaccion I2C para atualizar el total de registros
   i2c_start();

   //Envia la direccion de la memoria en el bus (trasaccion de escritura)
   if (i2c_write(0xA0)) return 0x01;

   //Se procedera a actualizar la localidad 0 (conteo de registros
almacenados)
   if (i2c_write(0)) return 0x01;
   if (i2c_write(0)) return 0x01;

   //Se envia el nuevo numero de registros almacenados
   if (i2c_write(NumReg.Byte0)) return 0x01;
   if (i2c_write(NumReg.Byte1)) return 0x01;

   //Tras haber actualizado el total de registros, termina la transaccion
i2c
   i2c_stop();

   return 0x00;      //Retorna con exito
}
```

COMUNICACIÓN SERIE.c

```
#include "Data Logger.h"
#include "Comunicacion serie.h"
#include "Manejo de RTC.h"
#include "Almacenamiento de datos.h"
```

```
#use fast_io(a)
#use fast_io(b)
#use fast_io(c)
#use fast_io(d)
#use fast_io(e)

#define LED_COMM PIN_C2

//Enumeracion de los estados posibles para la maquina de estados que
controla
//la comunicacion serie:
typedef enum _ESTADO_COMUNICACION {
   EC_HEADER = 0, EC_RX_DATOS, EC_CHECKSUM, EC_ERROR,
} ESTADO_COMUNICACION;

//Enumeracion de los comandos que se pueden interpretar:
typedef enum _ID_COMANDO_SERIE {
   IDCS_LEER_FECHA_HORA = 0, IDCS_ESTABLECER_FECHA_HORA, IDCS_LEER_REGISTRO,
   IDCS_BORRAR_REGISTROS,
   IDCS_ERROR_COMUNICACION = 0x80, IDCS_ERROR_RTC, IDCS_NUM_REG_INVALIDO,
   IDCS_ERROR_EEPROM,
} ID_COMANDO_SERIE;

//Estructura que alberga la informacion de la comunicacion actual (datos
//recibidos por el puerto serie):
typedef struct _INFO_COMUNICACION {
   ID_COMANDO_SERIE Cabecera;
   unsigned int8 Datos[32];
   unsigned int8 Checksum;
} INFO_COMUNICACION;

//Declaracion de las variables usadas en el modulo:
INFO_COMUNICACION InfoCom;                    //Datos recibidos
ESTADO_COMUNICACION EstadoCom = EC_HEADER;    //Estado de la comunicacion
unsigned int8 nBytesEsperados;                //Numero de bytes a recibir
unsigned int8 IndiceDatos;                    //Indice al siguiente dato

void EjecutarComando();
void EnviarTramaErrorComunicacion();
void EnviarTramaErrorRTC();
void EnviarTramaNumRegInvalido();
void EnviarTramaErrorEEPROM();

//Rutina de interrupcion del Timer0
//--------------------------------
//El modulo Timer0 es utilizado en esta aplicacion para indicar que ha
ocurrido
//un tiempo prolongado sin recibirse datos a traves del puerto serie.
Siempre
//que el contador se desborda, se genera la interrupcion y acto seguido la
ISR
//modifica el estado de la comunicacion para que se inicie una transacción
//nueva, abortando la transacción anterior.
#int_TIMER0
void  TIMER0_isr(void)
{
```

```
   EstadoCom = EC_HEADER;      //Lo siguiente que se recibira sera un header
   output_bit(LED_COMM, 0);    //Apaga el LED de comunicacion
}

void AtenderComunicacionSerie() {
   unsigned int8 i, Checksum;

   //Si no hay caracteres en el buffer, simplemente retorna
   if (!kbhit()) return;

   //Si existen caracteres, entonces se deshabilitan las interrupciones, se
   //limpia el Timer0 y se limpia su bandera de interrupción:
   disable_interrupts(INT_TIMER0);
   set_timer0(0x00);
   clear_interrupt(INT_TIMER0);

   //Asimismo se procede a encender el led de comunicacion
   output_bit(LED_COMM, 1);

   //Se determina la siguiente accion de acuerdo al estado de la
comunicacion
   switch (EstadoCom) {
   case EC_HEADER:
      //Se recibe el header
      InfoCom.Cabecera = getc();

      //Luego segun el header, se determina el siguiente estado y numero de
      //bytes a recibir
      switch (InfoCom.Cabecera) {
      case IDCS_LEER_FECHA_HORA:       //En caso que se desee leer la
fecha/hora
         EstadoCom = EC_CHECKSUM;      //solo se espera el checksum
         nBytesEsperados = 0;
         break;
      case IDCS_ESTABLECER_FECHA_HORA: //En caso que se desee establecer la
         EstadoCom = EC_RX_DATOS;      //fecha/hora, se esperaran 6 bytes
         nBytesEsperados = 6;
         IndiceDatos = 0;
         break;
      case IDCS_LEER_REGISTRO:         //En caso que se desee acceder un
         EstadoCom = EC_RX_DATOS;      //registro, se esperan los 2 bytes
con
         nBytesEsperados = 2;          //la direccion del mismo
         IndiceDatos = 0;
         break;
      case IDCS_BORRAR_REGISTROS:      //En caso que se deseen limpiar los
         EstadoCom = EC_CHECKSUM;      //registros solo se espera el
checksum
         nBytesEsperados = 0;
         break;
      default:
         EstadoCom = EC_ERROR;
         EnviarTramaErrorComunicacion();
         break;
      }
      break;
```

```
   case EC_RX_DATOS:
      //Se guarda el dato recien recibido (apunta de una vez al siguiente):
      InfoCom.Datos[IndiceDatos++] = getc();

      //Se verifica si ya se recibieron todos los datos:
      if (IndiceDatos >= nBytesEsperados)
         EstadoCom = EC_CHECKSUM;    //De recibirse todo, se esperara el
checksum
      break;
   case EC_CHECKSUM:
      //Se recibe el dato de checksum
      InfoCom.Checksum = getc();

      //Se calcula localmente el checksum en base a la informacion recibida
      Checksum = InfoCom.Cabecera;
      for (i=0; i<nBytesEsperados; i++)
         Checksum += InfoCom.Datos[i];
      Checksum = ~Checksum;

      //Se corrobora el checksum
      if (Checksum != InfoCom.Checksum) {
         EstadoCom = EC_ERROR;
         EnviarTramaErrorComunicacion();
         break;
      }

      //Si la trama se recibio correctamente, se ejecuta el comando que
contiene
      EjecutarComando();

      //Tras terminarse el comando, se puede recibir uno nuevo de inmediato:
      EstadoCom = EC_HEADER;

      break;
   case EC_ERROR:
      //Aqui simplemente no se hace nada (el error se limpia solo al darse
el
      //sobreflujo de Timer0)
      getc();  //Descarta el caracter recibido para limpiar el buffer
      break;
   }

   //Al final del proceso se vuelven a activar las interrupciones
   enable_interrupts(INT_TIMER0);
}

void EjecutarComando() {
   unsigned int8 Checksum;
   BYTE_WORD NumReg;
   unsigned int8 i;

   switch (InfoCom.Cabecera) {
   case IDCS_LEER_FECHA_HORA:
      //Para este comando, se obtiene la fecha/hora del RTC
      if (ObtenerFechaHora()) {
         EnviarTramaErrorRTC();
```

```
         break;
      }

      //A continuacion se envia la informacion de la fecha/hora a la vez que
se
      //calcula el checksum para luego enviarlo
      putc(IDCS_LEER_FECHA_HORA);      Checksum = IDCS_LEER_FECHA_HORA;
      putc(BufferFechaHora.Seconds);   CheckSum += BufferFechaHora.Seconds;
      putc(BufferFechaHora.Minutes);   CheckSum += BufferFechaHora.Minutes;
      putc(BufferFechaHora.Hours);     CheckSum += BufferFechaHora.Hours;
      putc(BufferFechaHora.Date);      CheckSum += BufferFechaHora.Date;
      putc(BufferFechaHora.Month);     CheckSum += BufferFechaHora.Month;
      putc(BufferFechaHora.Year);      CheckSum += BufferFechaHora.Year;
      putc(~Checksum);

      break;
   case IDCS_ESTABLECER_FECHA_HORA:
      //Guarda la informacion recibida en el buffer de fecha/hora
      BufferFechaHora.Seconds = InfoCom.Datos[0];
      BufferFechaHora.Minutes = InfoCom.Datos[1];
      BufferFechaHora.Hours = InfoCom.Datos[2];
      BufferFechaHora.Date = InfoCom.Datos[3];
      BufferFechaHora.Month = InfoCom.Datos[4];
      BufferFechaHora.Year = InfoCom.Datos[5];

      //Procede a establecer la fecha y hora recibidos
      if (EstablecerFechaHora()) {
         EnviarTramaErrorRTC();  //Si hubo un error, lo reporta acordemente
      }
      else {
         putc(IDCS_ESTABLECER_FECHA_HORA);   //Si hubo exito, Envia la
         putc(~IDCS_ESTABLECER_FECHA_HORA);  //cofirmacion
      }

      break;
   case IDCS_LEER_REGISTRO:
      //Se obtiene el numero de registro solicitado
      NumReg.Byte0 = InfoCom.Datos[0];
      NumReg.Byte1 = InfoCom.Datos[1];

      //Se procede a leer el registro solicitado y se actua acordemente
segun el
      //resultado
      switch (LeerRegistro(NumReg.Word)) {
      case 0:
         //Envia el ID de la respuesta exitosa e inicializa el checksum
         putc(IDCS_LEER_REGISTRO);          Checksum = IDCS_LEER_REGISTRO;

         //Se procede a enviar el contenido del registro a medida que se va
         //calculando el checksum
         for (i=0; i<sizeof(REGISTRO); i++) {
            putc(BufferRegistro.Datos[i]);  Checksum +=
BufferRegistro.Datos[i];
         }

         //Se termina de calcular el checksum y se envia
```

```
            putc(~Checksum);

            break;
         case 1:
            EnviarTramaNumRegInvalido();  //Error de registro invalido
            break;
         case 2:
            EnviarTramaErrorEEPROM();     //Error de transaccion I2C con EEPROM
            break;
         }
         break;
      case IDCS_BORRAR_REGISTROS:
         switch(LimpiarMemoria()) {
         case 0:
            putc(IDCS_BORRAR_REGISTROS);
            putc(~IDCS_BORRAR_REGISTROS);
            break;
         case 1:
         case 2:
            EnviarTramaErrorEEPROM();
            break;
         }
         break;
      }
}

void EnviarTramaErrorComunicacion() {
   putc(IDCS_ERROR_COMUNICACION);
   putc(~IDCS_ERROR_COMUNICACION);
}

void EnviarTramaErrorRTC() {
   putc(IDCS_ERROR_RTC);
   putc(~IDCS_ERROR_RTC);
}

void EnviarTramaNumRegInvalido() {
   putc(IDCS_NUM_REG_INVALIDO);
   putc(~IDCS_NUM_REG_INVALIDO);
}

void EnviarTramaErrorEEPROM() {
   putc(IDCS_ERROR_EEPROM);
   putc(~IDCS_ERROR_EEPROM);
}
```

CONTROL_TASACION.c

```
#include "Data Logger.h"
#include "Control Tasacion.h"

#include "Almacenamiento de datos.h"
#include "Manejo de RTC.h"

#use fast_io(a)
#use fast_io(b)
```

```
#use fast_io(c)
#use fast_io(d)
#use fast_io(e)

#define LED_TASACION PIN_C1

#define NOTIF_COLGADO PIN_C5
#define TONO_SENALIZACION PIN_D5
#define NOTIF_TONO_NUM PIN_D4

typedef enum _ESTADO_TASACION {
   ET_ESPERAR_COLGADO = 0, ET_COLGADO, ET_DESCOLGADO,
ET_TONO_INIVITACION_MARCAR,
   ET_MARCACION_DIGITOS, ET_TONO_LLAMADA, ET_LLAMADA_EN_PROGRESO,
   ET_FIN_LLAMADA,
} ESTADO_TASACION;

ESTADO_TASACION EstadoTasacion = ET_ESPERAR_COLGADO;
int1 bAtenderTasacion = 0;
int1 bEstadoPinTonoNum = 0;         //Registran el estado actual y el
anterior
int1 bUltimoEstadoPinTonoNum = 0;   //del pin de notificacion de tono
numerico
unsigned int8 nDigitosMarcados = 0;

//Variables de control de tiempo para el tono de señalizacion
unsigned int16 tPresenciaTonoSenalizacion = 0;
unsigned int16 tAusenciaTonoSenalizacion = 0;
unsigned int16 tUltimaPresenciaTono = 0;
unsigned int16 tUltimaAusenciaTono = 0;
const unsigned int16 UmbralPresenciaTono = 200;
const unsigned int16 UmbralAusenciaTono = 200;

//Variables de control de tiempo para señalizacion de LED
unsigned int16 tPulsacionLEDTasacion = 0;

//Tiempos estandar
// - Invitacion a marcar: 20 segundos maximo.
// - Tonos de ocupado o congestion: 0.5 on / 0.5 off durante 10 segundos
maximo
// - Tono de ringback: 1 on / 4 off hasta un maximo de 10 timbrazos (a veces
el
//   primer off dura menos de 1 seg).

//Declaracion previa de las funciones
//-----------------------------------
void RegistrarTonoSenalizacion();
void ActualizarLEDTasacion();

//Rutina de interrupcion del Timer2
//---------------------------------
#int_TIMER2
void TIMER2_isr(void)
{
   //La banera de atencion de tasacion se activa periodicamente a un ritmo
de
```

```
   //1KHz
   bAtenderTasacion = 1;
}

void AtenderTasacion() {
   //Se verifica si ya es momento de atender la tasacion
   if (!bAtenderTasacion) return;

   //Se procedera a atender, asi que se apaga la bandera de inmediato
   bAtenderTasacion = 0;

   //output_bit(LED_POWER, input(TONO_SENALIZACION));
   //output_bit(LED_POWER, input(NOTIF_COLGADO));
   //output_bit(LED_TASACION, input(TONO_SENALIZACION));
   //output_bit(LED_COMM, input(NOTIF_TONO_NUM));

   //Antes de entrar al proceso de la secuencia de llamada se monitorea la
   //entrada de tono de señalizacion para analizar la presencia o ausencia
del
   //tono asociado
   RegistrarTonoSenalizacion();

   //Se registra el estado del pin de notificacion de tono numerico:
   bUltimoEstadoPinTonoNum = bEstadoPinTonoNum; //Guarda el estado anterior
   bEstadoPinTonoNum = input(NOTIF_TONO_NUM);

   //Se procede a procesar la secuencia de la llamada
   switch (EstadoTasacion) {
   case ET_ESPERAR_COLGADO:
      //Verifica que el telefono este colgado
      if (!input(NOTIF_COLGADO)) EstadoTasacion = ET_COLGADO;
      break;
   case ET_COLGADO:
      //Verifica si recien se acaba de descolgar el telefono
      if (input(NOTIF_COLGADO)) EstadoTasacion = ET_DESCOLGADO;
      break;
   case ET_DESCOLGADO:
      //Vuelve al principio en caso de volver a colgar (haya o no tono de
      //llamada)
      if (!input(NOTIF_COLGADO)) EstadoTasacion = ET_COLGADO;

      //Avanza al siguiente estado en caso de detectarse tono de llamada
      if (tPresenciaTonoSenalizacion >= UmbralPresenciaTono)
         EstadoTasacion = ET_TONO_INIVITACION_MARCAR;

      //Nota: si no se obtiene nunca un tono de llamada, el sistema se queda
      //atrapado indefinidamente en este estado hasta que se cuelgue (caso
de
      //linea muerta)

      break;
   case ET_TONO_INIVITACION_MARCAR:
      //Caso de telefono colgado
      if (!input(NOTIF_COLGADO)) EstadoTasacion = ET_COLGADO;

      //Caso de primer digito marcado
```

```
      if (!bUltimoEstadoPinTonoNum && bEstadoPinTonoNum) {
         //Se registra el primer digito
         BufferRegistro.NumTel[0] = input_d() & 0x0F;
         nDigitosMarcados = 1;
         EstadoTasacion = ET_MARCACION_DIGITOS;

         tAusenciaTonoSenalizacion = 0;
         tPresenciaTonoSenalizacion = 0;
         tUltimaPresenciaTono = 0;
         tUltimaAusenciaTono = 0;
      }

      //Nota: En caso que no se marquen digitos en ningun momento hasta este
      //punto, no se considera que se ha llamado a ningun numero. La linea
      //presentara un tono de ocupado automaticamente, pero no hace falta
      //registrar el mismo si no se ha hecho llamada alguna. El programa
volvera
      //a estado de espera inicial en el momento que el usuario cuelgue.

      break;
   case ET_MARCACION_DIGITOS:
      //Caso de telefono colgado
      if (!input(NOTIF_COLGADO)) EstadoTasacion = ET_COLGADO;

      //Caso de digito marcado
      if (!bUltimoEstadoPinTonoNum && bEstadoPinTonoNum) {
         //Guarda el nuevo digito
         BufferRegistro.NumTel[nDigitosMarcados] = input_d() & 0x0F;
         nDigitosMarcados++;  //Incrementa la cuenta de digitos almacenados
      }

      //Caso de tono de llamada (inicio solamente)
      if (tPresenciaTonoSenalizacion > 300)
         EstadoTasacion = ET_TONO_LLAMADA;

      //Caso de tono de ocupado
      if (tUltimaPresenciaTono > 0 && tUltimaPresenciaTono <= 600 &&
          tUltimaAusenciaTono  > 0 && tUltimaAusenciaTono  <= 600)
         EstadoTasacion = ET_ESPERAR_COLGADO;

      break;
   case ET_TONO_LLAMADA:
      //Caso de telefono colgado
      if (!input(NOTIF_COLGADO)) EstadoTasacion = ET_COLGADO;

      //Caso de tono de ocupado
      if (tUltimaPresenciaTono > 0 && tUltimaPresenciaTono <= 600 &&
          tUltimaAusenciaTono  > 0 && tUltimaAusenciaTono  <= 600)
         EstadoTasacion = ET_ESPERAR_COLGADO;

      //Caso de fin de tono de llamada (llamada contestada)
      if (tAusenciaTonoSenalizacion > 5000) {
         ObtenerFechaHora();
         BufferRegistro.FH_Inicio.Seconds = BufferFechaHora.Seconds;
         BufferRegistro.FH_Inicio.Minutes = BufferFechaHora.Minutes;
         BufferRegistro.FH_Inicio.Hours = BufferFechaHora.Hours;
```

```
            BufferRegistro.FH_Inicio.Date = BufferFechaHora.Date;
            BufferRegistro.FH_Inicio.Month = BufferFechaHora.Month;
            BufferRegistro.FH_Inicio.Year = BufferFechaHora.Year;

            EstadoTasacion = ET_LLAMADA_EN_PROGRESO;
         }
         break;
      case ET_LLAMADA_EN_PROGRESO:
         //Caso de telefono colgado
         if (!input(NOTIF_COLGADO)) EstadoTasacion = ET_FIN_LLAMADA;

         //Caso de tono de fin de llamada
         //if (tUltimaPresenciaTono > 0 && tUltimaPresenciaTono <= 600 &&
             //tUltimaAusenciaTono  > 0 && tUltimaAusenciaTono  <= 600)
            //EstadoTasacion = ET_FIN_LLAMADA;
         break;
      case ET_FIN_LLAMADA:
         //Marca el final del numero telefonico con el dato 0x10
         BufferRegistro.NumTel[nDigitosMarcados] = 0x10;

         //Obtiene la fecha y hora del final de la llamada
         ObtenerFechaHora();
         BufferRegistro.FH_Final.Seconds = BufferFechaHora.Seconds;
         BufferRegistro.FH_Final.Minutes = BufferFechaHora.Minutes;
         BufferRegistro.FH_Final.Hours = BufferFechaHora.Hours;
         BufferRegistro.FH_Final.Date = BufferFechaHora.Date;
         BufferRegistro.FH_Final.Month = BufferFechaHora.Month;
         BufferRegistro.FH_Final.Year = BufferFechaHora.Year;

         //Tras tener la informacion completa en el buffer de registro, se
guarda
         //el mismo en la memoria EEPROM
         GuardarRegistro();

         //Tras guardar la llamada, vuelve a esperar una nueva llamada
         EstadoTasacion = ET_ESPERAR_COLGADO;
         break;
      }

      //Tras actualizar el estado del sistema, se actualiza el LED de tasacion
      ActualizarLEDTasacion();
}

void RegistrarTonoSenalizacion() {
      if (input(TONO_SENALIZACION)) {
         tPresenciaTonoSenalizacion++;
         if (tPresenciaTonoSenalizacion == UmbralPresenciaTono) {
            tUltimaAusenciaTono = tAusenciaTonoSenalizacion;
            tAusenciaTonoSenalizacion = 0;
         }
      }
      else {
         tAusenciaTonoSenalizacion++;
         if (tAusenciaTonoSenalizacion == UmbralAusenciaTono) {
            tUltimaPresenciaTono = tPresenciaTonoSenalizacion;
            tPresenciaTonoSenalizacion = 0;
```

```
      }
   }
}

void ActualizarLEDTasacion() {
   switch (EstadoTasacion) {
   case ET_ESPERAR_COLGADO:
   case ET_COLGADO:
      //Estando en espera de colgado o colgado el LED permanece apagado
      tPulsacionLEDTasacion = 0;
      output_bit(LED_TASACION, 0);
      break;
   case ET_DESCOLGADO:
   case ET_TONO_INIVITACION_MARCAR:
   case ET_MARCACION_DIGITOS:
   case ET_TONO_LLAMADA:
      //Estando en proceso de iniciar una llamada el LED parpadea
      tPulsacionLEDTasacion++;
      if (tPulsacionLEDTasacion >= 1000) tPulsacionLEDTasacion = 0;

      if (tPulsacionLEDTasacion < 500)
         output_bit(LED_TASACION, 1);
      else
         output_bit(LED_TASACION, 0);

      break;
   case ET_LLAMADA_EN_PROGRESO:
   case ET_FIN_LLAMADA:
      //Durante la llamada y justo cuando termina, el LED permanece
encendido
      output_bit(LED_TASACION, 1);
      break;
   }
}
```

DATA LOGGER.h

```
#include "Data Logger.h"
#include "Comunicacion serie.h"
#include "Control Tasacion.h"

#use fast_io(a)
#use fast_io(b)
#use fast_io(c)
#use fast_io(d)
#use fast_io(e)

#define LED_POWER PIN_C0

void main()
{
   output_a(0x00);
   output_b(0x00);
   output_c(0x00);
   output_d(0x00);
   output_e(0x00);
```

```
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DIV_BY_4,249,5);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

   set_tris_a(0x00);       //Puerto no utilizado
   set_tris_b(0xC0);       //Puerto utilizado solamente para programacion
ICSP
   set_tris_c(0xF8);       //RC7 y RC6: RX y TX - USART (entradas)
                           //RC5: Deteccion de colgado (entrada)
                           //RC4 y RC3: SDA y SCL - I2C (entradas)
                           //RC2, RC1 y RC0: LEDs indicadores (salidas)
   set_tris_d(0x3F);       //RD7 y RD6: No utilizados
                           //RD5: Deteccion de tono de señalizacion
(entrada)
                           //RD4: Notificacion de tono numerico (entrada)
                           //RD3 a RD0: Codigo de tono numerico (entradas)
   set_tris_e(0x00);       //Puerto no utilizado

   enable_interrupts(INT_TIMER0);
   enable_interrupts(INT_TIMER2);
   enable_interrupts(GLOBAL);

   output_bit(LED_POWER, 1);     //Enciende el LED de indicador de encendido

   for (;;) {
      AtenderComunicacionSerie();
      AtenderTasacion();
   }
}
```

MANEJO DE RTC.h

```
#ifndef Manejo_de_RTC_h_Incluida
#define Manejo_de_RTC_h_Incluida

typedef struct _FECHA_HORA {
   unsigned int8 Seconds;
   unsigned int8 Minutes;
   unsigned int8 Hours;
   unsigned int8 Date;
   unsigned int8 Month;
   unsigned int8 Year;
} FECHA_HORA;

extern FECHA_HORA BufferFechaHora;

unsigned int8 ObtenerFechaHora();
unsigned int8 EstablecerFechaHora();

#endif //Manejo_de_RTC_h_Incluida
```

COMUNICACIÓN SERIE.h

```
#ifndef Comunicacion_serie_h_Incluida
#define Comunicacion_serie_h_Incluida

void AtenderComunicacionSerie();

#endif //Comunicacion_serie_h_Incluida
```

CONTROL TASACION.h

```
#ifndef Comunicacion_serie_h_Incluida
#define Comunicacion_serie_h_Incluida

void AtenderComunicacionSerie();

#endif //Comunicacion_serie_h_Incluida
```

Anexo C: PROGRAMA DE LA PC

COMUNICACION SERIE.cpp

```
//Inclusion de las librerias estandar
#include <windows.h>
#include <stdio.h>

//Inclusion de la cabecera propia
#include "Comunicacion serie.h"

//Inclusion de las cabeceras de los otros modulos
#include "Principal.h"

//Declaracion de los tipos de datos usados localmente
typedef enum _ID_COMANDO_SERIE {
   IDCS_LEER_FECHA_HORA = 0, IDCS_ESTABLECER_FECHA_HORA, IDCS_LEER_REGISTRO,
   IDCS_BORRAR_REGISTROS,
} ID_COMANDO_SERIE;

typedef struct _TRAMA_SERIE {
   ID_COMANDO_SERIE ID;
   unsigned char Datos[32];
   unsigned long nDatos;
   unsigned char Checksum;
} TRAMA_SERIE;

//Declaracion previa de las variables globales
FECHA_HORA BufferFechaHora;
REGISTRO BufferRegistro;

//Declaracion previa de las variables locales
static TRAMA_SERIE TramaSerie;

//Declaracion previa de las funciones locales
```

```
static bool EnviarComandoTasador();
static bool RecibirRespuestaTasador();

//+------------------------------+
//| Inicio del codigo del modulo |
//+------------------------------+----------------------------------------
----------------------
bool LeerFechaHora() {
   //Se determina el comando a enviar asi como el numero de bytes
   TramaSerie.ID = IDCS_LEER_FECHA_HORA;
   TramaSerie.nDatos = 0;

   //Se procede a enviar el comando
   if (!EnviarComandoTasador()) return false;

   //Espera a que los datos se terminen de enviar y a que haya una respuesta
   Sleep(50);

   //Una vez enviado, se recibe respuesta
   if (!RecibirRespuestaTasador()) return false;

   //Se verifica que la respuesta del tasador sea la correcta:
   if (TramaSerie.ID != IDCS_LEER_FECHA_HORA) return false;

   //Finalmente, se convierte el contenido en BCD de la respuesta a binario
   BufferFechaHora.Seconds = (TramaSerie.Datos[0] >> 4) * 10 +
(TramaSerie.Datos[0] & 0x0F);
   BufferFechaHora.Minutes = (TramaSerie.Datos[1] >> 4) * 10 +
(TramaSerie.Datos[1] & 0x0F);
   BufferFechaHora.Hours =   (TramaSerie.Datos[2] >> 4) * 10 +
(TramaSerie.Datos[2] & 0x0F);
   BufferFechaHora.Date =    (TramaSerie.Datos[3] >> 4) * 10 +
(TramaSerie.Datos[3] & 0x0F);
   BufferFechaHora.Month =   (TramaSerie.Datos[4] >> 4) * 10 +
(TramaSerie.Datos[4] & 0x0F);
   BufferFechaHora.Year =    (TramaSerie.Datos[5] >> 4) * 10 +
(TramaSerie.Datos[5] & 0x0F);

   return true;
}

bool EstablecerFechaHora() {
   //Se determina el comando a enviar, el contenido de la trama y el numero
de bytes
   TramaSerie.ID = IDCS_ESTABLECER_FECHA_HORA;
   //Para los datos, se hace una conversion de binario a BCD enmascarando
los digitos no usados
   //respectivamente
   TramaSerie.Datos[0] = (((BufferFechaHora.Seconds / 10) % 10) << 4) & 0x70
| (BufferFechaHora.Seconds % 10);
   TramaSerie.Datos[1] = (((BufferFechaHora.Minutes / 10) % 10) << 4) & 0x70
| (BufferFechaHora.Minutes % 10);
   TramaSerie.Datos[2] = (((BufferFechaHora.Hours   / 10) % 10) << 4) & 0x30
| (BufferFechaHora.Hours % 10);
   TramaSerie.Datos[3] = (((BufferFechaHora.Date    / 10) % 10) << 4) & 0x30
| (BufferFechaHora.Date % 10);
```

```
   TramaSerie.Datos[4] = (((BufferFechaHora.Month   / 10) % 10) << 4) & 0x10
| (BufferFechaHora.Month % 10);
   TramaSerie.Datos[5] = (((BufferFechaHora.Year    / 10) % 10) << 4)
| (BufferFechaHora.Year % 10);
   TramaSerie.nDatos = 6;

   //Se envia el comando
   if (!EnviarComandoTasador()) return false;

   //Se recibe la respuesta
   if (!RecibirRespuestaTasador()) return false;

   //Verifica que la respuesta sea la correcta
   if (TramaSerie.ID != IDCS_ESTABLECER_FECHA_HORA) return false;

   //Si todo salio bien, se retorna con verdadero
   return true;
}

bool LeerRegistroTasador(unsigned short NumReg) {
   unsigned long i;

   //Se prepara el comando a enviar, incluyendo el ID y el numero de
registro a acceder
   TramaSerie.ID = IDCS_LEER_REGISTRO;
   TramaSerie.Datos[0] = NumReg & 0xFF;
   TramaSerie.Datos[1] = (NumReg >> 8) & 0xFF;
   TramaSerie.nDatos = 2;

   //Se envia el comando
   if (!EnviarComandoTasador()) return false;

   //Espera a que los datos se terminen de enviar y a que haya una respuesta
   Sleep(50);

   //Se recibe la respuesta
   if (!RecibirRespuestaTasador()) return false;

   //Verifica que la respuesta sea la correcta
   if (TramaSerie.ID != IDCS_LEER_REGISTRO) return false;

   //De ser asi, se copia toda la informacion recibida al buffer del
registro
   for (i=0; i<sizeof(REGISTRO); i++) {
      BufferRegistro.Datos[i] = TramaSerie.Datos[i];
   }

   //Si todo salio bien, se retorna con verdadero
   return true;
}

bool ObtenerTotalRegistrosTasador(unsigned short *TotalRegistros) {
   //Primeramente se lee el registro 0 del tasador que contiene el total de
registros almacenados
   if (!LeerRegistroTasador(0)) return false;
```

```
   //Devuelve el total de registros recien obtenidos en la variable por
referencia
   *TotalRegistros = TramaSerie.Datos[0] + (TramaSerie.Datos[1] << 8);

   //Todo ha salido bien, se retorna verdadero
   return true;
}

bool BorrarRegistrosTasador() {
   //Se prepara el comando a enviar (solo el ID)
   TramaSerie.ID = IDCS_BORRAR_REGISTROS;
   TramaSerie.nDatos = 0;

   //Se envia el comando
   if (!EnviarComandoTasador()) return false;

   //Se recibe la respuesta
   if (!RecibirRespuestaTasador()) return false;

   //Se verifica que la respuesta sea la correcta
   if (TramaSerie.ID != IDCS_BORRAR_REGISTROS) return false;

   //Como todo salio bien, se retorna con verdadero
   return true;
}

static bool EnviarComandoTasador() {
   unsigned long BytesEscritos;
   unsigned long i;

   //Se procede a limpiar el buffer de envio
   PurgeComm(hPuertoSerie, PURGE_RXCLEAR);

   //Se envia el ID del comando
   WriteFile(hPuertoSerie, &TramaSerie.ID, 1, &BytesEscritos, NULL);
   if (BytesEscritos != 1) return false;

   //A continuacion se envian los datos del comando, si los tiene
   if (TramaSerie.nDatos != 0) {
      WriteFile(hPuertoSerie, &TramaSerie.Datos, TramaSerie.nDatos,
&BytesEscritos, NULL);
      if (BytesEscritos != TramaSerie.nDatos) return false;
   }

   //Se calcula el checksum
   TramaSerie.Checksum = TramaSerie.ID;
   for (i=0; i<TramaSerie.nDatos; i++) TramaSerie.Checksum +=
TramaSerie.Datos[i];
   TramaSerie.Checksum = ~TramaSerie.Checksum;

   //Se envia el checksum
   WriteFile(hPuertoSerie, &TramaSerie.Checksum, 1, &BytesEscritos, NULL);
   if (BytesEscritos != 1) return false;

   return true;
}
```

```
static bool RecibirRespuestaTasador() {
   unsigned long BytesLeidos;
   unsigned long BytesEsperados;
   unsigned char ChecksumCalculado;
   unsigned long i;

   //Se recibe el ID de la respuesta
   ReadFile(hPuertoSerie, &TramaSerie.ID, 1, &BytesLeidos, NULL);
   if (BytesLeidos != 1) return false;

   //En base al ID, se determina la cantidad de bytes a esperar
   switch (TramaSerie.ID) {
   case IDCS_LEER_FECHA_HORA:
      BytesEsperados = 6;
      break;
   case IDCS_ESTABLECER_FECHA_HORA:
      BytesEsperados = 0;
      break;
   case IDCS_LEER_REGISTRO:
      BytesEsperados = sizeof(REGISTRO);
      break;
   case IDCS_BORRAR_REGISTROS:
      BytesEsperados = 0;
      break;
   default:
      return false;
      break;
   }

   //Se procede a tomar los bytes esperados
   ReadFile(hPuertoSerie, &TramaSerie.Datos, BytesEsperados, &BytesLeidos,
NULL);
   if (BytesLeidos != BytesEsperados) return false;
   else TramaSerie.nDatos = BytesLeidos;

   //Se recibe ahora el checksum
   ReadFile(hPuertoSerie, &TramaSerie.Checksum, 1, &BytesLeidos, NULL);
   if (BytesLeidos != 1) return false;

   //Se calcula el checksum localmente
   ChecksumCalculado = TramaSerie.ID;
   for (i=0; i<TramaSerie.nDatos; i++) ChecksumCalculado +=
TramaSerie.Datos[i];
   ChecksumCalculado = ~ChecksumCalculado;

   //Se comparan ambos checksum
   if (ChecksumCalculado != TramaSerie.Checksum) return false;

   return true;
}
```

COMUNICACION_SERIE.h

```
#ifndef Comunicacion_serie_h_Incluida
#define Comunicacion_serie_h_Incluida
```

```
//Declaracion de los tipos de datos compartidos
typedef struct _FECHA_HORA {
   unsigned char Seconds;
   unsigned char Minutes;
   unsigned char Hours;
   unsigned char Date;
   unsigned char Month;
   unsigned char Year;
} FECHA_HORA;

typedef union _REGISTRO {
   struct {
      //Numero telefonico marcado
      unsigned char NumTel[20];
      //Fecha y hora de inicio de la llamada
      FECHA_HORA FH_Inicio;
      //Fecha y hora de fin de la llamada
      FECHA_HORA FH_Final;
   };

   unsigned char Datos[32];
} REGISTRO;

//Declaracion de las variables y datos compartidos
extern FECHA_HORA BufferFechaHora;
extern REGISTRO BufferRegistro;

//Declaracion de las funciones compartidas
bool LeerFechaHora();
bool EstablecerFechaHora();
bool LeerRegistroTasador(unsigned short NumReg);
bool ObtenerTotalRegistrosTasador(unsigned short *TotalRegistros);
bool BorrarRegistrosTasador();

#endif //Comunicacion_serie_h_Incluida
```

PRINCIPAL.cpp

```
//Inclusion de las librerias estandar
#include <windows.h>
#include <stdio.h>

//Inclusion de la cabecera de recursos
#include "recursos.h"

//Inclusion de la cabecera propia
#include "Principal.h"

//Inclusion de las cabeceras de los otros modulos
#include "Comunicacion serie.h"

#define WMU_TICK_RTC WM_APP + 0

//Declaracion previa de las variables globales
HANDLE hPuertoSerie = INVALID_HANDLE_VALUE;
```

```
//Declaracion previa de las variables locales
static HWND hWndVentanaPrincipal = NULL;

//Declaracion previa de las funciones
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nShowCmd);
static BOOL CALLBACK FuncionVentanaPrincipal(HWND hWndDlg, UINT Msg, WPARAM
wParam, LPARAM lParam);
static bool AbrirPuertoSerie();
static void CerrarPuertoSerie();
static void ActualizarFechaHoraPantalla();
static bool SincronizarFechaHoraTasador();
static bool GuardarRegistrosTasador(HWND hVentanaPadre);
static bool BorrarContenidoTasador(HWND hVentanaPadre);

//+--------------------------------+
//| Inicio del codigo del programa |
//+--------------------------------+-------------------------------------
----------------------
//Funcion principal y punto de entrada del programa
//-----------------------------------------------
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nShowCmd) {
   HWND hVentanaPrincipal;
   MSG Mensaje;
   bool Salir = false;
   unsigned long tActual;
   unsigned long tAnterior;

   //Primeramente se abre el puerto serie
   if (!AbrirPuertoSerie()) 1;//return 0;

   //El programa solo corre la funcion de la ventana principal mientras esta
operativo
   //DialogBox(hInstance, MAKEINTRESOURCE(IDD_VENTANA_PRINCIPAL), NULL,
FuncionVentanaPrincipal);
   hVentanaPrincipal = CreateDialog(hInstance,
MAKEINTRESOURCE(IDD_VENTANA_PRINCIPAL), NULL, FuncionVentanaPrincipal);
   ShowWindow(hVentanaPrincipal, SW_SHOWNORMAL);

   //Configura el timer multimedia
   timeBeginPeriod(1);
   tAnterior = timeGetTime();    //Inicializa la lectura de tiempo

   //Bucle para el manejo de mensajes del thread
   while (!Salir) {  //El bucle se repite mientras no se deba salir
      //Verifica si hay un mensaje en cola
      if (PeekMessage(&Mensaje, NULL, 0, 0, PM_REMOVE)) {
         //Si lo hay, verifica si es el mensaje para terminar el programa
         if (Mensaje.message == WM_QUIT)
            //En caso de serlo, se provoca la salida del bucle
            Salir = true;
         else {
            //Sino, se procesa el mensaje de manera normal
            TranslateMessage(&Mensaje);
```

```
            DispatchMessage(&Mensaje);
         }
      }
      else {
         //Si no hay mensajes en cola verifica si es hora de actualizar el
reloj
         tActual = timeGetTime();

         if (tActual - tAnterior >= 100) {
            //Si es necesario actualizar el reloj, se envia el mensaje para
ello a la ventana
            //principal
            SendMessage(hVentanaPrincipal, WMU_TICK_RTC, 0, 0);
            tAnterior = tActual;    //Marca la ultima actualizacion del
reloj
         }
         else {
            //Si no es necesario atender mensajes ni actualizacion de reloj,
se deja inactivo el
            //thread por un milisegundo (para ahorrar uso de CPU)
            Sleep(1);
         }
      }
   }

   //Libera el timer multimedia
   timeEndPeriod(1);

   //Al final se cierra el puerto serie
   CerrarPuertoSerie();
   return 0;
}

//Funcion de manejo de mensajes de la ventana principal
//-----------------------------------------------------
static BOOL CALLBACK FuncionVentanaPrincipal(HWND hWndDlg, UINT Msg, WPARAM
wParam, LPARAM lParam) {
   switch (Msg) {
   case WM_INITDIALOG:
      {
         unsigned long AnchoPantalla, AltoPantalla;
         RECT RectVentana;
         unsigned long PosX, PosY;

         //Se obtienen las dimensiones de la pantalla en pixeles
         AnchoPantalla = GetSystemMetrics(SM_CXSCREEN);
         AltoPantalla = GetSystemMetrics(SM_CYSCREEN);

         //Se obtiene la ubicacion y medidas de la ventana principal
         GetWindowRect(hWndDlg, &RectVentana);

         //Se calcula la posicion centrada de la ventana
         PosX = (AnchoPantalla - (RectVentana.right - RectVentana.left)) /
2;
         PosY = (AltoPantalla - (RectVentana.bottom - RectVentana.top)) / 2;
```

```
         //Se establece la posicion de la ventana a la nueva ubicacion
centrada
         SetWindowPos(hWndDlg, NULL, PosX, PosY, 0, 0, SWP_NOSIZE);

         //Asimismo se guarda el handle de esta ventana:
         hWndVentanaPrincipal = hWndDlg;
      }
      return true;
   case WM_COMMAND:
      switch (LOWORD(wParam)) {
      case IDC_SINCRONIZAR:
         //Al presionar el boton para sincronizar las horas, se llama a la
funcion de sincronia
         if (!SincronizarFechaHoraTasador())
            MessageBox(hWndDlg, "Error de comunicacion con el tasador",
NULL, MB_OK);
         return true;
      case IDC_GUARDAR:
         //El boton de guardar provoca que se llame la funcion de guardado
de registro
         GuardarRegistrosTasador(hWndDlg);
         return true;
      case IDC_BORRAR:
         //El boton de borrar provoca el envio del comando de borrado al
tasador
         BorrarContenidoTasador(hWndDlg);
         return true;
      default:
         return false;
      }
   case WMU_TICK_RTC:
      //Al darse el momento de tener que actualizar la fecha/hora, se llama
a la funcion
      //correspondiente para ello
      ActualizarFechaHoraPantalla();
      return true;
   case WM_CLOSE:
      //Al recibir un comando de cierre, se destruye el cuadro de dialogo
      DestroyWindow(hWndDlg);
      return true;
   case WM_DESTROY:
      //Cuando el cuadro de dialogo se destruye, se le indica al bucle de
manejo de mensajes que
      //termine el programa
      PostQuitMessage(0);
      return true;
   default:
      return false;
   }
}

//Funcion de apertura y configuracion del puerto serie
//-----------------------------------------------------
static bool AbrirPuertoSerie() {
   DCB Dcb;
   COMMTIMEOUTS CommTimeOuts;
```

```
   //Se crea un handle al puerto serie denominado por "COM1"
   hPuertoSerie = CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING,
                              FILE_ATTRIBUTE_NORMAL, NULL);
   if (hPuertoSerie == INVALID_HANDLE_VALUE) {
      MessageBox(NULL, "No se pudo abrir el puerto serie.", NULL, MB_OK);
      return false;
   }

   //Se define el tamaño de la estructura con la configuracion del puerto
serie y luego se llena
   //con la configuracion actual
   Dcb.DCBlength = sizeof(DCB);
   GetCommState(hPuertoSerie, &Dcb);

   //A partir de la configuracion actual, se modifican los parametros de
comunicacion para
   //adaptarlos al formato usado por el tasador
   Dcb.BaudRate = CBR_9600;
   Dcb.fBinary = TRUE;
   Dcb.fParity = FALSE;
   Dcb.fOutxCtsFlow = FALSE;
   Dcb.fOutxDsrFlow = FALSE;
   Dcb.fDtrControl = DTR_CONTROL_DISABLE;
   Dcb.fDsrSensitivity = FALSE;
   Dcb.fOutX = FALSE;
   Dcb.fInX = FALSE;
   Dcb.fNull = FALSE;
   Dcb.fRtsControl = RTS_CONTROL_DISABLE;
   Dcb.fAbortOnError = FALSE;
   Dcb.ByteSize = 8;
   Dcb.Parity = NOPARITY;
   Dcb.StopBits = ONESTOPBIT;

   //Se stablece la nueva configuracion
   SetCommState(hPuertoSerie, &Dcb);

   //Se procede a configurar los time outs del puerto
   CommTimeOuts.ReadIntervalTimeout = 10;         //Se requiere normalmente
1ms entre caracteres
   CommTimeOuts.ReadTotalTimeoutMultiplier = 2;   //Maximo 2ms por caracter
(el doble de lo necesario)
   CommTimeOuts.ReadTotalTimeoutConstant = 500;   //Se espera cuando minimo
500 milisegundos
   CommTimeOuts.WriteTotalTimeoutMultiplier = 2;  //Se otorga el doble del
tiempo necesario
   CommTimeOuts.WriteTotalTimeoutConstant = 50;  //Otorga cuando minimo 50
milisegundos

   SetCommTimeouts(hPuertoSerie, &CommTimeOuts);

   return true;
}

//Funcion de cierre del puerto serie
```

```
//---------------------------------
static void CerrarPuertoSerie() {
   CloseHandle(hPuertoSerie);
}

//Funcion de actualizacion de fecha y hora en pantalla
//-------------------------------------------------
static void ActualizarFechaHoraPantalla() {
   char Cadena[256];
   SYSTEMTIME HoraSistema;

   //Obtiene la hora local del sistema
   GetLocalTime(&HoraSistema);

   //Actualiza la hora del sistema en pantalla
   sprintf(Cadena, "%.2d:%.2d:%.2d", HoraSistema.wHour, HoraSistema.wMinute,
HoraSistema.wSecond);
   SendDlgItemMessage(hWndVentanaPrincipal, IDC_HORA_SISTEMA, WM_SETTEXT, 0,
(LPARAM)Cadena);

   //Actualiza la fecha del sistema en pantalla
   sprintf(Cadena, "%.2d/%.2d/%.2d", HoraSistema.wDay, HoraSistema.wMonth,
HoraSistema.wYear % 100);
   SendDlgItemMessage(hWndVentanaPrincipal, IDC_FECHA_SISTEMA, WM_SETTEXT,
0, (LPARAM)Cadena);

   //Verifica si el puerto serie esta abierto
   if (hPuertoSerie == INVALID_HANDLE_VALUE) {
      SendDlgItemMessage(hWndVentanaPrincipal, IDC_HORA_TASADOR, WM_SETTEXT,
0,
                         (LPARAM)"Puerto serie no abierto");
      return;
   }

   //Si esta abierto, hace la transaccion para leer la fecha y la hora
   if (!LeerFechaHora()) {
      SendDlgItemMessage(hWndVentanaPrincipal, IDC_HORA_TASADOR, WM_SETTEXT,
0,
                         (LPARAM)"Tasador no detectado");
      SendDlgItemMessage(hWndVentanaPrincipal, IDC_FECHA_TASADOR,
WM_SETTEXT, 0, (LPARAM)NULL);
      return;
   }

   //Actualiza la hora del tasador en pantalla
   sprintf(Cadena, "%.2d:%.2d:%.2d", BufferFechaHora.Hours,
BufferFechaHora.Minutes,
           BufferFechaHora.Seconds);
   SendDlgItemMessage(hWndVentanaPrincipal, IDC_HORA_TASADOR, WM_SETTEXT, 0,
(LPARAM)Cadena);

   //Actualiza la fecha del tasador en pantalla
   sprintf(Cadena, "%.2d/%.2d/%.2d", BufferFechaHora.Date,
BufferFechaHora.Month, BufferFechaHora.Year);
   SendDlgItemMessage(hWndVentanaPrincipal, IDC_FECHA_TASADOR, WM_SETTEXT,
0, (LPARAM)Cadena);
```

```
}

//Funcion de sincronia de hora y fecha del tasador con la PC
//-------------------------------------------------------
static bool SincronizarFechaHoraTasador() {
   SYSTEMTIME HoraSistema;

   //Obtiene la hora local del sistema
   GetLocalTime(&HoraSistema);

   //Copia el contenido de la estructura al buffer
   BufferFechaHora.Seconds = (unsigned char) HoraSistema.wSecond;
   BufferFechaHora.Minutes = (unsigned char) HoraSistema.wMinute;
   BufferFechaHora.Hours   = (unsigned char) HoraSistema.wHour;
   BufferFechaHora.Date    = (unsigned char) HoraSistema.wDay;
   BufferFechaHora.Month   = (unsigned char) HoraSistema.wMonth;
   BufferFechaHora.Year    = (unsigned char) (HoraSistema.wYear % 100);

   //Establece la fecha y hora del sistema al tasador y retorna con el mismo
codigo de la funcion
   return EstablecerFechaHora();
}

//Funcion de conversion de BCD empaquetado a numero entero (binario)
//-----------------------------------------------------------------
static unsigned char BCD2INT(unsigned char Dato) {
   return (Dato >> 4) * 10 + (Dato & 0x0F);
}

//Funcion de almacenamiento del contenido del tasador al disco duro
//-----------------------------------------------------------------
static bool GuardarRegistrosTasador(HWND hVentanaPadre) {
   OPENFILENAME Ofn;
   char RutaArchivo[256] = "";
   HANDLE hArchivo = INVALID_HANDLE_VALUE;
   unsigned short TotalRegistros;
   char Cadena[256];
   char CadNumero[21];
   unsigned short i, j;
   unsigned long BytesEscritos;

   //Primero se obtiene la cantidad de registros almacenada en el tasador
   if (!ObtenerTotalRegistrosTasador(&TotalRegistros)) {
      MessageBox(hVentanaPadre, "Error al obtener la informacion del
tasador", NULL, MB_OK);
      return false;
   }

   //Se verifica que haya al menos un registro contenido en la memoria del
tasador
   if (TotalRegistros == 0) {
      MessageBox(hVentanaPadre, "No existen datos en el tasador", NULL,
MB_OK);
      return false;
   }
```

```
   //Se prepara la estructura para mostrar el cuadro de dialogo "Guardar
Como"
   Ofn.lStructSize = sizeof(OPENFILENAME);
   Ofn.hwndOwner = hVentanaPadre;
   Ofn.hInstance = NULL;
   Ofn.lpstrFilter = "Documento de Texto\0*.txt\0";
   Ofn.lpstrCustomFilter = NULL;
   Ofn.nMaxCustFilter = 0;
   Ofn.nFilterIndex = 1;
   Ofn.lpstrFile = RutaArchivo;
   Ofn.nMaxFile = sizeof(RutaArchivo);
   Ofn.lpstrFileTitle = NULL;
   Ofn.nMaxFileTitle = 0;
   Ofn.lpstrInitialDir = NULL;
   Ofn.lpstrTitle = NULL;
   Ofn.Flags = OFN_OVERWRITEPROMPT | OFN_PATHMUSTEXIST;
   Ofn.lpstrDefExt = "TXT";
   Ofn.lCustData = 0;
   Ofn.lpfnHook = NULL;
   Ofn.lpTemplateName = NULL;

   //Se muestra el cuadro de dialogo
   if (!GetSaveFileName(&Ofn)) {
      MessageBox(hVentanaPadre, "Operacion cancelada", NULL, MB_OK);
      return false;
   }

   //Se procede a crear un nuevo archivo de texto con el nombre y ruta
solicitados por el usuario
   hArchivo = CreateFile(RutaArchivo, GENERIC_READ | GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS,
                         FILE_ATTRIBUTE_NORMAL, NULL);
   if (hArchivo == INVALID_HANDLE_VALUE) {
      MessageBox(hVentanaPadre, "No se pudo crear el archivo", NULL, MB_OK);
      return false;
   }

   //Se procede a leer los registros uno a uno
   for (i=1; i<=TotalRegistros; i++) {
      //Se procede a leer el siguiente registro del tasador
      if (!LeerRegistroTasador(i)) {
         MessageBox(hVentanaPadre, "Error al obtener la informacion del
tasador", NULL, MB_OK);
         CloseHandle(hArchivo);
         return false;
      }

      //Se convierte el numero telefonico almacenado en el registro actual a
una cadena de texto
      for (j=0; j<20; j++) {
         //Si se encuentra el caracter que marca el final del numero, se
detiene el proceso
         if (BufferRegistro.NumTel[j] == 0x10) break;

         //Caso contrario, se convierte y agrega el nuevo digito a la cadena
         switch (BufferRegistro.NumTel[j]) {
```

```
            case 1:  CadNumero[j] = '1'; break;
            case 2:  CadNumero[j] = '2'; break;
            case 3:  CadNumero[j] = '3'; break;
            case 4:  CadNumero[j] = '4'; break;
            case 5:  CadNumero[j] = '5'; break;
            case 6:  CadNumero[j] = '6'; break;
            case 7:  CadNumero[j] = '7'; break;
            case 8:  CadNumero[j] = '8'; break;
            case 9:  CadNumero[j] = '9'; break;
            case 10: CadNumero[j] = '0'; break;
            case 11: CadNumero[j] = '*'; break;
            case 12: CadNumero[j] = '#'; break;
            default: CadNumero[j] = '?'; break;
            }
      }
      //Se anexa un cero terminador al final de la cadena
      CadNumero[j] = 0x00;

      //Copia el numero telefonico a la cadena
      strcpy(Cadena, CadNumero);

      //Prepara la cadena con la hora inicial
      sprintf(CadNumero, " %.2d:%.2d:%.2d",
BCD2INT(BufferRegistro.FH_Inicio.Hours),
              BCD2INT(BufferRegistro.FH_Inicio.Minutes),
              BCD2INT(BufferRegistro.FH_Inicio.Seconds));

      //Anexa la cadena
      strcat(Cadena, CadNumero);

      //Prepara la cadena con la fecha inicial
      sprintf(CadNumero, " %.2d/%.2d/%.2d",
BCD2INT(BufferRegistro.FH_Inicio.Date),
              BCD2INT(BufferRegistro.FH_Inicio.Month),
              BCD2INT(BufferRegistro.FH_Inicio.Year));

      //Concatena a lo que lleva
      strcat(Cadena, CadNumero);

      //Prepara la cadena con la hora final
      sprintf(CadNumero, " %.2d:%.2d:%.2d",
BCD2INT(BufferRegistro.FH_Final.Hours),
              BCD2INT(BufferRegistro.FH_Final.Minutes),
              BCD2INT(BufferRegistro.FH_Final.Seconds));

      strcat(Cadena, CadNumero);

      //Prepara la cadena con la fecha final
      sprintf(CadNumero, " %.2d/%.2d/%.2d\r\n",
BCD2INT(BufferRegistro.FH_Final.Date),
              BCD2INT(BufferRegistro.FH_Final.Month),
              BCD2INT(BufferRegistro.FH_Final.Year));

      strcat(Cadena, CadNumero);

      //Se escribe al archivo la informacion generada
```

```
      WriteFile(hArchivo, Cadena, strlen(Cadena), &BytesEscritos, NULL);
      if (BytesEscritos != strlen(Cadena)) {
         MessageBox(hVentanaPadre, "Error al escribir en el archivo", NULL,
MB_OK);
         CloseHandle(hArchivo);
         return false;
      }

   }

   //Al terminar el proceso se cierra el archivo
   CloseHandle(hArchivo);

   //Finalmente se muestra un mensaje de exito en la operacion
   sprintf(Cadena, "Se han guardado exitosamente %i llamadas",
TotalRegistros);
   MessageBox(hWndVentanaPrincipal, Cadena, "Operacion exitosa", MB_OK);
   return true;
}

//Funcion de confirmacion y borrado del contenido de registros del tasador
//---------------------------------------------------------------------------
static bool BorrarContenidoTasador(HWND hVentanaPadre) {
   int ValorRetorno;

   //Se confirma al usuario si realmente desea proseguir con la operacion
   ValorRetorno =  MessageBox(hWndVentanaPrincipal, "Esta seguro de que
desea borrar el"
                              "contenido de llamadas del tasador?",
                              "Confirmacion de borrado",
                              MB_OKCANCEL | MB_ICONWARNING | MB_DEFBUTTON2);

   //Si el usuario cancela, se notifica que no se ha tomado accion
   if (ValorRetorno != IDOK) {
      MessageBox(hVentanaPadre, "No se han hecho cambios al registro del
tasador", "Notificacion", MB_OK);
      return false;
   }

   //Se procede a borrar el contenido del tasador
   if (BorrarRegistrosTasador()) {
      //En caso de exito, se notifica acordemente
      MessageBox(hVentanaPadre, "El contenido del tasador ha sido borrado
con exito", "Notificacion", MB_OK);
      return true;
   }
   else {
      //En caso de falla, se notifica acordemente
      MessageBox(hVentanaPadre, "Error al borrar el contenido del tasador",
NULL, MB_OK);
      return false;
   }
}
```

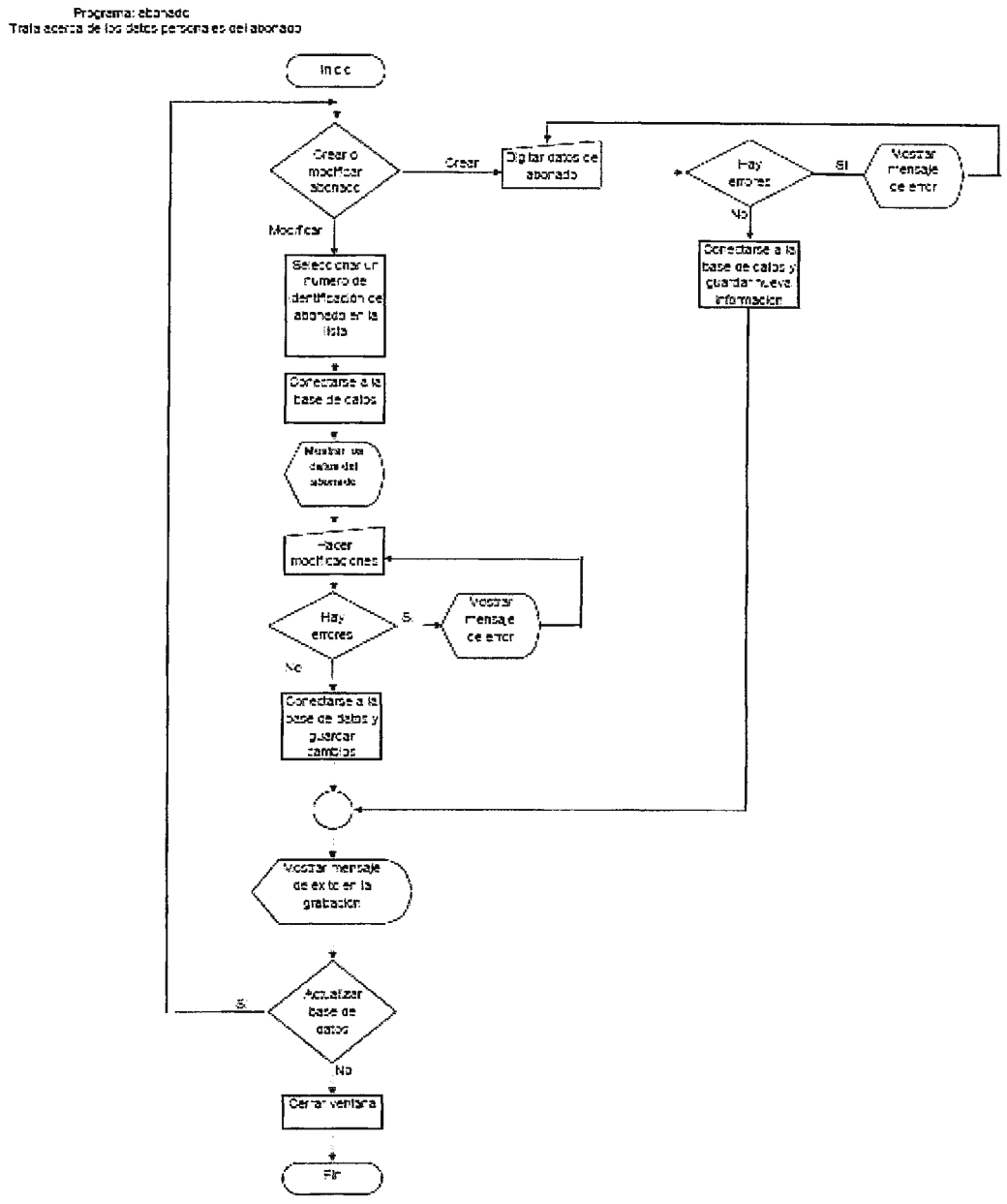
PRINCIPAL.h

```
#ifndef Principal_h_Incluida
#define Principal_h_Incluida

#include <windows.h>

extern HANDLE hPuertoSerie;

#endif //Principal_h_Incluida
```

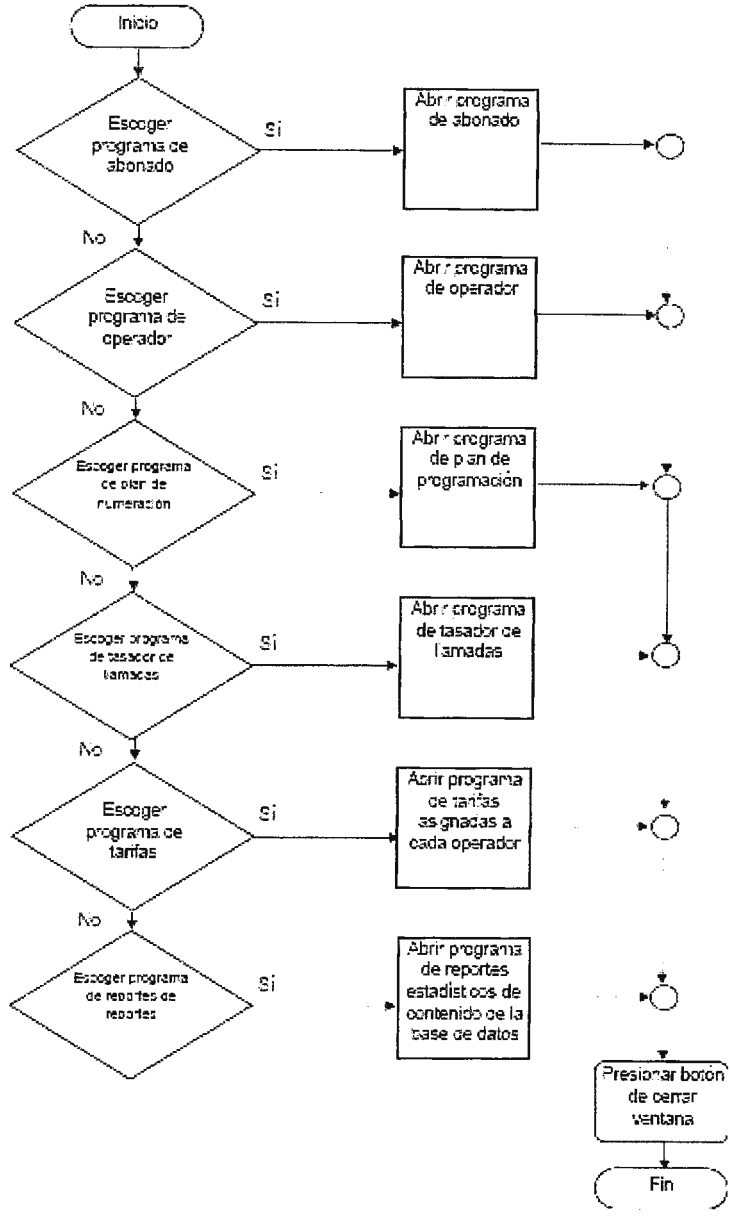
RECURSOS.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Recursos.rc
//
#define IDD_VENTANA_PRINCIPAL           107
#define IDC_IMPORTAR                    1000
#define IDC_GUARDAR                     1000
#define IDC_BORRAR                      1001
#define IDC_BUTTON1                     1002
#define IDC_SINCRONIZAR                 1002
#define IDC_HORA_TASADOR                1004
#define IDC_HORA_SISTEMA                1005
#define IDC_FECHA_TASADOR               1006
#define IDC_FECHA_SISTEMA               1007

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1008
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

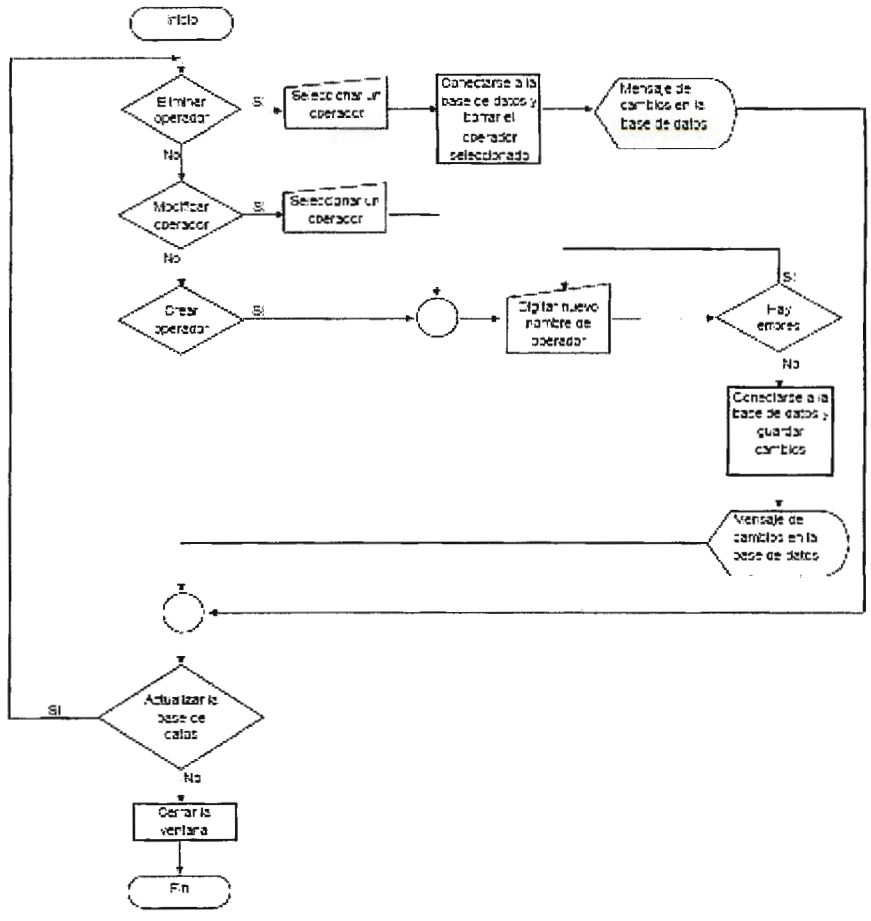
Anexo D: FLUJOGRAMAS DE LA APLICACIÓN MENU TASADOR



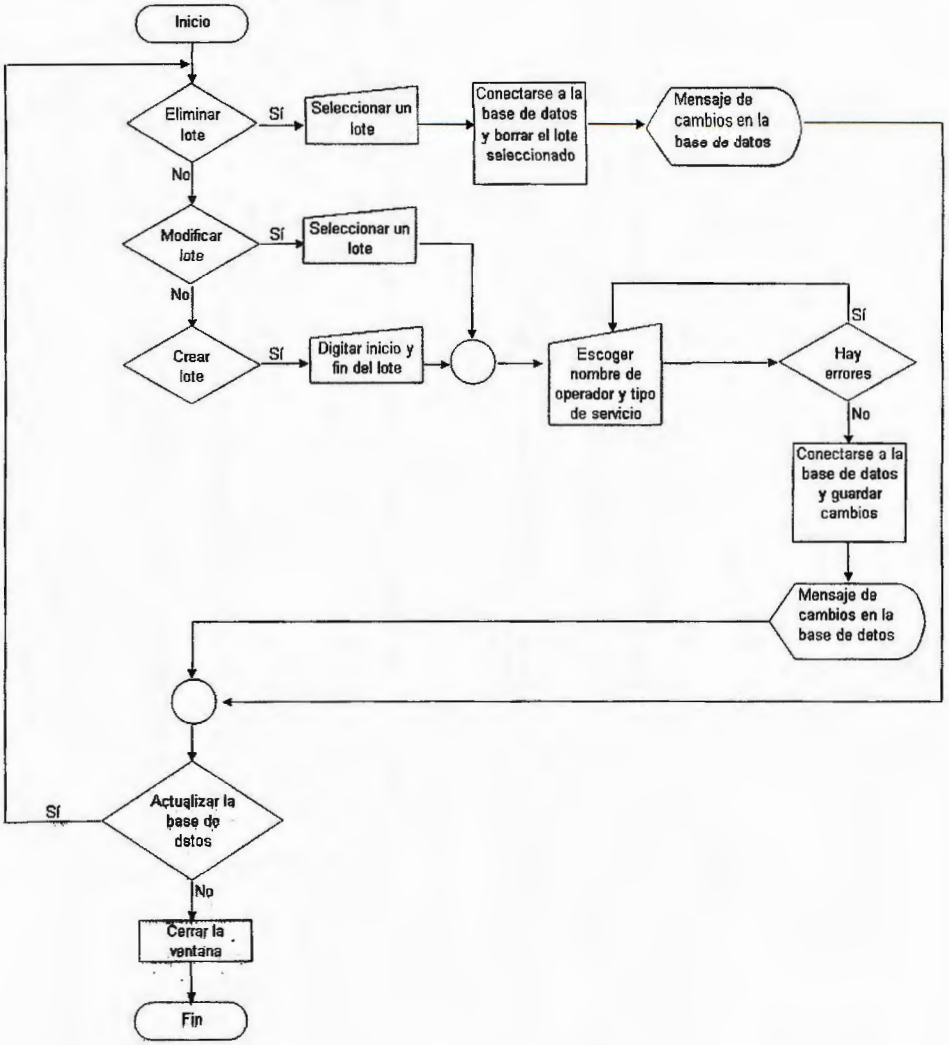
Programa: menu_tasador
Trata acerca del menú principal de la aplicación



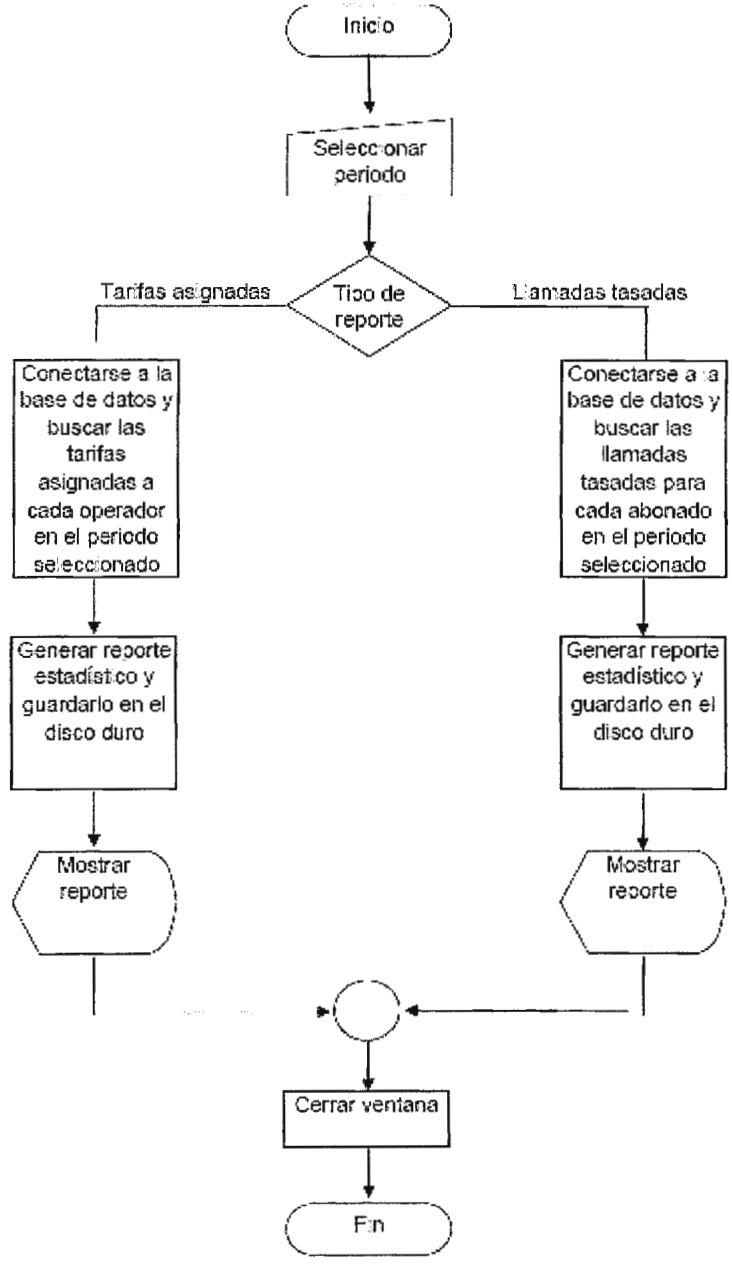
Programa: operacor
Trata acerca del alta y nombre de cada operador



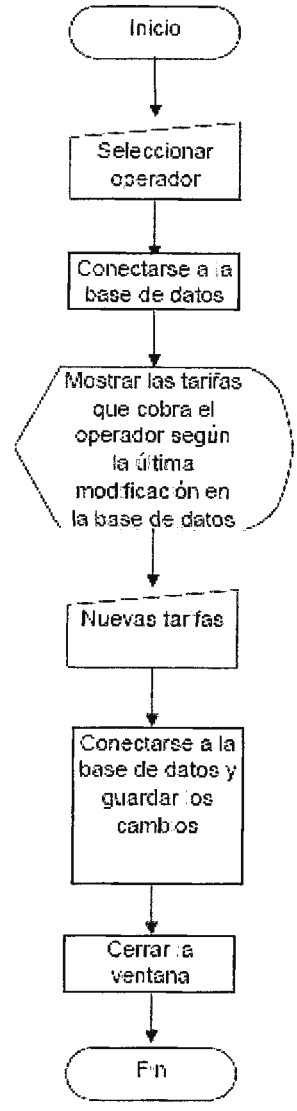
Programa: plan
Trata acerca de los lotes o series numéricas en el plan de numeración



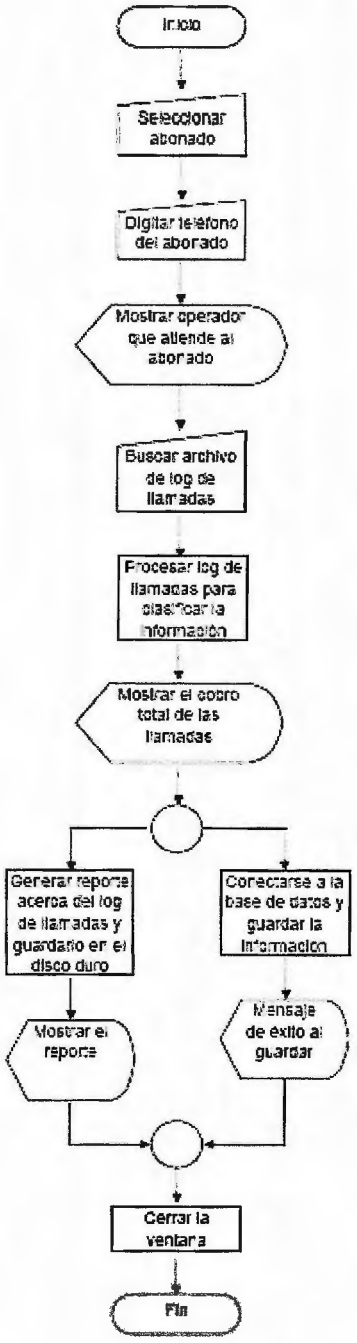
Programa: reportes
Trata acerca de generación de reportes estadísticos según el contenido de la base de datos



Programa: tarifas
Trata acerca de las tarifas de cada operador de telefonía fija



Programa: tasador
Trata acerca de tasación de llamadas de telefonía fija



Anexo E: CODIGO DE PROGRAMAS DE MATLAB

ABONADO

```
function varargout = abonado(varargin)
% Programa para crear nuevos abonados en la base de datos, o modificar los
% que ya existen en la misma.

% ABONADO M-file for abonado.fig
%      ABONADO, by itself, creates a new ABONADO or raises the existing
%      singleton*.
%
%      H = ABONADO returns the handle to a new ABONADO or the handle to
%      the existing singleton*.
%
%      ABONADO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ABONADO.M with the given input arguments.
%
%      ABONADO('Property','Value',...) creates a new ABONADO or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before abonado_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to abonado_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help abonado

% Last Modified by GUIDE v2.5 17-Dec-2009 16:26:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @abonado_OpeningFcn, ...
                   'gui_OutputFcn',  @abonado_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before abonado is made visible.
function abonado_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to abonado (see VARARGIN)

% Choose default command line output for abonado
handles.output = hObject;

% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes abonado wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = abonado_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% ************************************************************************
% ************************** POPUP MENU **********************************
% ************************************************************************

function tipo_Callback(hObject, eventdata, handles)
% Tipo de identificación

% --- Executes during object creation, after setting all properties.
function tipo_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ************************ TEXTOS EDITABLES ******************************
```

```
% ************************************************************************

function nombre_Callback(hObject, eventdata, handles)
% Nombre del abonado

% Nombre del abonado en mayúsculas
set(handles.nombre,'String',upper(get(handles.nombre,'String')));

% --- Executes during object creation, after setting all properties.
function nombre_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------------------------------------------------------------------------

function apellido_Callback(hObject, eventdata, handles)
% Apellido del abonado

% Apellido del abonado en mayúsculas
set(handles.apellido,'String',upper(get(handles.apellido,'String')));

% --- Executes during object creation, after setting all properties.
function apellido_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------------------------------------------------------------------------

function id_Callback(hObject, eventdata, handles)
% # de identificación del abonado

global arreglo_id

if strcmp(get(handles.id,'Style'),'popupmenu'); % si es popupmenu
    opcion=get(handles.id,'Value'); % abonado seleccionado por el usuario
    id=arreglo_id{opcion}; % # id del abonado seleccionado por el usuario
    conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
    query=sprintf('select * from ABONADO where identificacion=''%s'' ',id);
    cursor=exec(conexion,query); % Busca los #id de los abonadoes
    cursor=fetch(cursor); % recorre los datos de la lista uno por uno
    datos=cursor.Data; % datos es un arreglo de celdas que contiene los
datos del abonado seleccionado
    close(conexion); % Se desconecta de la base de datos

    % Muestra los datos del abonado seleccionado:
    set(handles.nombre,'String',datos{1});
    set(handles.apellido,'String',datos{2});
    set(handles.tipo,'String',datos{3});
    set(handles.tipo,'Value',1);
```

```
    set(handles.direccion,'String',datos{5});

    % Habilita los campos para que puedan modificarse:
    set(handles.nombre,'Enable','on');
    set(handles.apellido,'Enable','on');
    set(handles.direccion,'Enable','on');
    set(handles.tipo,'Enable','on');

    % Cambia de popup menu a edit text para que se pueda modificar el #id
    set(handles.id,'String',id);
    set(handles.id,'Style','edit');
    set(handles.aceptar,'Enable','on');
else % si es texto editable
    set(handles.id,'String',upper(get(handles.id,'String'))); % pone las
letras en mayúsculas
end

% --- Executes during object creation, after setting all properties.
function id_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function direccion_Callback(hObject, eventdata, handles)
% Dirección del abonado

% Dirección del abonado en mayúsculas
set(handles.direccion,'String',upper(get(handles.direccion,'String')));

% --- Executes during object creation, after setting all properties.
function direccion_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ********************** RADIOBUTTONS ********************************
% ************************************************************************

function crear_Callback(hObject, eventdata, handles)
% Opción de crear un abonado

set(handles.crear,'Value',1);
set(handles.modificar,'Value',0);
set(handles.id,'Style','edit'); % campo de id como texto editable
set(handles.id,'BackgroundColor',[1,1,1]); % edit color blanco
set(handles.id,'String',''); % borra el texto del campo de id
set(handles.nombre,'String','');
set(handles.apellido,'String','');
set(handles.direccion,'String','');
```

```
set(handles.nombre,'Enable','on'); % habilita el campo de nombre
set(handles.apellido,'Enable','on');
set(handles.direccion,'Enable','on');
set(handles.tipo,'Enable','on');
set(handles.aceptar,'Enable','on');
set(handles.tipo,'String',{'NIT','Pasaporte'}); % establece la lista de
opciones del tipo de identificación

% --------------------------------------------------------------------

function modificar_Callback(hObject, eventdata, handles)
% Opción de modificar los datos personales de un abonado

global arreglo_id

set(handles.crear,'Value',0);
set(handles.modificar,'Value',1);
set(handles.id,'Style','popupmenu'); % campo de id como lista desplegable
set(handles.id,'BackgroundColor',[1,1,1]); % color blanco
set(handles.nombre,'Enable','off'); % habilita el campo de nombre
set(handles.apellido,'Enable','off');
set(handles.direccion,'Enable','off');
set(handles.tipo,'Enable','off');
set(handles.aceptar,'Enable','off');

% Listado de abonados:
conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
cursor=exec(conexion,'select identificacion from ABONADO order by
identificacion'); % Busca los #id de los abonadoes
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
arreglo_id=cursor.Data; % es un arreglo de celdas
set(handles.id,'String',arreglo_id); % ahora los #id van a aparecer en el
pop up menu
close(conexion) % Se desconecta de la base de datos;

% ************************************************************************
% ********************** BOTONES *****************************************
% ************************************************************************

function aceptar_Callback(hObject, eventdata, handles)
% Botón Aceptar

global arreglo_id

set(handles.id,'BackgroundColor',[1,1,1]); % campo #id de color blanco
opcion_id=get(handles.tipo,'Value'); % posición de la lista del tipo de
identificación
if opcion_id==1 % si la posición es 1
    tipo='NIT'; % entonces es NIT
else tipo='Pasaporte'; % si no, entonces es pasaporte
end

nombre=get(handles.nombre,'String'); % nombre del abonado
apellido=get(handles.apellido,'String'); % apellido
```

```
direccion=get(handles.direccion,'String'); % dirección

conexion=database('ODBCTasador','',''); % Se conecta a la base de datos

if get(handles.crear,'Value') % crear abonado:
    % Verifica si se escribió un #id o se ha dejado vacío ese campo:
    % Si no hay #id da mensaje de error. Si hay #id entonces se conecta a
    % la base de datos:
    id=get(handles.id,'String'); % #id escrito por el usuario
    caracteres_escritos=length(id); % cuántos caracteres tiene el #id
    if caracteres_escritos==0
    cadena=sprintf('Debe escribir un número de identificación para el
abonado de telefonía fija.\n\nSi desea cancelar sólo cierre la ventana.');
    uiwait(msgbox(cadena,'Error','error')); % mensaje de error
    for k=1:20 % el edit cambia de color

eval(sprintf('set(handles.id,''BackgroundColor'',[%d/20,0.5,0.5])',k));
        pause(0.05);
    end % fin de for
    else % si hay #id se crear y se conecta a la base de datos

insert(conexion,'ABONADO',{'nombre','apellido','tipo_identificacion','identi
ficacion','direccion'},{nombre,apellido,tipo,id,direccion});
        msgbox('Ha agregado un nuevo abonado a la base de
datos.','Aviso','warn');
    end
else % modificar abonado:
    opcion=get(handles.id,'Value');
    id=arreglo_id{opcion};
    condicion=sprintf('where identificacion=''%s''',id);
    id_nuevo=get(handles.id,'String');

update(conexion,'ABONADO',{'nombre','apellido','direccion','identificacion'}
,{nombre,apellido,direccion,id_nuevo},condicion);
    msgbox('Ha modificado los datos de un abonado en la base de
datos.','Aviso','warn');
end

close(conexion); % Se desconecta de la base de datos
set(handles.actualizar,'Visible','on'); % aparece el botón de actualizar

% --------------------------------------------------------------------

function actualizar_Callback(hObject, eventdata, handles)
% Botón Actualizar
% Actualiza el contenido de la base datos regresando al inicio de este
programa
abonado;
set(handles.actualizar,'Visible','off');
set(handles.id,'BackgroundColor',[1,1,1]); % campo #id de color blanco

% --------------------------------------------------------------------

function volver_Callback(hObject, eventdata, handles)
```

```
% Botón para regresar al menú principal
menu_tasador;
```

MENU TASADOR

```
function varargout = menu_tasador(varargin)
% Programa que muestra el menú principal de la aplicación

% MENU_TASADOR M-file for menu_tasador.fig
%      MENU_TASADOR, by itself, creates a new MENU_TASADOR or raises the
existing
%      singleton*.
%
%      H = MENU_TASADOR returns the handle to a new MENU_TASADOR or the
handle to
%      the existing singleton*.
%
%      MENU_TASADOR('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MENU_TASADOR.M with the given input
arguments.
%
%      MENU_TASADOR('Property','Value',...) creates a new MENU_TASADOR or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before menu_tasador_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to menu_tasador_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help menu_tasador

% Last Modified by GUIDE v2.5 16-Nov-2009 21:57:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @menu_tasador_OpeningFcn, ...
                   'gui_OutputFcn',  @menu_tasador_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before menu_tasador is made visible.
function menu_tasador_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes menu_tasador wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = menu_tasador_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% ************************************************************************
% ****************************** BOTONES *********************************
% ************************************************************************

function boton_aceptar_Callback(hObject, eventdata, handles)
% Botón Aceptar

% Dependiendo de la opción escogida por el usuario, abre la ventana para
% efectuar los cambios que el usuario desea
if get(handles.radio_abonado,'Value')
    abonado; % datos personales del abonado
elseif get(handles.radio_operador,'Value')
    operador; % datos de operadores
elseif get(handles.radio_plan,'Value')
    plan; % plan de numeración
```

```
elseif get(handles.radio_tasacion,'Value')
    tasador; % tasador de llamadas telefónicas
elseif get(handles.radio_tarifas,'Value')
    tarifa; % tarifas asignadas a cada operador
elseif get(handles.radio_reportes,'Value')
    reporte; % reportes estadísticos de la base de datos
end


% ************************************************************************
% ************************* BARRA DE MENÚ ********************************
% ************************************************************************

function Ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to Ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Manual_Callback(hObject, eventdata, handles)
% Abre el manual de usuario, el cual debe estar contenido dentro de la
% misma carpeta que el resto de archivos
!Manual.pdf


% --------------------------------------------------------------------
function Acerca_Callback(hObject, eventdata, handles)
% Texto acerca de la creación de la aplicación

texto=sprintf('Creado para la Superintendencia General de Electricidad y
Telecomunicaciones de El Salvador por D. E. Torres y \nJ. B. Ventura, como
proyecto de graduación de la \nUniversidad Don Bosco.\nDiciembre de
2009.\n\nSoftware: MATLAB\nVersión: 7.5.0 (R2007b)\n© 1984-2007 The
MathWorks, Inc.');

% Logotipo UDB:
[a,map]=imread('udb.png');
[r,c,d]=size(a);
x=ceil(r/54);
y=ceil(c/54);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;

% Ventana de mensaje de texto:
msgbox(texto,'Tasador de línea de telefonía fija','custom',g);
```

OPERADOR

```
function varargout = operador(varargin)
% Programa para agregar un nuevo operador a la base de datos, ya sea que
% preste servicio de telefonía fija o no; y para modificar el nombre de los
% operadores ya existentes en la misma.

% OPERADOR M-file for operador.fig
%      OPERADOR, by itself, creates a new OPERADOR or raises the existing
%      singleton*.
%
%      H = OPERADOR returns the handle to a new OPERADOR or the handle to
%      the existing singleton*.
%
%      OPERADOR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in OPERADOR.M with the given input arguments.
%
%      OPERADOR('Property','Value',...) creates a new OPERADOR or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before operador_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to operador_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help operador

% Last Modified by GUIDE v2.5 17-Dec-2009 18:39:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @operador_OpeningFcn, ...
                   'gui_OutputFcn',  @operador_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before operador is made visible.
```

```
function operador_OpeningFcn(hObject, eventdata, handles, varargin)
global arreglo_nombres
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to operador (see VARARGIN)

% Choose default command line output for operador
handles.output = hObject;

% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Listado de operadores:
conexion=database('ODBCTasador','','');% Se conecta a la base de datos
cursor=exec(conexion,'select Nombre,Fijo,Id from OPERADOR order by Nombre');
% Busca los nombres de los operadores
cursor=fetch(cursor); % recorre los datos de la lista uno por uno eso es lo
que hace el fetch
arreglo_nombres=cursor.Data; % es un arreglo de celdas

set(handles.lista_operadores,'String',arreglo_nombres(:,1)); % ahora los
nombres van a aparecer en el pop up menu
close(conexion); % Se desconecta de la base de datos

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes operador wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = operador_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% ************************************************************************
% ********************** POPUP MENU **********************************
% ************************************************************************
```

```
function lista_operadores_Callback(hObject, eventdata, handles)
% Lista desplegable con los nombres de los operadores

global arreglo_nombres

% Indica si el operador es de telefonía fija o no:
opcion=get(handles.lista_operadores,'Value'); % opción escogida por el
usuario
fijo=arreglo_nombres{opcion,2}; % toma el estado de servicio de telefonía
fija
if strcmp(fijo,'no') % si no da servicio de telefonía fija
    set(handles.tel_fija,'Value',0) % entonces desmarca la casilla de
selección
else set(handles.tel_fija,'Value',1) % pero si lo da entonces marca la
casilla de selección
end

% --- Executes during object creation, after setting all properties.
function lista_operadores_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ********************** TEXTOS EDITABLES ********************************
% ************************************************************************

function edit1_Callback(hObject, eventdata, handles)
% Texto editable del nombre del operador

% Nombre del operador en mayúsculas
set(handles.edit1,'String',upper(get(handles.edit1,'String')));

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ********************** RADIOBUTTONS ************************************
% ************************************************************************

function crear_Callback(hObject, eventdata, handles)
% Opción de crear un operador

set(handles.lista_operadores,'Visible','off'); %desaparece el popup
set(handles.text2,'String','Nombre:');
set(handles.edit1,'BackgroundColor',[1,1,1]); % edit color blanco
set(handles.text2,'Visible','on'); % aparece texto
set(handles.edit1,'Visible','on'); % aparece edit
```

```
set(handles.tel_fija,'Value',1); % inicializa el operador como de telefonía
fija
set(handles.tel_fija,'Visible','on'); % aparece la casilla de telefonía fija

% --------------------------------------------------------------------

function modificar_Callback(hObject, eventdata, handles)
% Opción de modificar un operador

set(handles.lista_operadores,'Visible','on'); %aparece el popup
set(handles.text2,'String','Nuevo nombre:');
set(handles.edit1,'BackgroundColor',[1,1,1]); % edit color blanco
set(handles.text2,'Visible','on'); % aparece texto
set(handles.edit1,'Visible','on'); % aparece edit
set(handles.tel_fija,'Visible','on'); % aparece la casilla de telefonía fija

% --------------------------------------------------------------------

function eliminar_Callback(hObject, eventdata, handles)
% Opción de eliminar un operador

set(handles.lista_operadores,'Visible','on'); %aparece el popup
set(handles.text2,'Visible','off'); % desaparece texto
set(handles.edit1,'BackgroundColor',[1,1,1]); % edit color blanco
set(handles.edit1,'Visible','off'); % desaparece edit
set(handles.tel_fija,'Visible','off'); % desaparece la casilla de telefonía
fija

% ************************************************************************
% ***************************** BOTONES **********************************
% ************************************************************************

% --- Executes on button press in aceptar.
function aceptar_Callback(hObject, eventdata, handles)
% Botón Aceptar

global arreglo_nombres
set(handles.edit1,'BackgroundColor',[1,1,1]); % color blanco para texto
editable del nombre de operador

if get(handles.eliminar,'Value') % si se desea eliminar un operador
    conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
    opcion=get(handles.lista_operadores,'Value'); % posición escogida en la
lista de operadores
    nombre_op=arreglo_nombres{opcion,1}; % nombre de operador escogido para
eliminar
    orden_sql=sprintf('delete * from OPERADOR where
Nombre=''%s''',nombre_op);
    cursor=exec(conexion,orden_sql); % elimina el operador en la base de
datos (tablas: operador, plan_numeracion y tarifa)
    close(conexion); % Se desconecta de la base de datos
    msgbox('Ha eliminado un operador en la base de datos. También se ha
modificado el plan de numeración y las tarifas asignadas a dicho
operador.','Aviso','warn');
```

```
    set(handles.actualizar,'Visible','on'); % aparece el botón de actualizar
else % crear o modificar
    % Verifica si el operador es de telefonía fija
    fijo=get(handles.tel_fija,'Value');
    if fijo==1
        fijo='si';
    else fijo='no';
    end

    % Verifica si se escribió un nombre o se ha dejado vacío el campo de
    % nombre:
    nombre_nuevo=get(handles.edit1,'String'); % nombre escrito por el
usuario
    caracteres_escritos=length(nombre_nuevo); % cuántos caracteres tiene el
nombre

% Si no hay nombre da mensaje de error. Si hay nombre entonces se conecta a
% la base de datos:
    if caracteres_escritos==0
        cadena=sprintf('Debe escribir un nombre para el operador de
telefonía fija.\n\nSi desea cancelar sólo cierre la ventana.');
        uiwait(msgbox(cadena,'Error','error')); % mensaje de error
        for k=1:20 % el edit cambia de color

eval(sprintf('set(handles.edit1,''BackgroundColor'',[%d/20,0.5,0.5])',k));
            pause(0.05);
        end % fin de for
    else % si hay nombre verifica si se desea crear o modificar y se conecta
a la base de datos
        conexion=database('ODBCTasador','',''); % Se conecta a la base de
datos
        if get(handles.crear,'Value') % CREAR UN OPERADOR:
            % Primero verifica si ya existe otro operador con el mismo
            % nombre. Si existe, la búsqueda da el nombre. Si no existe, da
No Data:
            consulta_sql=sprintf('select Nombre from OPERADOR where
Nombre=''%s''',nombre_nuevo);
            cursor3=exec(conexion,consulta_sql); % busca el Id del último
operador agregado a la base de datos
            cursor3=fetch(cursor3); % recorre los datos de la lista uno por
uno
            operador_repetido=cursor3.Data; % Id del último operador
agregado a la base de datos
            if ~strcmp(operador_repetido,'No Data') % si existe
                msgbox('No se puede acceder a la base de datos. Ya existe un
operador con el mismo nombre.','Error','error'); % entonces da error
            else % si no existe entonces procede con agregar el nuevo
operador

insert(conexion,'OPERADOR',{'Nombre','Fijo'},{nombre_nuevo,fijo}); % lo crea
en la tabla de operadores
                msgbox('Ha agregado un nuevo operador a la base de
datos.','Aviso','warn');
            end
        else % MODIFICAR UN OPERADOR:
```

```
            opcion=get(handles.lista_operadores,'Value');
            nombre_op=arreglo_nombres{opcion}; % operador escogido para
modificar
            condicion=sprintf('where Nombre=''%s''',nombre_op);

update(conexion,'OPERADOR',{'Nombre','Fijo'},{nombre_nuevo,fijo},condicion);
% hace el cambio en la base de datos
            msgbox('Ha modificado un operador en la base de
datos.','Aviso','warn');
        end
        close(conexion); % Se desconecta de la base de datos
        set(handles.actualizar,'Visible','on'); % aparece el botón de
actualizar
    end
end

% --------------------------------------------------------------------

function actualizar_Callback(hObject, eventdata, handles)
% Botón Actualizar.
% Actualiza el contenido de la base datos regresando al inicio de este
programa
operador;
set(handles.actualizar,'Visible','off');
set(handles.edit1,'String','');
set(handles.lista_operadores,'Value',1);

% --------------------------------------------------------------------

function volver_Callback(hObject, eventdata, handles)
% Botón para volver al menú principal
menu_tasador;

% --------------------------------------------------------------------

function tel_fija_Callback(hObject, eventdata, handles)
% Casilla de selección de telefonía fija
```

PLAN

```
function varargout = plan(varargin)
% Programa para agregar series numéricas a la base de datos

% PLAN M-file for plan.fig
%      PLAN, by itself, creates a new PLAN or raises the existing
%      singleton*.
%
%      H = PLAN returns the handle to a new PLAN or the handle to
%      the existing singleton*.
%
%      PLAN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PLAN.M with the given input arguments.
%
%      PLAN('Property','Value',...) creates a new PLAN or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before plan_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to plan_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help plan

% Last Modified by GUIDE v2.5 18-Dec-2009 14:14:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @plan_OpeningFcn, ...
                   'gui_OutputFcn',  @plan_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before plan is made visible.
function plan_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to plan (see VARARGIN)

% Choose default command line output for plan
handles.output = hObject;

global arreglo_nombres arreglo_servicios

% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Listado de operadores de telefonía fija, móvil, etc.:
conexion=database('ODBCTasador','','');  % Se conecta a la base de datos
cursor=exec(conexion,'select Nombre,Fijo,Id from OPERADOR order by Nombre');
% Busca los nombres de los operadores
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
arreglo_nombres=cursor.Data; % arreglo de celdas, aqui se van a guardar los
nombres
set(handles.operador,'String',arreglo_nombres(:,1)); % ahora los nombres van
a aparecer en el pop up menu
set(handles.operador,'Value',1); % primera posición de la lista

% Servicios:
cursor=exec(conexion,'select distinct tipo from PLAN_NUMERACION order by
tipo'); % Busca los nombres de los servicios
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
arreglo_servicios=cursor.Data; % datos es un arreglo de celdas, aqui se van
a guardar los diferentes servicios
set(handles.servicio,'String',arreglo_servicios); % ahora los servicios van
a aparecer en el pop up menu

close(conexion); % Se desconecta de la base de datos

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes plan wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = plan_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;

% ************************************************************************
% ************************** POPUP MENU **********************************
% ************************************************************************

% --- Executes on selection change in operador.
function operador_Callback(hObject, eventdata, handles)
% Popupmenu con el listado de los operadores

% --- Executes during object creation, after setting all properties.
function operador_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------------------------------------------------------------------------

% --- Executes on selection change in servicio.
function servicio_Callback(hObject, eventdata, handles)
% Popupmenu de los diferentes tipos de servicios de telefonía

% --- Executes during object creation, after setting all properties.
function servicio_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ************************** TEXTO EDITABLE ******************************
% ************************************************************************

function inicio_Callback(hObject, eventdata, handles)
% Texto editable de Inicio del lote, que puede cambiar a lista desplegable

global arreglo_lotes arreglo_servicios arreglo_nombres

if strcmp(get(handles.inicio,'Style'),'edit') % si es texto editable
    cadena=get(handles.inicio,'String'); % captura la cadena y
    numero=convertir_num(cadena,'inicio',handles); % la convierte en tipo
número
    set(handles.inicio,'String',num2str(numero));

    % Verifica si el fin del lote es mayor que el inicio
    fin=str2num(get(handles.fin,'String')); % captura el fin del lote
    if ~isempty(fin) % si ya se escribió el fin de lote
        if fin<numero % si el final de lote es menor que el inicio
```

```
            msgbox('El número con el que inicia el lote es mayor que número
con que finaliza. Por favor corrija, pues el inicio debe ser menor o igual
que el final.','Error','error');
            set(handles.aceptar,'UserData',1); % hay un error y no se podrá
modificar la base de datos
        else set(handles.aceptar,'UserData',0); % no hay error y se podrá
modificar la base de datos
        end
    end

else % si es popup menú
    set(handles.inicio,'Enable','off'); % se deshabilita el edit del inicio
de lote
    opcion=get(handles.inicio,'Value'); % busca la posición escogida del
popup,
    if get(handles.modificar,'Value') % y si se va a modificar entonces:
        cadena=arreglo_lotes{opcion,1}; % captura la cadena de texto
correspondiente,
        set(handles.inicio,'String',cadena) % la deja como único texto y
        set(handles.inicio,'Style','edit') % cambia a texto editable
        % Habilita otros campos
        set(handles.operador,'Enable','on');
        set(handles.servicio,'Enable','on');
        set(handles.solo_fijo,'Enable','on');
        set(handles.todos,'Enable','on');
    end

    % Para el fin del lote
    cadena=arreglo_lotes{opcion,2}; % captura la cadena de texto
correspondiente,
    set(handles.fin,'String',cadena) % y la pone en el campo de fin de lote

    % Para el servicio
    servicio=arreglo_lotes{opcion,3}; % captura el servicio correspondiente
    posicion=find(strcmp(arreglo_servicios,servicio)); % busca en el listado
de servicios el que coincida con el seleccionado
    set(handles.servicio,'Value',posicion); % se ubica en el servicio
correspondiente

    % Para el operador
    idoperador=arreglo_lotes{opcion,4}; % captura el id de operador al que
corresponde el lote seleccionado
    [filas,columnas]=size(arreglo_nombres); % cantidad de operadores en la
base de datos
    for k=1:filas % para cada operador
        if idoperador==arreglo_nombres{k,3} % compara con el id del operador
escogido de la lista de lotes
            posicion=k; % si son iguales entonces toma la posición del
nombre correspondiente en la lista de operadores
        end
    end
    set(handles.operador,'Value',posicion); % se ubica en el servicio
correspondiente
end
```

```
% --- Executes during object creation, after setting all properties.
function inicio_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function fin_Callback(hObject, eventdata, handles)
% Texto editable de Fin del lote

cadena=get(handles.fin,'String');
numero=convertir_num(cadena,'fin',handles);
set(handles.fin,'String',num2str(numero));

% Verifica si el fin del lote es mayor que el inicio
inicio=str2num(get(handles.inicio,'String')); % captura el inicio del lote
if ~isempty(inicio) % si ya se escribió el inicio de lote
    if inicio>numero % si el final de lote es menor que el inicio
        msgbox('El número con el que inicia el lote es mayor que número con
que finaliza. Por favor corrija, pues el inicio debe ser menor o igual que
el final.','Error','error');
        set(handles.aceptar,'UserData',1); % hay un error y no se podrá
modificar la base de datos
    else set(handles.aceptar,'UserData',0); % no hay error y se podrá
modificar la base de datos
    end
end

% --- Executes during object creation, after setting all properties.
function fin_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ************************* RADIOBUTTONS *********************************
% ************************************************************************

function crear_Callback(hObject, eventdata, handles)
% Radiobutton de crear

% Habilitación y deshabilitación de otros campos
set(handles.crear,'Value',1);
set(handles.modificar,'Value',0);
set(handles.eliminar,'Value',0);
set(handles.operador,'Enable','on');
set(handles.servicio,'Enable','on');
```

```
set(handles.solo_fijo,'Enable','on');
set(handles.todos,'Enable','on');
set(handles.inicio,'Enable','on');
set(handles.fin,'Enable','on');

set(handles.inicio,'String',''); % el inicio del lote es campo en blanco
set(handles.inicio,'Style','edit'); % el inicio del lote es texto editable
set(handles.inicio,'BackgroundColor',[1,1,1]); % edit color blanco
set(handles.fin,'String',''); % el inicio del lote es campo en blanco
set(handles.fin,'BackgroundColor',[1,1,1]); % edit color blanco


% --------------------------------------------------------------------

function modificar_Callback(hObject, eventdata, handles)
% Radiobutton de modificar

% Habilitación y deshabilitación de otros campos
set(handles.crear,'Value',0);
set(handles.modificar,'Value',1);
set(handles.eliminar,'Value',0);
set(handles.inicio,'Enable','on');
set(handles.fin,'Enable','off');
ModificaElimina(handles); % Función auxiliar

% --------------------------------------------------------------------

function eliminar_Callback(hObject, eventdata, handles)
% Radiobutton de eliminar

% Habilitación y deshabilitación de otros campos
set(handles.crear,'Value',0);
set(handles.modificar,'Value',0);
set(handles.eliminar,'Value',1);
set(handles.inicio,'Enable','on');
set(handles.fin,'Enable','off');
ModificaElimina(handles); % Función auxiliar

% --------------------------------------------------------------------

function solo_fijo_Callback(hObject, eventdata, handles)
% Radiobutton para mostrar sólo los operadores de telefonía fija. Es sólo un
atajo
% para escoger a un operador en una lista más corta.

% Listado de operadores de telefonía fija:
conexion=database('ODBCTasador','','');
% Se conecta a la base de datos
cursor=exec(conexion,'select Nombre,Fijo,Id from OPERADOR where Fijo=''si''
order by nombre');
% Busca los nombres de los operadores
cursor=fetch(cursor);
% recorre los datos de la lista uno por uno
arreglo_nombres_fijo=cursor.Data;
% arreglo de celdas
set(handles.operador,'String',arreglo_nombres_fijo(:,1));
% ahora los
nombres van a aparecer en el pop up menu
```

```
set(handles.operador,'Value',1); % primera posición de la lista
close(conexion); % Se desconecta de la base de datos

% --------------------------------------------------------------------

function todos_Callback(hObject, eventdata, handles)
% Radiobutton para mostrar todos los operadores

global arreglo_nombres

% Si no es operador de telefonía fija entonces la lista es más larga
set(handles.operador,'String',arreglo_nombres(:,1)); % ahora los nombres van
a aparecer en el pop up menu
set(handles.operador,'Value',1); % primera posición de la lista


% ************************************************************************
% ***************************** BOTONES **********************************
% ************************************************************************

% --- Executes on button press in aceptar.
function aceptar_Callback(hObject, eventdata, handles)
% Botón de aceptar

global arreglo_lotes arreglo_servicios arreglo_nombres

set(handles.inicio,'BackgroundColor',[1,1,1]); % edit de inicio de lote de
color blanco
set(handles.fin,'BackgroundColor',[1,1,1]); % edit de fin de lote de color
blanco
errores=get(handles.aceptar,'UserData'); % cúantos errores hay en los textos
editados por el usuario
posicion_operador=get(handles.operador,'Value'); % posición de la lista de
operadores
nombre_operador=get(handles.operador,'String'); % listado de los operadores
mostrados en la lista
operador=nombre_operador{posicion_operador}; % nombre del operador
correspondiente en la lista según la opción escogida
posicion_servicio=get(handles.servicio,'Value'); % posición de la lista de
tipo de servicio
servicio=arreglo_servicios{posicion_servicio}; % nombre del tipo de servicio
correspondiente en la lista según la opción escogida

% Id de operador:
[filas,columnas]=size(arreglo_nombres); % cantidad de operadores en la base
de datos
for k=1:filas % para cada operador
    if strcmp(operador,arreglo_nombres{k,1}) % compara con el nombre
escogido de la lista
        IdOperador=arreglo_nombres{k,3}; % si son iguales entonces toma el
id correspondiente
    end
end
```

```
% Pasa el cell array arreglo_lotes a matriz:
[filas,columnas]=size(arreglo_lotes); % cantidad de series numéricas o lotes
en el plan de numeración
matriz_lotes=zeros(filas,2); % se inicializa la matriz que contendrá los
inicios y finales de cada lote
for k=1:filas
    matriz_lotes(k,1)=str2double(arreglo_lotes{k,1}); % inicio de lote
    matriz_lotes(k,2)=str2double(arreglo_lotes{k,2}); % fin de lote
end

if errores==0 % no hay errores de escritura en el inicio o final del lote
numérico
    if get(handles.eliminar,'Value') % ELIMINAR lote
        conexion=database('ODBCTasador','',''); % Se conecta a la base de
datos
        opcion=get(handles.inicio,'Value');
        lote_op=str2double(arreglo_lotes{opcion,1}); % lote escogido para
eliminar
        cadena=sprintf('delete * from PLAN_NUMERACION where
Lote_Inicio=%d',lote_op);
        cursor=exec(conexion,cadena); % elimina el lote en la tabla del plan
de numeración
        close(conexion); % Se desconecta de la base de datos
        msgbox('Ha eliminado un lote de números telefónicos del plan de
numeración en la base de datos.','Aviso','warn');
        set(handles.actualizar,'Visible','on'); % aparece el botón de
actualizar
    else % crear o modificar
        % Verifica si se escribió un número o se ha dejado vacío el campo de
        % inicio de lote:
        inicio=get(handles.inicio,'String'); % número escrito por el usuario
para inicio de lote
        caracteres_escritos=length(inicio); % cuántos caracteres tiene el
inicio del lote
        fin=get(handles.fin,'String'); % número escrito por el usuario para
fin de lote
        caracteres_escritos2=length(fin); % cuántos caracteres tiene el fin
del lote

    % Si no hay números de lotes da mensaje de error. Si hay número entonces
se conecta a
    % la base de datos:
        if caracteres_escritos==0
            cadena=sprintf('Debe escribir un número para el inicio del lote
de números telefónicos.\n\nSi desea cancelar sólo cierre la ventana.');
            uiwait(msgbox(cadena,'Error','error')); % mensaje de error
            for k=1:20 % el edit cambia de color

eval(sprintf('set(handles.inicio,''BackgroundColor'',[%d/20,0.5,0.5])',k));
                pause(0.05);
            end % fin de for
        elseif caracteres_escritos2==0
            cadena=sprintf('Debe escribir un número para el final del lote
de números telefónicos.\n\nSi desea cancelar sólo cierre la ventana.');
```

```
            uiwait(msgbox(cadena,'Error','error')); % mensaje de error
            for k=1:20 % el edit cambia de color

eval(sprintf('set(handles.fin,''BackgroundColor'',[%d/20,0.5,0.5])',k));
                pause(0.05);
            end % fin de for
        elseif caracteres_escritos>0 && caracteres_escritos2>0 % si hay
números verifica si se desea crear o modificar y se conecta a la base de
datos
            inicio=str2double(inicio); % inicio convertido de texto a
numérico
            fin=str2double(fin); % fin convertido de texto a numérico
            if get(handles.crear,'Value') % CREAR:
                % Verifica si los datos que ha digitado el usuario se
traslapan
                % con otro lote ya existente busca en la matriz si ya existe
en la base de datos el número
                % digitado por el usuario:
                traslape_inicio=0;
                traslape_fin=0;
                for k=1:filas
                    if inicio>=matriz_lotes(k,1) &&
inicio<=matriz_lotes(k,2)
                        traslape_inicio=1;
                    end
                    if fin>=matriz_lotes(k,1) && fin<=matriz_lotes(k,2)
                        traslape_fin=1;
                    end
                end

                if traslape_inicio==0 && traslape_fin==0 % si no hay
traslape procede a conectarse a la base de datos
                    conexion=database('ODBCTasador','',''); % Se conecta a
la base de datos

insert(conexion,'PLAN_NUMERACION',{'Lote_Inicio','Lote_Fin','Tipo','Id_Opera
dor'},{inicio,fin,servicio,IdOperador}); % agrega el nuevo operador en la
base de datos
                    msgbox('Ha agregado un nuevo lote de números telefónicos
a la base de datos.','Aviso','warn');
                    close(conexion); % Se desconecta de la base de datos
                else % si hay traslape
                    msgbox('No es posible acceder a la base de datos porque
el lote de números telefónicos digitado por el usuario se traslapa con otro
lote existente en la base de datos. Por favor corrija.','Error','error');
                end
            elseif get(handles.modificar,'Value') % MODIFICAR:
                conexion=database('ODBCTasador','',''); % Se conecta a la
base de datos
                opcion=get(handles.inicio,'Value');
                lote_op=str2double(arreglo_lotes{opcion,1}); % lote escogido
para modificar
                condicion=sprintf('where Lote_Inicio=%d',lote_op);

update(conexion,'PLAN_NUMERACION',{'Lote_Inicio','Lote_Fin','Tipo','Id_Opera
```

```
dor'},{inicio,fin,servicio,IdOperador},condicion); %hace el cambio en la
base de datos
                msgbox('Ha modificado un lote de números telefónicos en la
base de datos.','Aviso','warn');
                close(conexion); % Se desconecta de la base de datos
            end
            set(handles.actualizar,'Visible','on'); % aparece el botón de
actualizar
        end
    end
else % si hay un error en la escritura del inicio o del final del lote
numérico
    msgbox('No es posible acceder a la base de datos. Por favor verifique
que estén correctamente escritos el inicio y el final del lote de números
telefónicos.','Aviso','warn');
end

% --------------------------------------------------------------------

% --- Executes on button press in actualizar.
function actualizar_Callback(hObject, eventdata, handles)
% Botón de actualizar. Actualiza el contenido de la base datos regresando al
inicio de este programa

global arreglo_lotes

set(handles.actualizar,'Visible','off');
set(handles.inicio,'Style','edit');
set(handles.inicio,'String','');
set(handles.fin,'String','');
set(handles.crear,'Value',1);
set(handles.inicio,'Enable','on');
set(handles.fin,'Enable','on');
set(handles.operador,'Enable','on');
set(handles.servicio,'Enable','on');
set(handles.solo_fijo,'Enable','on');
set(handles.todos,'Enable','on');

% Listado de lotes del plan de numeración:
conexion=database('ODBCTasador','','');% Se conecta a la base de datos
cursor=exec(conexion,'select Lote_Inicio,Lote_Fin,Tipo,Id_Operador from
PLAN_NUMERACION order by Lote_Inicio'); % Busca los lotes del plan de
numeración
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
datos=cursor.Data; % datos es un arreglo de celdas
arreglo_lotes={}; % aqui se van a guardar los lotes
[filas,col]=size(datos); %es la cantidad de datos: filas
for k=1:filas
    arreglo_lotes{k,1}=num2str(datos{k,1}); % inicio de lotes distintos
pasados de tipo numérico a texto
    arreglo_lotes{k,2}=num2str(datos{k,2}); % fin de lotes distintos pasados
de tipo numérico a texto
    arreglo_lotes{k,3}=datos{k,3}; % tipo de servicio
    arreglo_lotes{k,4}=datos{k,4}; % id de operador
end
```

```
close(conexion); % Se desconecta de la base de datos

% --------------------------------------------------------------------

% --- Executes on button press in volver.
function volver_Callback(hObject, eventdata, handles)
% Botón para regresar al menú principal
menu_tasador;

% ************************************************************************
% ************************ FUNCIONES AUXILIARES ************************
% ************************************************************************

function numero=convertir_num(s,n,handles)
% Función que convierte un texto en número

if isempty(str2num(s)) % si está en blanco o es letra
    uiwait(msgbox('Debe digitar un número positivo. El carácter inválido que
usted digitó se sustituirá por un cero. Por favor
corrija.','Adevertencia','warn'));
    numero=0;
    for k=1:20 % cambia de color para llamar la atención

eval(sprintf('set(handles.%s,''BackgroundColor'',[%d/20,0.5,0.5])',n,k));
        pause(0.05);
    end
else % si se digitó un número, lo convierte en tipo número
    if str2num(s)<0 % si es negativo da error y llama la atención
        uiwait(msgbox('Ha digitado un número negativo, y éste se sustituirá
por uno positivo para que usted ya no tenga que corregir.','Aviso','warn'));
        numero=abs(str2num(s));
        for k=1:20 % cambia de color para llamar la atención

eval(sprintf('set(handles.%s,''BackgroundColor'',[0,%d/20,0])',n,k));
            pause(0.05);
        end
        eval(sprintf('set(handles.%s,''BackgroundColor'',[1,1,1])',n)); %
vuelve a fondo blanco
    else % si no es negativo
        numero=abs(str2num(s));
        eval(sprintf('set(handles.%s,''BackgroundColor'',[1,1,1])',n));
    end
end

% --------------------------------------------------------------------

function ModificaElimina(handles)
% Función que establece las condiciones de los campos cuando se desea
% modificar o eliminar un lote

global arreglo_lotes arreglo_nombres

set(handles.operador,'Enable','off');
set(handles.servicio,'Enable','off');
```

```
set(handles.solo_fijo,'Enable','off');
set(handles.todos,'Enable','off');
set(handles.todos,'Value',1);
set(handles.operador,'String',arreglo_nombres(:,1));

set(handles.inicio,'Style','popup'); % el inicio del lote es popup menú
set(handles.inicio,'BackgroundColor',[1,1,1]); % edit color blanco
set(handles.fin,'BackgroundColor',[1,1,1]); % edit color blanco

% Listado de lotes del plan de numeración:
conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
cursor=exec(conexion,'select Lote_Inicio,Lote_Fin,Tipo,Id_Operador from
PLAN_NUMERACION order by Lote_Inicio'); % Busca los lotes del plan de
numeración
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
datos=cursor.Data; % datos es un arreglo de celdas
arreglo_lotes={}; % aqui se van a guardar los lotes
[filas,col]=size(datos); % es la cantidad de datos: filas
for k=1:filas
    arreglo_lotes{k,1}=num2str(datos{k,1}); % inicio de lotes distintos
pasados de tipo numérico a texto
    arreglo_lotes{k,2}=num2str(datos{k,2}); % fin de lotes distintos pasados
de tipo numérico a texto
    arreglo_lotes{k,3}=datos{k,3}; % tipo de servicio
    arreglo_lotes{k,4}=datos{k,4}; % id operador
end
set(handles.inicio,'String',arreglo_lotes(:,1)); % ahora los lotes van a
aparecer en el pop up menu
close(conexion); % Se desconecta de la base de datos
```

TARIFA

```
function varargout = tarifa(varargin)
% Programa para modificar las tarifas que cobran los operadores de
% telefonía fija

% TARIFA M-file for tarifa.fig
%      TARIFA, by itself, creates a new TARIFA or raises the existing
%      singleton*.
%
%      H = TARIFA returns the handle to a new TARIFA or the handle to
%      the existing singleton*.
%
%      TARIFA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TARIFA.M with the given input arguments.
%
%      TARIFA('Property','Value',...) creates a new TARIFA or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before tarifa_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to tarifa_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help tarifa

% Last Modified by GUIDE v2.5 19-Dec-2009 19:57:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @tarifa_OpeningFcn, ...
                   'gui_OutputFcn',  @tarifa_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tarifa is made visible.
function tarifa_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to tarifa (see VARARGIN)

global arreglo_operador horario

% Choose default command line output for tarifa
handles.output = hObject;

% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Horarios
horario={'00:00','01:00','02:00','03:00','04:00','05:00','06:00',...
    '07:00','08:00','09:00','10:00','11:00','12:00',...
    '13:00','14:00','15:00','16:00','17:00','18:00',...
    '19:00','20:00','21:00','22:00','23:00'};

set(handles.horario_reducido,'String',horario);
set(handles.horario_pleno,'String',horario);


% Listado de operadores:
conexion=database('ODBCTasador','','');
% Se conecta a la base de datos
cursor=exec(conexion,'select Nombre,Fijo,Id from OPERADOR where fijo=''si''
order by nombre');
% Busca los nombres de los operadores
cursor=fetch(cursor);
% recorre los datos de la lista uno por uno
datos=cursor.Data;
% datos es un arreglo de celdas
arreglo_operador=datos;
% Datos de los distintos operadores
set(handles.operador,'String',arreglo_operador(:,1));
% ahora los nombres van a aparecer en el pop up menu
close(conexion);
% Se desconecta de la base de datos



% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tarifa wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = tarifa_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% ************************************************************************
% ************************** POPUP MENUES ********************************
% ************************************************************************

% --- Executes on selection change in operador.
function operador_Callback(hObject, eventdata, handles)
% Pop up menú de los nombres de los operadores

global arreglo_operador horario arreglo_tarifa

% Busca el id del operador escogido:
opcion=get(handles.operador,'Value'); % operador escogido por el usuario,
toma la posición en la lista de operadores
id_operador=arreglo_operador{opcion,3}; % id del operador correspondiente en
la lista según la opción escogida

% Tarifas del operador escogido. Busca todas las tarifas del operador y las
% ordena de menor a mayor según la fecha de modificación, pero se presenta
% en la ventana sólo la más reciente
conexion=database('ODBCTasador','','');% Se conecta a la base de datos
consulta_sql=sprintf('select * from TARIFA where Id_Operador=%d order by
Fecha_modificacion',id_operador);
cursor=exec(conexion,consulta_sql); % Busca las tarifas del operador
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
datos=cursor.Data; % datos es un arreglo de celdas
arreglo_tarifa=datos(end,:); % Toma las tarifas con la fecha más reciente
close(conexion); % Se desconecta de la base de datos

% Verifica si ya hay datos en la tabla de tarifas para operador escogido
if strcmp(arreglo_tarifa{1},'No Data') % si no existen sus tarifas en la
base de datos entonces las muestra como ceros
    set(handles.local_plena,'String','0');
    set(handles.nacional_plena,'String','0');
    set(handles.local_reducida,'String','0');
    set(handles.nacional_reducida,'String','0');
    set(handles.internacional,'String','0');
    set(handles.a_celular,'String','0');
    fecha=datestr(now,'dd/mm/yy HH:MM:SS');
    set(handles.horario_reducido,'Value',13);
    set(handles.horario_pleno,'Value',13);
    set(handles.fecha_modificacion,'String',fecha);
else % pero si existen tarifas entonces las muestra
    horario_pleno=arreglo_tarifa{3}; % Inicio de horario pleno del operador
    % Compara ese horario con las posiciones en el listado del popup menú de
    % horario pleno
```

```
    for k=1:24
        if ~strcmp(horario_pleno,'null') % si se ha establecido un horario
            if strcmp(horario{k},horario_pleno) % entonces compara
                set(handles.horario_pleno,'Value',k); % y si son iguales se
coloca en la posición correspondiente en el listado
            end
        end
    end

    horario_reducido=arreglo_tarifa{4}; % Inicio de horario reducido del
operador
    % Compara ese horario con las posiciones en el listado del popup menú de
    % horario reducido
    for k=1:24
        if ~strcmp(horario_reducido,'null') % si se ha establecido un
horario
            if strcmp(horario{k},horario_reducido) % entonces compara
                set(handles.horario_reducido,'Value',k); % y si son iguales
se coloca en la posición correspondiente en el listado
            end
        end
    end

    % Busca las tarifas y las coloca en los campos respectivos:
    set(handles.local_plena,'String',num2str(arreglo_tarifa{5}));
    set(handles.nacional_plena,'String',num2str(arreglo_tarifa{6}));
    set(handles.local_reducida,'String',num2str(arreglo_tarifa{7}));
    set(handles.nacional_reducida,'String',num2str(arreglo_tarifa{8}));
    set(handles.internacional,'String',num2str(arreglo_tarifa{9}));
    set(handles.a_celular,'String',num2str(arreglo_tarifa{10}));
    fecha=datestr(arreglo_tarifa{2},'dd/mm/yy HH:MM:SS');
    set(handles.fecha_modificacion,'String',fecha);
end

% Habilita los campos:
set(handles.horario_reducido,'Enable','on');
set(handles.horario_pleno,'Enable','on');
set(handles.local_plena,'Enable','on');
set(handles.nacional_plena,'Enable','on');
set(handles.local_reducida,'Enable','on');
set(handles.nacional_reducida,'Enable','on');
set(handles.internacional,'Enable','on');
set(handles.fecha_modificacion,'Enable','on');
set(handles.a_celular,'Enable','on');



% --- Executes during object creation, after setting all properties.
function operador_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --------------------------------------------------------------------

function horario_reducido_Callback(hObject, eventdata, handles)
% Listado de opciones para inicio del horario de tarifa reducida


% --- Executes during object creation, after setting all properties.
function horario_reducido_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function horario_pleno_Callback(hObject, eventdata, handles)
% Listado de opciones para inicio del horario de tarifa plena

% --- Executes during object creation, after setting all properties.
function horario_pleno_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ********************** TEXTOS EDITABLES ********************************
% ************************************************************************

function local_reducida_Callback(hObject, eventdata, handles)
% Tarifa local reducida

cadena=get(handles.local_reducida,'String');
numero=convertir_num(cadena,'local_reducida',handles);
set(handles.local_reducida,'String',num2str(numero));

% --- Executes during object creation, after setting all properties.
function local_reducida_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function local_plena_Callback(hObject, eventdata, handles)
% Tarifa local plena

cadena=get(handles.local_plena,'String');
numero=convertir_num(cadena,'local_plena',handles);
set(handles.local_plena,'String',num2str(numero));
```

```
% --- Executes during object creation, after setting all properties.
function local_plena_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function nacional_reducida_Callback(hObject, eventdata, handles)
% Tarifa nacional reducida

cadena=get(handles.nacional_reducida,'String');
numero=convertir_num(cadena,'nacional_reducida',handles);
set(handles.nacional_reducida,'String',num2str(numero));

% --- Executes during object creation, after setting all properties.
function nacional_reducida_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function nacional_plena_Callback(hObject, eventdata, handles)
% Tarifa nacional plena

cadena=get(handles.nacional_plena,'String');
numero=convertir_num(cadena,'nacional_plena',handles);
set(handles.nacional_plena,'String',num2str(numero));

% --- Executes during object creation, after setting all properties.
function nacional_plena_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function internacional_Callback(hObject, eventdata, handles)
% Tarifa internacional

cadena=get(handles.internacional,'String');
numero=convertir_num(cadena,'internacional',handles);
set(handles.internacional,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function internacional_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function a_celular_Callback(hObject, eventdata, handles)
% Tarifa a a_celular

cadena=get(handles.a_celular,'String');
numero=convertir_num(cadena,'a_celular',handles);
set(handles.a_celular,'String',num2str(numero));

% --- Executes during object creation, after setting all properties.
function a_celular_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ***************************** BOTONES **********************************
% ************************************************************************

% --- Executes on button press in guardar.
function guardar_Callback(hObject, eventdata, handles)
% Botón para guardar los cambios

global arreglo_operador horario

% Busca el id del operador escogido:
opcion=get(handles.operador,'Value'); % operador escogido por el usuario,
toma la posición en la lista de operadores
id_operador=arreglo_operador{opcion,3}; % id del operador correspondiente en
la lista según la opción escogida

% Tarifas:
local_reducida=str2double(get(handles.local_reducida,'String'));
nacional_reducida=str2double(get(handles.nacional_reducida,'String'));
local_plena=str2double(get(handles.local_plena,'String'));
nacional_plena=str2double(get(handles.nacional_plena,'String'));
internacional=str2double(get(handles.internacional,'String'));
a_celular=str2double(get(handles.a_celular,'String'));

% Horarios:
horario_reducido=horario{get(handles.horario_reducido,'Value')};
horario_pleno=horario{get(handles.horario_pleno,'Value')};

% Fecha de modificación:
fecha_mod=datestr(now,'dd/mm/yy HH:MM:SS');

% Se guardan los cambios en la base de datos:
conexion=database('ODBCTasador','','');  % Se conecta a la base de datos
%condicion=sprintf('where Id_Operador=%d',id_operador);
```

```
insert(conexion,'TARIFA',{'Inicio_horario_pleno','Inicio_horario_reducido','
Tarifa_plena_local','Tarifa_plena_nacional','Tarifa_reducida_local','Tarifa_
reducida_nacional','Tarifa_internacional','Tarifa_a_celular','Fecha_modifica
cion','Id_Operador'},{horario_pleno,horario_reducido,local_plena,nacional_pl
ena,local_reducida,nacional_reducida,internacional,a_celular,fecha_mod,id_op
erador}); % se guardan los cambios en la base de datos
close(conexion); % Se desconecta de la base de datos
msgbox('Ha modificado las tarifas de un operador de telefonía fija en la
base de datos.','Aviso','warn');
set(handles.actualizar,'Visible','on'); % habilita el botón de actualizar

% --------------------------------------------------------------------

% --- Executes on button press in actualizar.
function actualizar_Callback(hObject, eventdata, handles)
% Actualiza el contenido de la base datos regresando al inicio de este
programa
tarifa;

set(handles.actualizar,'Visible','off');
set(handles.local_reducida,'Enable','off');
set(handles.nacional_reducida,'Enable','off');
set(handles.local_plena,'Enable','off');
set(handles.nacional_plena,'Enable','off');
set(handles.internacional,'Enable','off');
set(handles.horario_reducido,'Enable','off');
set(handles.horario_pleno,'Enable','off');
set(handles.fecha_modificacion,'String','');
set(handles.local_reducida,'String','');
set(handles.nacional_reducida,'String','');
set(handles.local_plena,'String','');
set(handles.nacional_plena,'String','');
set(handles.internacional,'String','');
set(handles.horario_reducido,'Value',1);
set(handles.horario_pleno,'Value',1);
set(handles.a_celular,'Enable','off');
set(handles.a_celular,'String','');

% --------------------------------------------------------------------

% --- Executes on button press in volver.
function volver_Callback(hObject, eventdata, handles)
% Botón para regresar el menú principal
menu_tasador;

% ************************************************************************
% ************************ FUNCIONES AUXILIARES ************************
% ************************************************************************

function numero=convertir_num(s,n,handles)
% Función que convierte un texto en número

if isempty(str2num(s)) % si está en blanco o es letra
```

```
    uiwait(msgbox('Debe digitar un número positivo. El carácter inválido que
usted digitó se sustituirá por un cero. Por favor
corrija.','Adevertencia','warn'));
    numero=0;
    for k=1:20 % cambia de color para llamar la atención

eval(sprintf('set(handles.%s,''BackgroundColor'',[%d/20,0.5,0.5])',n,k));
        pause(0.05);
    end
else % si se digitó un número, lo convierte en tipo número
    if str2num(s)<0 % si es negativo da error y llama la atención
        uiwait(msgbox('Ha digitado un número negativo, y éste se sustituirá
por uno positivo para que usted ya no tenga que corregir.','Aviso','warn'));
        numero=abs(str2num(s));
        for k=1:20 % cambia de color para llamar la atención

eval(sprintf('set(handles.%s,''BackgroundColor'',[0,%d/20,0])',n,k));
            pause(0.05);
        end
        eval(sprintf('set(handles.%s,''BackgroundColor'',[1,1,1])',n)); %
vuelve a fondo blanco
    else % si no es negativo
        numero=abs(str2num(s));
        eval(sprintf('set(handles.%s,''BackgroundColor'',[1,1,1])',n));
    end
end
```

TASADOR

```
function varargout = tasador(varargin)
% Programa que lee el data log de llamadas de telefonía fija, guarda esta
% información en la base de datos y presenta un reporte en Excel para las
% mismas.

% TASADOR M-file for tasador.fig
%      TASADOR, by itself, creates a new TASADOR or raises the existing
%      singleton*.
%
%      H = TASADOR returns the handle to a new TASADOR or the handle to
%      the existing singleton*.
%
%      TASADOR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TASADOR.M with the given input arguments.
%
%      TASADOR('Property','Value',...) creates a new TASADOR or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before tasador_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to tasador_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help tasador

% Last Modified by GUIDE v2.5 28-Dec-2009 01:43:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @tasador_OpeningFcn, ...
                   'gui_OutputFcn',  @tasador_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tasador is made visible.
function tasador_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to tasador (see VARARGIN)

% Choose default command line output for tasador
handles.output = hObject;

global arreglo_abonado arreglo_lotes matriz_lotes abonados


% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Inicializa edits de número telefónico en blanco
for k=2:8
    eval(sprintf('set(handles.edit%d,''String'','''')',k));
```

```
end

% Se conecta a la base de datos
conexion=database('ODBCTasador','','');

% Lotes de números telefónicos o series numéricas:
cursor=exec(conexion,'select Lote_Inicio,Lote_Fin,Tipo,Id_Operador from
PLAN_NUMERACION order by Lote_Inicio'); % Busca los lotes del plan de
numeración
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
datos=cursor.Data; % datos es un arreglo de celdas
arreglo_lotes={}; % aqui se van a guardar los lotes
[filas,col]=size(datos); %es la cantidad de datos: filas
arreglo_lotes=cell(filas,4); % se inicializa el cell array que contendrá los
datos obtenidos del query
matriz_lotes=zeros(filas,2); % se inicializa la matriz que contendrá los
inicios y finales de cada lote
for k=1:filas
    arreglo_lotes{k,1}=num2str(datos{k,1}); % inicio de lotes distintos
pasados de tipo numérico a texto
    arreglo_lotes{k,2}=num2str(datos{k,2}); % fin de lotes distintos pasados
de tipo numérico a texto
    arreglo_lotes(k,3:4)=datos(k,3:4); % tipo de servicio y operador
    matriz_lotes(k,1)=datos{k,1}; % inicio de lote en numérico
    matriz_lotes(k,2)=datos{k,2}; % fin de lote en numérico
end

% Números de indentificación de cada abonado
cursor=exec(conexion,'select * from ABONADO order by
tipo_identificacion,identificacion'); % Busca los números de identificación
de cada abonado
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
arreglo_abonado=cursor.Data; % cell array que contiene los #id de cada
abonado

close(conexion); % Se desconecta de la base de datos

abonados=arreglo_abonado(1:end,4); % toma los #id de los abonados
set(handles.id_abonado,'String',abonados); % muestra en el popup menu los
#id de los abonados


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tasador wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = tasador_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% ************************************************************************
% ********************** TEXTOS EDITABLES ********************************
% ************************************************************************

function edit1_Callback(hObject, eventdata, handles)
% Primer dígito del número telefónico del abonado, ya está establecido como
% 2 para que el usuario no lo digite.


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------------------------------------------------------------------------

function edit2_Callback(hObject, eventdata, handles)
% Segundo dígito del número telefónico del abonado

cadena=get(handles.edit2,'String');
numero=convertir_num(cadena,2,handles);
set(handles.edit2,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------------------------------------------------------------------------

function edit3_Callback(hObject, eventdata, handles)
% Tercer dígito del número telefónico del abonado

cadena=get(handles.edit3,'String');
numero=convertir_num(cadena,3,handles);
set(handles.edit3,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function edit4_Callback(hObject, eventdata, handles)
% Cuarto dígito del número telefónico del abonado

cadena=get(handles.edit4,'String');
numero=convertir_num(cadena,4,handles);
set(handles.edit4,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function edit5_Callback(hObject, eventdata, handles)
% Quinto dígito del número telefónico del abonado

cadena=get(handles.edit5,'String');
numero=convertir_num(cadena,5,handles);
set(handles.edit5,'String',num2str(numero));



% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function edit6_Callback(hObject, eventdata, handles)
% Sexto dígito del número telefónico del abonado

cadena=get(handles.edit6,'String');
numero=convertir_num(cadena,6,handles);
set(handles.edit6,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function edit7_Callback(hObject, eventdata, handles)
% Séptimo dígito del número telefónico del abonado

cadena=get(handles.edit7,'String');
numero=convertir_num(cadena,7,handles);
set(handles.edit7,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function edit8_Callback(hObject, eventdata, handles)
% Octavo dígito del número telefónico del abonado

cadena=get(handles.edit8,'String');
numero=convertir_num(cadena,8,handles);
set(handles.edit8,'String',num2str(numero));


% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function cobro_Callback(hObject, eventdata, handles)
% Texto editable en el que se muestra automáticamente el cobro total de las
% llamadas. Queda deshabilitado para que el usuario no lo toque.

% --- Executes during object creation, after setting all properties.
function cobro_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------
```

```
function nombre_operador_Callback(hObject, eventdata, handles)
% Texto no editable en el que aparece el nombre del operador.

function nombre_operador_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ************************** POPUP MENU **********************************
% ************************************************************************

function id_abonado_Callback(hObject, eventdata, handles)
% Popup menu o lista desplegable que contiene los abonados en la base de
% datos.


% --- Executes during object creation, after setting all properties.
function id_abonado_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ************************************************************************
% ************************** BOTONES *************************************
% ************************************************************************

function buscar_operador_Callback(hObject, eventdata, handles)
% Botón de buscar operador al que pertence el número telefónico digitado

global telefono id_operador

% Dígitos del número telefónico en tipo numérico
for k=2:8 % para edit text del número telefónico crea una variable que lee
lo escrito por el usuario
    eval(sprintf('d%d=get(handles.edit%d,''String'');',k,k));
end

% Concatenación de las cadenas de texto de los dígitos del teléfono para
% formar un solo número y luego se convierte a tipo numérico:
telefono=str2double(['2',d2,d3,d4,d5,d6,d7,d8]);

% Verifica si no hay errores en el número telefónico digitado por el
% usuario:
for k=2:8 % para edit text del número telefónico crea una variable que lee
si hay error
    eval(sprintf('n%d=get(handles.edit%d,''UserData'');',k,k));
end

if n2*n3*n4*n5*n6*n7*n8==0 % si hay error de escritura da mensaje de error
```

```
    msgbox('Uno de los dígitos del número telefónico está mal escrito o no
está escrito. Por favor corrija.','Error','error');
else % si no hay error se conecta a la base de datos y busca al operador al
que pertenece el número
    conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
    query_sql=sprintf('select Id_Operador from PLAN_NUMERACION where
%d>=Lote_Inicio AND %d<=Lote_Fin',telefono,telefono);
    cursor=exec(conexion,query_sql); % Busca el id del perador
    cursor=fetch(cursor); % recorre los datos de la lista uno por uno
    datos=cursor.Data; % datos es un arreglo de celdas
    id_operador=datos{1}; % se obtiene el id del operador del arreglo de
celdas
    close(conexion); % Se desconecta de la base de datos
    if strcmp(id_operador,'No Data') % si no encuentra al operador
        texto=sprintf('Este número telefónico no está asignado a ningún
operador dentro del plan de numeración. Por favor corrija los datos que
digitó y vuelva a intentar.\n\nSi el número telefónico es correcto entonces
actualice la base de datos:\n- Vaya al Menú Principal\n- Escoja la opción
Plan de numeración\n- Cree un nuevo lote de números telefónicos o serie
numérica.');
        msgbox(texto,'Error','error');
    else % si encuentra al operador habilita y deshabilita campos
        conexion=database('ODBCTasador','',''); % Se conecta a la base de
datos
        query_sql=sprintf('select Nombre from OPERADOR where
Id=%d',id_operador);
        cursor=exec(conexion,query_sql); % Busca el nombre del operador
        cursor=fetch(cursor); % recorre los datos de la lista uno por uno
        datos=cursor.Data; % datos es un arreglo de celdas
        nombre_operador=datos{1}; % se obtiene el nombre del operador del
arreglo de celdas para tenerlo como tipo texto
        close(conexion); % Se desconecta de la base de datos
        set(handles.text5,'Visible','on');
        set(handles.nombre_operador,'String',nombre_operador);
        set(handles.buscar_operador,'Visible','off');
        set(handles.nombre_operador,'Visible','on');
        set(handles.abrir,'Enable','on');
        for k=2:8 % deshabiita los edit text del número telefónico
            eval(sprintf('set(handles.edit%d,''Enable'',''Inactive'');',k));
        end
    end
end

% --------------------------------------------------------------------

function abrir_Callback(hObject, eventdata, handles)
% Botón para abrir el archivo de texto que contiene el data log de las
% llamadas

global nombre nombre_archivo abonados telefono abonado
global arreglo_lotes matriz_lotes id_operador llamada arreglo_llamada

cobro_total=0; % cobro total por las llamadas realizadas
arreglo_llamada={}; % arreglo de celdas que contendrá cada línea del archivo
de texto
```

```
k=1; % se inicializa contador de filas del archivo de texto
fecha_toma_datos=datestr(now,'dd/mm/yy HH:MM:SS'); % fecha y hora en que se
toman los datos
posicion_abonado=get(handles.id_abonado,'Value'); % opción escogida del
listado de abonados
abonado=abonados{posicion_abonado}; % #id del abonado correspondiente

% Tarifas vigentes para el operador que atiende al abonado A:
conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
consulta_sql=sprintf('select * from TARIFA where Id_Operador=%d order by
Fecha_modificacion',id_operador);
cursor=exec(conexion,consulta_sql); % Busca las tarifas del operador
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
datos=cursor.Data; % datos es un arreglo de celdas
arreglo_tarifa=datos(end,:); % Toma las tarifas con la fecha más reciente
close(conexion); % Se desconecta de la base de datos

% Escoger archivo
[nombre_archivo,ruta]=uigetfile('*.txt','Escoja un archivo txt:');
nombre=[ruta,nombre_archivo];

if ischar(nombre) % si se escoge un archivo
   id_archivo=fopen(nombre,'r'); % abre archivo para leerlo
   if id_archivo==-1 % si hay error, da mensaje de error
       texto=sprintf('El archivo "%s" no pudo abrirse para
lectura.',nombre);
       msgbox(texto,'Error','error');
   else % si no hay error
       set(handles.texto_ruta,'String',nombre); % muestra la ruta completa
del archivo escogido

       while ~feof(id_archivo) % se ejecuta el bucle o lazo mientras no
llegue al final del archivo de texto
           linea=fgetl(id_archivo); % lee línea por línea y la agrega al
cell array arreglo_llamada
           espacio_blanco=find(isspace(linea)); % busca los espacios en
blanco que separan las columnas de datos. Es un vector.
           arreglo_llamada{k,1}=fecha_toma_datos; % agrega al cell array la
fecha y hora de toma de datos
           arreglo_llamada{k,2}=abonado; % agrega al cell array el #id del
abonado A
           arreglo_llamada{k,3}=telefono; % agrega al cell array el teléfono
del abonado A
           arreglo_llamada{k,4}=id_operador; % agrega al cell array el id
del operador que presta servicio al abonado A
           arreglo_llamada{k,5}=get(handles.nombre_operador,'String'); %
agrega al cell array el operador que da servicio al abonado A
           arreglo_llamada{k,6}=linea(1:espacio_blanco(1)-1); % agrega al
cell array el teléfono del abonado B

arreglo_llamada{k,8}=[linea(espacio_blanco(2)+1:espacio_blanco(3)-
1),linea(espacio_blanco(1):espacio_blanco(2)-1)]; % agrega al cell array la
fecha y hora de inicio de la llamada

arreglo_llamada{k,9}=[linea(espacio_blanco(4)+1:end),linea(espacio_blanco(3)
```

```
:espacio_blanco(4)-1)]; % agrega al cell array la fecha y hora de
finalización de la llamada
            k=k+1;
        end % fin de for de leer archivo txt

        [f,c]=size(arreglo_lotes); % series numéricas en el plan de
numeración

        % Servicio telefónico del abonado A:
        for n=1:f % para cada lote o serie numérica del plan de numeración
            if telefono>=matriz_lotes(n,1) && telefono<=matriz_lotes(n,2) %
si telefono de abonado A está contenido dentro de un lote
                conexion=database('ODBCTasador','',''); % Se conecta a la
base de datos
                query_sql=sprintf('select Tipo from PLAN_NUMERACION where
Lote_Inicio=%d',matriz_lotes(n,1));
                cursor=exec(conexion,query_sql); % Busca el nombre del
operador
                cursor=fetch(cursor); % recorre los datos de la lista uno por
uno
                datos=cursor.Data; % datos es un arreglo de celdas
                close(conexion); % Se desconecta de la base de datos
                servicio_A=datos{1}; % Servicio telefónico del abonado A
                break; % si encuentra el servicio entonces detiene el bucle
            end
        end

        % Si el teléfono B es fijo entonces el número se acota a 8 dígitos:
        h=waitbar(0,'Procesando data log de llamadas... 0%');
        [filas,col]=size(arreglo_llamada);
        for k=1:filas
            tel_B=arreglo_llamada{k,6}; % teléfono del abonado B
            if tel_B(1)=='2' && length(tel_B)>8 % si es número fijo y se ha
capturado un dígito extra por error
                tel_B=tel_B(1:8); % entonces lo acota a 8 dígitos
            end
            arreglo_llamada{k,6}=tel_B; % guarda el teléfono del abonado B
corregido, como texto
            arreglo_llamada{k,7}=str2double(tel_B); % guarda el teléfono del
abonado B corregido, como tipo numérico

            % Detecta el operador al que pertenece el teléfono B y el tipo de
servicio:
            tel_B=arreglo_llamada{k,7};
            for n=1:f % para cada serie numérica del plan de numeración
                if tel_B>=matriz_lotes(n,1) && tel_B<=matriz_lotes(n,2) % si
tel_B está contenido dentro de un lote
                    arreglo_llamada{k,10}=arreglo_lotes{n,4}; % entonces toma
el id del operador al que se asignó dicho lote ( id operador B)
                    conexion=database('ODBCTasador','',''); % Se conecta a la
base de datos
                    query_sql=sprintf('select Nombre from OPERADOR where
Id=%d',arreglo_llamada{k,10});
                    cursor=exec(conexion,query_sql); % Busca el nombre del
operador
```

```
                    cursor=fetch(cursor); % recorre los datos de la lista uno
por uno
                    datos=cursor.Data; % datos es un arreglo de celdas
                    close(conexion); % Se desconecta de la base de datos
                    arreglo_llamada{k,11}=datos{1}; % se obtiene el nombre
del operador B
                    arreglo_llamada{k,12}=arreglo_lotes{n,3}; % y el tipo de
servicio de ese lote o serie numérica
                    % no es posible llamar a un NCSP, por lo que la llamada
                    % se tomará como PBX:
                    if strcmp(arreglo_llamada{k,12},'Asignación de NSPC/
canal común No.7 (CCS7)')
                        arreglo_llamada{k,11}='PBX'; % entonces es llamada a
la misma PBX
                        arreglo_llamada{k,12}=servicio_A;
                    end
                    break; % si encuenta el operador entonces detiene el
bucle
                else % si el teléfono B no está en el plan de numeración
                    if tel_B<1e8 % si el teléfono B tiene menos de 8 dígitos
                        arreglo_llamada{k,11}='PBX'; % entonces es llamada a
la misma PBX
                        arreglo_llamada{k,12}=servicio_A;
                    else % si no, es internacional
                        arreglo_llamada{k,11}='INTERNACIONAL'; % entonces es
llamada internacional
                        arreglo_llamada{k,12}='Servicio de sistema
Multiportador'; % tipo de servicio
                    end % fin de if de tel B de 8 dígitos
                end % fin de if de tel B dentro del plan de numeración
            end % fin de for para cada lote

            % Duración de cada llamada:
            inicio=datenum(arreglo_llamada{k,8},'dd/mm/yy HH:MM:SS'); % incio
de llamada
            fin=datenum(arreglo_llamada{k,9},'dd/mm/yy HH:MM:SS'); % fin de
llamada
            duracion=fin-inicio; % resta fin menos inicio
            duracion=datestr(duracion,'HH:MM:SS'); % lo pasa a formato
horas:minutos:segundos
            arreglo_llamada{k,13}=duracion; % duración de la llamada
            inicio_llamada=arreglo_llamada{k,8}; % tipo texto
            fin_llamada=arreglo_llamada{k,9}; % tipo texto
            h_pleno=[inicio_llamada(1:9),arreglo_tarifa{3},':00']; % tipo
texto
            h_reducido=[inicio_llamada(1:9),arreglo_tarifa{4},':00']; % tipo
texto
            inicio_llamada_num=datenum(inicio_llamada,'dd/mm/yy HH:MM:SS'); %
tipo numérico
            fin_llamada_num=datenum(fin_llamada,'dd/mm/yy HH:MM:SS'); % tipo
numérico
            h_pleno_num=datenum(h_pleno,'dd/mm/yy HH:MM:SS'); % tipo numérico
            h_reducido_num=datenum(h_reducido,'dd/mm/yy HH:MM:SS'); % tipo
numérico
```

```
            if inicio_llamada_num>=h_pleno_num &&
fin_llamada_num<=h_reducido_num % si toda la llamada cae en horario de
tarifa plena

duracion_plena=str2double(duracion(1:2))*60+str2double(duracion(4:5))+str2do
uble(duracion(7:8))/60; % duración en minutos
                duracion_reducida=0;
            elseif (inicio_llamada_num<h_pleno_num &&
fin_llamada_num<=h_pleno_num) || inicio_llamada_num>=h_reducido_num % si
toda la llamada cae en horario de tarifa reducida

duracion_reducida=str2double(duracion(1:2))*60+str2double(duracion(4:5))+str
2double(duracion(7:8))/60; % duración en minutos
                duracion_plena=0;
            elseif inicio_llamada_num<h_pleno_num &&
fin_llamada_num>h_pleno_num % si la llamada cae una parte en horario pleno y
otra parte en horario reducido, pasando por el inicio de horario pleno
                duracion_reducida=datestr(h_pleno_num-
inicio_llamada_num,'HH:MM:SS'); % en horas:min:seg

duracion_reducida=str2double(duracion_reducida(1:2))*60+str2double(duracion_
reducida(4:5))+str2double(duracion_reducida(7:8))/60; % en minutos
                duracion_plena=datestr(fin_llamada_num-
h_pleno_num,'HH:MM:SS'); % en horas:min:seg

duracion_plena=str2double(duracion_plena(1:2))*60+str2double(duracion_plena(
4:5))+str2double(duracion_plena(7:8))/60; % en minutos
            elseif inicio_llamada_num<h_reducido_num &&
fin_llamada_num>h_reducido_num % si la llamada cae una parte en horario
pleno y otra parte en horario reducido, pasando por el inicio de horario
reducido
                duracion_reducida=datestr(fin_llamada_num-
h_reducido_num,'HH:MM:SS'); % en horas:min:seg

duracion_reducida=str2double(duracion_reducida(1:2))*60+str2double(duracion_
reducida(4:5))+str2double(duracion_reducida(7:8))/60; % en minutos
                duracion_plena=datestr(h_reducido_num-
inicio_llamada_num,'HH:MM:SS'); % en horas:min:seg

duracion_plena=str2double(duracion_plena(1:2))*60+str2double(duracion_plena(
4:5))+str2double(duracion_plena(7:8))/60; % en minutos
            end

        % cobro para cada llamada. Se toma en cuenta la duración, el tipo de
llamada, el horario de inicio.
        % Limitante: Si es internacional se toma como destino Estados Unidos
        %for k=1:filas
           switch arreglo_llamada{k,12}
               case 'Servicio especial  '
                   arreglo_llamada{k,14}='SERVICIO ESPECIAL';
                   arreglo_llamada{k,15}=0;
               case 'Servicio de larga distancia internacional por operadora
de cobro revertido'
                   arreglo_llamada{k,14}='INTERNACIONAL SIN COBRO';
                   arreglo_llamada{k,15}=0;
```

```
            case 'Servicio telefónico automático de cobro revertido
nacional'
                arreglo_llamada{k,14}='NACIONAL SIN COBRO';
                arreglo_llamada{k,15}=0;
            case 'Servicio telefónico automático de cobro revertido
internacional'
                arreglo_llamada{k,14}='INTERNACIONAL SIN COBRO';
                arreglo_llamada{k,15}=0;
            case 'Servicio de llamadas a diferentes destinos en forma
semiautomática a través de sistemas prepagados'
                arreglo_llamada{k,14}='INTERNACIONAL PREPAGADA';

arreglo_llamada{k,15}=arreglo_tarifa{9}*(duracion_plena+duracion_reducida);
            case 'Servicio telefónico automático con terminacion
internacional con sobrecuota para el abonado A'
                arreglo_llamada{k,14}='INTERNACIONAL CON SOBRECUOTA';

arreglo_llamada{k,15}=1.5*ceil(duracion_plena+duracion_reducida);
            case 'Servicio telefónico automático con terminacion nacional
con sobrecuota para el abonado A'
                arreglo_llamada{k,14}='NACIONAL CON SOBRECUOTA';

arreglo_llamada{k,15}=1.5*ceil(duracion_plena+duracion_reducida);
            case 'Servicio de telefonía móvil'
                arreglo_llamada{k,14}='A CELULAR';

arreglo_llamada{k,15}=arreglo_tarifa{10}*(duracion_plena+duracion_reducida);
            case 'Servicio de sistema Multiportador'
                arreglo_llamada{k,14}='INTERNACIONAL';

arreglo_llamada{k,15}=arreglo_tarifa{9}*(duracion_plena+duracion_reducida);
            otherwise
                if strcmp(servicio_A,arreglo_llamada{k,12}) % si es la
misma zona geográfica
                    arreglo_llamada{k,14}='LOCAL';
                    arreglo_llamada{k,15}=arreglo_tarifa{5}*duracion_plena
+ arreglo_tarifa{7}*duracion_reducida;
                else
                    arreglo_llamada{k,14}='NACIONAL';
                    arreglo_llamada{k,15}=arreglo_tarifa{6}*duracion_plena
+ arreglo_tarifa{8}*duracion_reducida;
                end % fin de if de zona geográfica
        end % fin de switch
         waitbar(k/filas,h,sprintf('Procesando data log de llamadas...
%d%%',round(k/filas*100))); % ventana de barra de progreso
        cobro_total=cobro_total+arreglo_llamada{k,15};
    end

    set(handles.cobro,'String',num2str(cobro_total)); % Muestra el cobro
total de las llamadas
    set(handles.guardar,'Enable','on'); % habilita botón de guardar en
base de datos
    set(handles.reporte,'Enable','on'); % habilita botón de reporte en
Excel
```

```
        llamada={}; % arreglo de celdas que se guardará en la base de datos
        llamada(:,1:4)=arreglo_llamada(:,1:4);
        llamada(:,5)=arreglo_llamada(:,6);
        llamada(:,6)=arreglo_llamada(:,11);
        llamada(:,7:8)=arreglo_llamada(:,8:9);
        llamada(:,9)=arreglo_llamada(:,14);
        llamada(:,10)=arreglo_llamada(:,13);
        llamada(:,11)=arreglo_llamada(:,15);
        close(h); % cierra la ventana de la barra de progreso

    end % fin de if de error de lectura
    fclose(id_archivo); % cierra el archivo de texto
end % fin de if de escoger archivo

% --------------------------------------------------------------------

function guardar_Callback(hObject, eventdata, handles)
% Botón para guardar en la base de datos la información extraída del data
log

global llamada

[filas,col]=size(llamada);
conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
for k=1:filas

insert(conexion,'LLAMADAS',{'Toma_datos','Abonado','Telefono_A','Operador_A'
,'Telefono_B','Operador_B','Inicio_llamada','Fin_llamada','Tipo_llamada','Du
racion_llamada','Tarifa'},...

{llamada{k,1},llamada{k,2},llamada{k,3},llamada{k,4},llamada{k,5},llamada{k,
6},llamada{k,7},llamada{k,8},llamada{k,9},llamada{k,10},llamada{k,11}});
end
msgbox('Ha agregado un bloque de llamadas telefónicas a la base de
datos.','Aviso','warn');
close(conexion); % Se desconecta de la base de datos

% --------------------------------------------------------------------

% --- Executes on button press in reporte.
function reporte_Callback(hObject, eventdata, handles)
% Botón para generar reporte de para la información extraída del data
% log de llamadas

global llamada abonado telefono

conexion=database('ODBCTasador','',''); % Se conecta a la base de datos
cursor=exec(conexion,sprintf('select
Nombre,Apellido,Tipo_identificacion,Direccion from Abonado where
Identificacion=''%s''',abonado)); % Busca los datos del abonado
cursor=fetch(cursor); % recorre los datos de la lista uno por uno
datos_abonado=cursor.Data; % datos_abonado es un arreglo de celdas
close(conexion); % Se desconecta de la base de datos
```

```
operador=get(handles.nombre_operador,'String');
fecha=datestr(now,'dd-mm-yyyy'); % fecha de descarga de datos
nombre_reporte=['C:\Reportes\Llamadas_abonado_',abonado,'_',fecha,'.htm']; %
nombre del archivo de reporte
id_reporte=fopen(nombre_reporte,'w'); % permiso para escribir
if id_reporte==-1
    error(sprintf('El archivo "%s" no pudo abrirse para
escritura.',nombre_reporte))
else

    [filas,col]=size(llamada);
    titulo=sprintf('Reporte de llamadas de telefonía fija para el abonado %s
(fecha %s)',abonado,fecha);
    texto=sprintf('<html><head><title>%s</title></head>',titulo); % se
inicializa el bloque de texto que contendrá el código fuente del reporte
html
    texto=[texto,'<body><p align="center"><img border="0"
src="file:///C:/Tasación de telefonía fija/Siget1.PNG" width="225"
height="80"></p>'];
    texto=[texto,sprintf('<p align="center"><b><font size="4"
color="#0000FF">%s</font></b></p>',titulo)];
    texto=[texto,sprintf('<p>Nombre: %s
%s<br>',datos_abonado{1},datos_abonado{2})];
    texto=[texto,sprintf('Identificación: %s
%s<br>',datos_abonado{3},abonado)];
    texto=[texto,sprintf('Dirección: %s<br>',datos_abonado{4})];
    texto=[texto,sprintf('Teléfono: %d<br>',telefono)];
    texto=[texto,sprintf('Operador: %s<br>',operador)];
    texto=[texto,sprintf('Cobro: US$ %s</p>',get(handles.cobro,'String'))];
    texto=[texto,'<hr>'];

    % Conteo de llamadas por tipo
    local=0; % total de llamadas locales
    nacional=0; %total de llamadas nacionales
    inter=0; % total de llamadas nacionales sin cobro
    especial=0; % total de llamadas de servicio especial
    a_celular=0; % total de llamadas a celular
    min_local=0; % total de minutos de llamadas locales
    min_nacional=0; %total de minutos de llamadas nacionales
    min_inter=0; % total de minutos de llamadas nacionales sin cobro
    min_especial=0; % total de minutos de llamadas de servicio especial
    min_a_celular=0; % total de minutos de llamadas a celular

    % Conteo de llamadas por mismo operador
    mismo_operador=0;
    min_mismo_operador=0; % total de minutos al mismo operador
    diferente_operador=0;
    min_diferente_operador=0; % total de minutos a diferente operador

    h=waitbar(0,'Resumen por tipo de llamada... 0%'); % Inicio de barra de
progreso
    for k=1:filas
        switch llamada{k,9} % tipo de llamada
            case 'LOCAL'
                local=local+1;
```

```
                min_local(end+1)=datenum(llamada{k,10});
            case {'NACIONAL','NACIONAL SIN COBRO','NACIONAL CON SOBRECUOTA'}
                nacional=nacional+1;
                min_nacional(end+1)=datenum(llamada{k,10});
            case {'INTERNACIONAL','INTERNACIONAL SIN COBRO','INTERNACIONAL
PREPAGADA','INTERNACIONAL CON SOBRECUOTA'}
                inter=inter+1;
                min_inter(end+1)=datenum(llamada{k,10});
            case 'SERVICIO ESPECIAL'
                especial=especial+1;
                min_especial(end+1)=datenum(llamada{k,10});
            case 'A CELULAR'
                a_celular=a_celular+1;
                min_a_celular(end+1)=datenum(llamada{k,10});
        end
        % Mismo operador o diferente
        if strcmp(operador,llamada{k,6})
            mismo_operador=mismo_operador+1;
            min_mismo_operador(end+1)=datenum(llamada{k,10});
        else
            diferente_operador=diferente_operador+1;
            min_diferente_operador(end+1)=datenum(llamada{k,10});
        end
        waitbar(k/filas,h,sprintf('Resumen por tipo de llamada...
%d%%',round(k/filas*100))); % ventana de barra de progreso
    end
    close(h); % cierra la ventana de la barra de progreso

    texto=[texto,'<ul><li><p align="center"><b><span lang="es"><font
size="4">RESUMEN</font></span></b></p></li></ul>'];
    texto=[texto,sprintf('<p>Total de llamadas: %d llamadas, tiempo total
(hh:mm:ss)
%s</p>',filas,datestr(sum(min_mismo_operador)+sum(min_diferente_operador),'H
H:MM:SS'))];
    texto=[texto,sprintf('<p>Llamadas locales: %d llamadas,
%s<br>',local,datestr(sum(min_local),'HH:MM:SS'))];
    texto=[texto,sprintf('Llamadas nacionales: %d llamadas,
%s<br>',nacional,datestr(sum(min_nacional),'HH:MM:SS'))];
    texto=[texto,sprintf('Llamadas internacionales: %d llamadas,
%s<br>',inter,datestr(sum(min_inter),'HH:MM:SS'))];
    texto=[texto,sprintf('Llamadas a celular: %d llamadas,
%s<br>',a_celular,datestr(sum(min_a_celular),'HH:MM:SS'))];
    texto=[texto,sprintf('Llamadas a servicio especial: %d llamadas,
%s</p>',especial,datestr(sum(min_especial),'HH:MM:SS'))];

    % Gráfica de pastel
    pastel=[local,nacional,inter,a_celular,especial]+0.001;
    figure; % nueva ventana
    pie(pastel); % gráfica de pastel
    legend('Local','Nacional','Internacional','A celular','Serv.
especial','Location','NorthEastOutside');
    ruta='C:\Reportes\Gráficos\Pastel1';
    nombre_grafica=sprintf('%s_%s_%s',ruta,abonado,fecha);
    print('-f1','-dpng',nombre_grafica); % guarda la imagen de la gráfica de
pastel
```

```
    close(1); % cierra la nueva ventana
    texto=[texto,sprintf('<p align="center"><img border="0" src="%s.PNG"
width="800" height="600"></p>',nombre_grafica)];

    texto=[texto,sprintf('<p>Llamadas al mismo operador: %d llamadas,
%s<br>',mismo_operador,datestr(sum(min_mismo_operador),'HH:MM:SS'))];
    texto=[texto,sprintf('Llamadas a diferente operador: %d llamadas,
%s</p>',diferente_operador,datestr(sum(min_diferente_operador),'HH:MM:SS'))]
;

    pastel=[mismo_operador,diferente_operador];
    figure; % nueva ventana
    pie(pastel); % gráfica de pastel
    legend(operador,'Otro operador');
    ruta='C:\Reportes\Gráficos\Pastel2';
    nombre_grafica=sprintf('%s_%s_%s',ruta,abonado,fecha);
    print('-f1','-dpng',nombre_grafica); % guarda la imagen de la gráfica de
pastel
    close(1);
    texto=[texto,sprintf('<p align="center"><img border="0" src="%s.PNG"
width="800" height="600"></p>',nombre_grafica)];

    texto=[texto,'<hr>'];

    texto=[texto,'<ul><li><p align="center"><b><span lang="es"><font
size="4">DETALLE DE CADA LLAMADA</font></span></b></p></li></ul>'];

    h=waitbar(0,'Generando reporte de cada llamada... 0%'); % Inicio de
barra de progreso

    texto=[texto,'<p> </p>'];

    texto=[texto,'<p align="center"><table style="width:110%;" border="3"
cellpadding="0" cellspacing="0" style="border-collapse: collapse"
bordercolor="#111111" width="74%" height="118">'];
    texto=[texto,'<tr>'];
    texto=[texto,sprintf('<td width="15%%" align="center" bgcolor="#FFFFCC"
height="50">Número llamado</td>')];
    texto=[texto,sprintf('<td width="15%%" align="center" bgcolor="#FFFFCC"
height="50">Operador</td>')];
    texto=[texto,sprintf('<td width="10%%" align="center" bgcolor="#FFFFCC"
height="50">Inicio de llamada</td>')];
    texto=[texto,sprintf('<td width="10%%" align="center" bgcolor="#FFFFCC"
height="50">Fin de llamada</td>')];
    texto=[texto,sprintf('<td width="30%%" align="center" bgcolor="#FFFFCC"
height="50">Tipo de llamada</td>')];
    texto=[texto,sprintf('<td width="10%%" align="center" bgcolor="#FFFFCC"
height="50">Duración (hh:mm:ss)</td>')];
    texto=[texto,sprintf('<td width="20%%" align="center" bgcolor="#FFFFCC"
height="50">Costo en US$ (incluye IVA)</td>')];
    texto=[texto,'</tr>'];


    for k=2:filas % para cada llamada
```

```
        texto=[texto,'<tr>'];
        texto=[texto,sprintf('<td width="15%%" align="center"
height="70">%s</td>',llamada{k,5})];
        texto=[texto,sprintf('<td width="15%%" align="center"
height="70">%s</td>',llamada{k,6})];
        texto=[texto,sprintf('<td width="10%%" align="center"
height="70">%s</td>',llamada{k,7})];
        texto=[texto,sprintf('<td width="10%%" align="center"
height="70">%s</td>',llamada{k,8})];
        texto=[texto,sprintf('<td width="30%%" align="center"
height="70">%s</td>',llamada{k,9})];
        texto=[texto,sprintf('<td width="10%%" align="center"
height="70">%s</td>',llamada{k,10})];
        texto=[texto,sprintf('<td width="20%%" align="center"
height="70">%f</td>',llamada{k,11})];
        texto=[texto,'</tr>'];

        waitbar(k/filas,h,sprintf('Generando reporte de cada llamada...
%d%%',round(k/filas*100))); % ventana de barra de progreso
    end
    texto=[texto,'</table></p>']; % termina la tabla de todas las llamadas

    texto=[texto,'</body></html>'];

    fwrite(id_reporte,texto,'uchar'); % se escribe el archivo nuevo
    fclose(id_reporte); % cierra el archivo que se acaba de escribir
    close(h); % cierra la ventana de la barra de progreso
    uiwait(msgbox('El reporte que se presenta a continuación está almacenado
en C:\Reportes','Aviso','help'));
    pause(0.5); % pausa de 1/2 segundo
    eval(sprintf('web file:///%s -browser',nombre_reporte)); % abre el
reporte en página web
end

% --------------------------------------------------------------------

% --- Executes on button press in volver.
function volver_Callback(hObject, eventdata, handles)
% Botón para volver al menún principal

menu_tasador;

% ************************************************************************
% ********************** FUNCIONES AUXILIARES ****************************
% ************************************************************************

function numero=convertir_num(s,id,handles)
% Función que convierte un texto en número

if isempty(str2num(s)) || str2num(s)>9  % si está en blanco o es letra o es
número mayor que 9
    uiwait(msgbox('Debe digitar un número positivo entre 0 y 9. El carácter
inválido que usted digitó se sustituirá por un cero. Por favor
corrija.','Adevertencia','warn'));
```

```
    numero=0;
    for k=1:20 % cambia de color

eval(sprintf('set(handles.edit%d,''BackgroundColor'',[%d/20,0.5,0.5])',id,k)
);
        pause(0.05);
    end
    eval(sprintf('set(handles.edit%d,''UserData'',0)',id)); % indica que hay
error
else % si se digitó un número, lo convierte en número
    if str2num(s)<0 % si es negativo
        uiwait(msgbox('Ha digitado un número negativo, y éste se sustituirá
por uno positivo para que usted ya no tenga que corregir.','Aviso','warn'));
        numero=abs(str2num(s));
        for k=1:20 % cambia de color

eval(sprintf('set(handles.edit%d,''BackgroundColor'',[0,%d/20,0])',id,k));
            pause(0.05);
        end
        eval(sprintf('set(handles.edit%d,''BackgroundColor'',[1,1,1])',id));
% vuelve a fondo blanco
        eval(sprintf('set(handles.edit%d,''UserData'',1)',id)); % indica que
no hay error
    else % si no es negativo
        numero=abs(str2num(s));
        eval(sprintf('set(handles.edit%d,''BackgroundColor'',[1,1,1])',id));
        eval(sprintf('set(handles.edit%d,''UserData'',1)',id)); % indica que
no hay error
    end
end
```

REPORTES

```
function varargout = reportes(varargin)
% REPORTES M-file for reportes.fig
%      REPORTES, by itself, creates a new REPORTES or raises the existing
%      singleton*.
%
%      H = REPORTES returns the handle to a new REPORTES or the handle to
%      the existing singleton*.
%
%      REPORTES('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in REPORTES.M with the given input arguments.
%
%      REPORTES('Property','Value',...) creates a new REPORTES or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before reportes_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to reportes_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help reportes

% Last Modified by GUIDE v2.5 08-Feb-2010 23:56:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @reportes_OpeningFcn, ...
                   'gui_OutputFcn',  @reportes_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before reportes is made visible.
function reportes_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to reportes (see VARARGIN)

% Choose default command line output for reportes
handles.output = hObject;

% Agrega una imagen
axes(handles.axes1);
[I,map] = imread('siget1','png');
h1 = image(I);
set(handles.axes1,'Xtick',[]);
set(handles.axes1,'Ytick',[]);
colormap(map);
setappdata(0,'h',h1);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes reportes wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = reportes_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% ************************************************************************
% ************************ POPUP MENUES **********************************

function desde_dia_Callback(hObject, eventdata, handles)


function desde_dia_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------------------------------------------------------------------------

function desde_mes_Callback(hObject, eventdata, handles)
mes=get(handles.desde_mes,'Value');

% Pone 28, 30, o 31 días dependiendo del mes
switch mes
    case {1,3,5,7,8,10,12}

set(handles.desde_dia,'String',{'1','2','3','4','5','6','7','8','9',...

'10','11','12','13','14','15','16','17','18','19','20','21','22',...
            '23','24','25','26','27','28','29','30','31'});
    case {4,6,9,11}

set(handles.desde_dia,'String',{'1','2','3','4','5','6','7','8','9',...

'10','11','12','13','14','15','16','17','18','19','20','21','22',...
            '23','24','25','26','27','28','29','30'});
    otherwise

set(handles.desde_dia,'String',{'1','2','3','4','5','6','7','8','9',...

'10','11','12','13','14','15','16','17','18','19','20','21','22',...
            '23','24','25','26','27','28'});
end

set(handles.desde_dia,'Value',1); % regresa al primer día
```

```
function desde_mes_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------

function hasta_dia_Callback(hObject, eventdata, handles)


function hasta_dia_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --------------------------------------------------------------------

function hasta_mes_Callback(hObject, eventdata, handles)
mes=get(handles.hasta_mes,'Value');

% Pone 28, 30, o 31 días dependiendo del mes
switch mes
    case {1,3,5,7,8,10,12}

set(handles.hasta_dia,'String',{'1','2','3','4','5','6','7','8','9',...

'10','11','12','13','14','15','16','17','18','19','20','21','22',...
            '23','24','25','26','27','28','29','30','31'});
    case {4,6,9,11}

set(handles.hasta_dia,'String',{'1','2','3','4','5','6','7','8','9',...

'10','11','12','13','14','15','16','17','18','19','20','21','22',...
            '23','24','25','26','27','28','29','30'});
    otherwise

set(handles.hasta_dia,'String',{'1','2','3','4','5','6','7','8','9',...

'10','11','12','13','14','15','16','17','18','19','20','21','22',...
            '23','24','25','26','27','28'});
end

set(handles.hasta_dia,'Value',1); % regresa al primer día

function hasta_mes_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --------------------------------------------------------------------

function desde_ano_Callback(hObject, eventdata, handles)
% Popup menu de inicio de intervalo de tiempo


% --- Executes during object creation, after setting all properties.
function desde_ano_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------



function hasta_ano_Callback(hObject, eventdata, handles)
% Popup menu de final de intervalo de tiempo

% --- Executes during object creation, after setting all properties.
function hasta_ano_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------



% ************************************************************************
% ************************ BOTONES ***************************************

function Volver_Callback(hObject, eventdata, handles)
% Botón para regresar al menú principal
menu_tasador;

% --------------------------------------------------------------------

% --- Executes on button press in aceptar.
function aceptar_Callback(hObject, eventdata, handles)
% Botón Aceptar

desde_ano=get(handles.desde_ano,'Value')+1997; % inicio de intervalo de
tiempo
desde_dia=get(handles.desde_dia,'Value');
desde_mes=get(handles.desde_mes,'Value');
hasta_ano=get(handles.hasta_ano,'Value')+1997; % fin de intervalo de tiempo
hasta_dia=get(handles.hasta_dia,'Value');
hasta_mes=get(handles.hasta_mes,'Value');
```

```
desde=sprintf('%d-%d-%d',desde_dia,desde_mes,desde_ano);
desde_num=datenum(desde,'dd-mm-yyyy');

hasta=sprintf('%d-%d-%d',hasta_dia,hasta_mes,hasta_ano);
hasta_num=datenum(hasta,'dd-mm-yyyy');


% Tipo de reporte solicitado
if get(handles.radio_tarifas,'Value') % si se escoge la variación de tarifas
    tipo_reporte='Tarifas';
else tipo_reporte='Llamadas';
end

if desde_num>hasta_num % si el inicio es mayor que el final
    msgbox('El final del intervalo de tiempo es menor que el inicio. Por
favor corrija.','Error','error'); % mensaje de error
else % se procede a generar el reporte solicitado
    switch(tipo_reporte)
        case 'Tarifas' % REPORTE DE TARIFAS
            conexion=database('ODBCTasador','',''); % Se conecta a la base
de datos
            cursor=exec(conexion,'select distinct Id_Operador,Nombre from
Tarifa inner join Operador on Tarifa.Id_Operador=Operador.Id order by
Nombre'); % Busca los nombres de los operadores
            cursor=fetch(cursor); % recorre los datos de la lista uno por
uno
            arreglo_operador=cursor.Data; % arreglo de celdas
            close(conexion); % Se desconecta de la base de datos

            fecha=datestr(now,'dd-mm-yyyy'); % fecha actual
            nombre_reporte=['C:\Reportes\Variación_Tarifas_',desde,'--
',hasta,'__(',fecha,')','.htm'];
            id_reporte=fopen(nombre_reporte,'w');
            if id_reporte==-1
                error(sprintf('El archivo "%s" no pudo abrirse para
escritura.',nombre_reporte));
            else
                [filas,col]=size(arreglo_operador);
                titulo=sprintf('Reporte de tarifas de telefonía fija de
todos los operadores desde %s hasta %s',desde,hasta);

texto=sprintf('<html><head><title>%s</title></head>',titulo); % se
inicializa el bloque de texto que contendrá el código fuente del reporte
html
                texto=[texto,'<body><p align="center"><img border="0"
src="file:///C:/Tasación de telefonía fija/Siget1.PNG" width="225"
height="80"></p>'];
                texto=[texto,sprintf('<p align="center"><b><font size="4"
color="#0000FF">%s</font></b></p>',titulo)];
                texto=[texto,'<p align="left"><span lang="es">NOTA: las
tarifas ya incluyen IVA</span></p>'];
                h=waitbar(0,'Generando reporte... 0%');

                for k=1:filas % para cada operador
```

```
                    texto=[texto,'<p> </p>'];

                    conexion=database('ODBCTasador','','');  % Se conecta a
la base de datos
                    nombre=arreglo_operador{k,2};
                    texto=[texto,sprintf('<ul><li><p align="center"><b><span
lang="es"><font size="4">%s</font></span></b></p></li></ul>',nombre)];
                    cursor=exec(conexion,sprintf('select * from Tarifa where
Id_Operador=%d and Fecha_modificacion between #%s# and #%s# order by
Fecha_modificacion',arreglo_operador{k,1},desde,hasta));  % Tarifas del
operador
                    cursor=fetch(cursor);  % recorre los datos de la lista
uno por uno
                    datos_tarifa=cursor.Data;  % datos_tarifa es un arreglo
de celdas que almacena el resultado de la consulta SQL
                    close(conexion);  % Se desconecta de la base de datos

                    if strcmp(datos_tarifa,'No Data')  % si hay datos hace la
tabla, en caso contrario sólo deja encabezados
                        texto=[texto,'<p align="center">No hay datos</p>'];
                    else
                        texto=[texto,'<p align="center"><table border="3"
cellpadding="0" cellspacing="0" style="border-collapse: collapse"
bordercolor="#111111" width="74%" height="118">'];
                        texto=[texto,'<tr>'];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Fecha de
modificación</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Inicio de horario
pleno</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Inicio de horario
reducido</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Tarifa local
plena</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Tarifa nacional
plena</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Tarifa local
reducida</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Tarifa nacional
reducida</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Tarifa
internacional</span></td>')];
                        texto=[texto,sprintf('<td width="11.1%%"
align="center" bgcolor="#FFFFCC" height="50"><span>Tarifa a
celular</span></td>')];
                        texto=[texto,'</tr>'];

                        [fil,col]=size(datos_tarifa);
```

```
                        for n=1:fil % para cada registro obtenido en la
consulta SQL
                            texto=[texto,'<tr>'];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center"
height="20"><span>%s</span></td>',datestr(datos_tarifa{n,2},'dd-mm-yyyy
HH:MM:SS'))];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%s</span></td>',datos_tarifa{n,3})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%s</span></td>',datos_tarifa{n,4})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%f</span></td>',datos_tarifa{n,5})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%f</span></td>',datos_tarifa{n,6})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%f</span></td>',datos_tarifa{n,7})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%f</span></td>',datos_tarifa{n,8})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%f</span></td>',datos_tarifa{n,9})];
                            texto=[texto,sprintf('<td width="11.1%%"
align="center" height="20"><span>%f</span></td>',datos_tarifa{n,10})];
                            texto=[texto,'</tr>'];
                        end
                    end
                    texto=[texto,'</table></p>']; % termina la tabla para el
operador
                    waitbar(k/filas,h,sprintf('Generando reporte...
%d%%',round(k/filas*100))); % ventana de barra de progreso
                end

                texto=[texto,'</body></html>'];

                fwrite(id_reporte,texto,'uchar'); % se escribe el archivo
nuevo
                fclose(id_reporte); % cierra el archivo que se acaba de
escribir
                close(h); % cierra la ventana de la barra de progreso
                uiwait(msgbox('El reporte que se presenta a continuación
está almacenado en C:\Reportes','Aviso','help'));
                pause(0.5); % pausa de 1/2 segundo
                eval(sprintf('web file:///%s -browser',nombre_reporte)); %
abre el reporte en página web
            end

        case 'Llamadas' % REPORTE DE LLAMADAS
            conexion=database('ODBCTasador','',''); % Se conecta a la base
de datos
            cursor=exec(conexion,sprintf('select
Toma_datos,Abonado,Telefono_A,Nombre,Telefono_B,Operador_B,Inicio_llamada,Fi
n_llamada,Tipo_llamada,Duracion_llamada,Tarifa from LLAMADAS inner join
OPERADOR on LLAMADAS.Operador_A=Operador.Id where Inicio_llamada between
#%s# and #%s# order by Toma_datos',desde,hasta)); % Busca todas las llamadas
efectuadas en el intervalo de tiempo
```

```
            cursor=fetch(cursor); % recorre los datos de la lista uno por
uno
            llamadas=cursor.Data; % llamadas es un arreglo de celdas
            close(conexion); % Se desconecta de la base de datos

            fecha=datestr(now,'dd-mm-yyyy'); % fecha actual
            abonado=llamadas{1,2};
            nombre_reporte=['C:\Reportes\Llamadas_Telefónicas_',desde,'--
',hasta,'__(',fecha,')','.htm'];
            id_reporte=fopen(nombre_reporte,'w');
            if id_reporte==-1
                error(sprintf('El archivo "%s" no pudo abrirse para
escritura.',nombre_reporte));
            else
                [filas,col]=size(llamadas);
                titulo=sprintf('Reporte de llamadas de telefonía fija de
todos los abonados desde %s hasta %s',desde,hasta);

texto=sprintf('<html><head><title>%s</title></head>',titulo); % se
inicializa el bloque de texto que contendrá el código fuente del reporte
html
                texto=[texto,'<body><p align="center"><img border="0"
src="file:///C:/Tasación de telefonía fija/Siget1.PNG" width="225"
height="80"></p>'];
                texto=[texto,sprintf('<p align="center"><b><font size="4"
color="#0000FF">%s</font></b></p>',titulo)];
                texto=[texto,'<p align="left"><span lang="es">NOTA: las
tarifas ya incluyen IVA</span></p>'];
                %h=waitbar(0,'Generando reporte... 0%');

                if strcmp(llamadas,'No Data') % si hay datos hace la tabla,
en caso contrario sólo da mensaje de no hay datos
                    texto=[texto,'<p align="center">No hay datos</p>'];
                else
                    % Conteo de llamadas por tipo:
                    local=0; % total de llamadas locales
                    nacional=0; %total de llamadas nacionales
                    inter=0; % total de llamadas nacionales sin cobro
                    especial=0; % total de llamadas de servicio especial
                    a_celular=0; % total de llamadas a celular
                    min_local=0; % total de minutos de llamadas locales
                    min_nacional=0; %total de minutos de llamadas nacionales
                    min_inter=0; % total de minutos de llamadas nacionales
sin cobro
                    min_especial=0; % total de minutos de llamadas de
servicio especial
                    min_a_celular=0; % total de minutos de llamadas a
celular

                    % Conteo de llamadas por mismo operador
                    mismo_operador=0;
                    min_mismo_operador=0; % total de minutos al mismo
operador
                    diferente_operador=0;
```

```
                    min_diferente_operador=0; % total de minutos a diferente
operador

                    h=waitbar(0,'Resumen por tipo de llamada... 0%'); %
Inicio de barra de progreso
                    for k=1:filas
                        operador=llamadas{1,4};
                        switch llamadas{k,9} % tipo de llamada
                            case 'LOCAL'
                                local=local+1;
                                min_local(end+1)=datenum(llamadas{k,10});
                            case {'NACIONAL','NACIONAL SIN COBRO','NACIONAL
CON SOBRECUOTA'}
                                nacional=nacional+1;
                                min_nacional(end+1)=datenum(llamadas{k,10});
                            case {'INTERNACIONAL','INTERNACIONAL SIN
COBRO','INTERNACIONAL PREPAGADA','INTERNACIONAL CON SOBRECUOTA'}
                                inter=inter+1;
                                min_inter(end+1)=datenum(llamadas{k,10});
                            case 'SERVICIO ESPECIAL'
                                especial=especial+1;
                                min_especial(end+1)=datenum(llamadas{k,10});
                            case 'A CELULAR'
                                a_celular=a_celular+1;

min_a_celular(end+1)=datenum(llamadas{k,10});
                        end
                        % Mismo operador o diferente
                        if strcmp(operador,llamadas{k,6})
                            mismo_operador=mismo_operador+1;

min_mismo_operador(end+1)=datenum(llamadas{k,10});
                        else
                            diferente_operador=diferente_operador+1;

min_diferente_operador(end+1)=datenum(llamadas{k,10});
                        end
                        waitbar(k/filas,h,sprintf('Resumen por tipo de
llamada... %d%%',round(k/filas*100))); % ventana de barra de progreso
                    end
                    close(h); % cierra la ventana de la barra de progreso

                    texto=[texto,'<ul><li><p align="center"><b><span
lang="es"><font size="4">RESUMEN</font></span></b></p></li></ul>'];
                    texto=[texto,sprintf('<p>Total de llamadas: %d llamadas,
tiempo total (hh:mm:ss)
%s</p>',filas,datestr(sum(min_mismo_operador)+sum(min_diferente_operador),'H
H:MM:SS'))];
                    texto=[texto,sprintf('<p>Llamadas locales: %d llamadas,
%s<br>',local,datestr(sum(min_local),'HH:MM:SS'))];
                    texto=[texto,sprintf('Llamadas nacionales: %d llamadas,
%s<br>',nacional,datestr(sum(min_nacional),'HH:MM:SS'))];
                    texto=[texto,sprintf('Llamadas internacionales: %d
llamadas, %s<br>',inter,datestr(sum(min_inter),'HH:MM:SS'))];
```

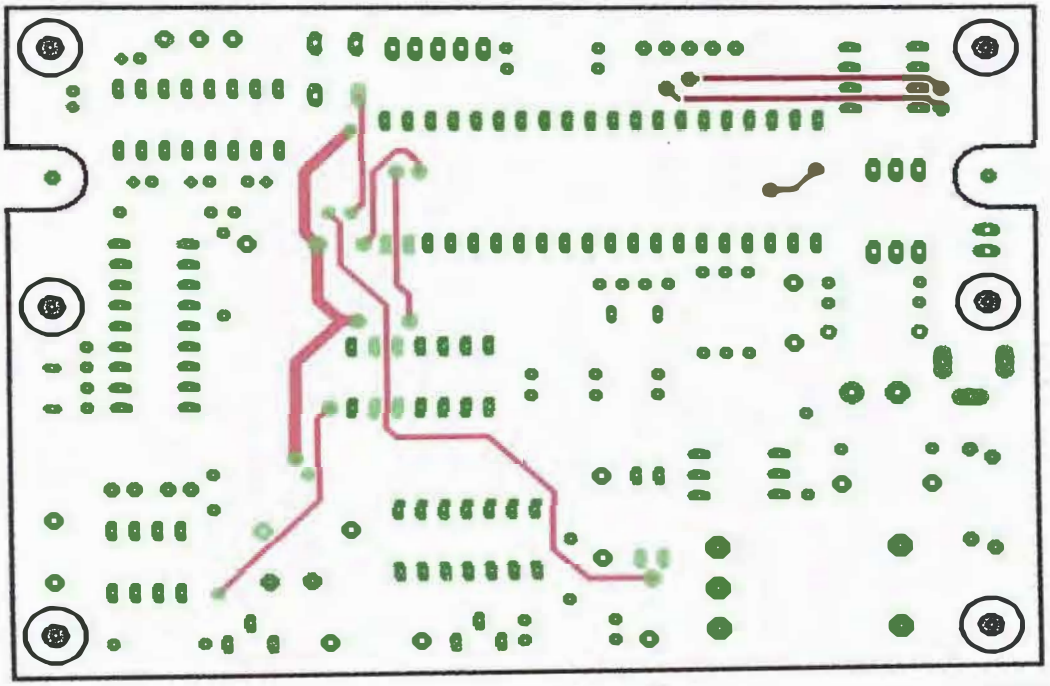
```
                    texto=[texto,sprintf('Llamadas a celular: %d llamadas,
%s<br>',a_celular,datestr(sum(min_a_celular),'HH:MM:SS'))];
                    texto=[texto,sprintf('Llamadas a servicio especial: %d
llamadas, %s</p>',especial,datestr(sum(min_especial),'HH:MM:SS'))];

                    % Gráfica de pastel
                    pastel=[local,nacional,inter,a_celular,especial]+0.001;
                    figure; % nueva ventana
                    pie(pastel); % gráfica de pastel
                    legend('Local','Nacional','Internacional','A
celular','Serv. especial','Location','NorthEastOutside');
                    ruta='C:\Reportes\Gráficos\Pastel1';
                    nombre_grafica=sprintf('%s_%s_%s',ruta,abonado,fecha);
                    print('-f1','-dpng',nombre_grafica); % guarda la imagen
de la gráfica de pastel
                    close(1); % cierra la nueva ventana
                    texto=[texto,sprintf('<p align="center"><img border="0"
src="%s.PNG" width="800" height="600"></p>',nombre_grafica)];

                    texto=[texto,sprintf('<p>Llamadas al mismo operador: %d
llamadas,
%s<br>',mismo_operador,datestr(sum(min_mismo_operador),'HH:MM:SS'))];
                    texto=[texto,sprintf('Llamadas a diferente operador: %d
llamadas,
%s</p>',diferente_operador,datestr(sum(min_diferente_operador),'HH:MM:SS'))]
;

                    pastel=[mismo_operador,diferente_operador];
                    figure; % nueva ventana
                    pie(pastel); % gráfica de pastel
                    legend(operador,'Otro operador');
                    ruta='C:\Reportes\Gráficos\Pastel2';
                    nombre_grafica=sprintf('%s_%s_%s',ruta,abonado,fecha);
                    print('-f1','-dpng',nombre_grafica); % guarda la imagen
de la gráfica de pastel
                    close(1);
                    texto=[texto,sprintf('<p align="center"><img border="0"
src="%s.PNG" width="800" height="600"></p>',nombre_grafica)];

                    texto=[texto,'<hr>'];

                    texto=[texto,'<ul><li><p align="center"><b><span
lang="es"><font size="4">DETALLE DE CADA
LLAMADA</font></span></b></p></li></ul>'];

                    h=waitbar(0,'Generando reporte de cada llamada... 0%');
% Inicio de barra de progreso

                    texto=[texto,'<p> </p>'];


                    texto=[texto,'<p align="center"><table
style="width:165%;" border="3" cellpadding="0" cellspacing="0"
```

```
style="border-collapse: collapse" bordercolor="#111111" width="74%"
height="118">'];
                    texto=[texto,'<tr>'];
                    texto=[texto,sprintf('<td width="10%%" align="center"
bgcolor="#FFFFCC" height="50">Fecha de toma de datos</td>')];
                    texto=[texto,sprintf('<td width="15%%" align="center"
bgcolor="#FFFFCC" height="50">Identificación del abonado</td>')];
                    texto=[texto,sprintf('<td width="10%%" align="center"
bgcolor="#FFFFCC" height="50">Teléfono</td>')];
                    texto=[texto,sprintf('<td width="10%%" align="center"
bgcolor="#FFFFCC" height="50">Operador</td>')];
                    texto=[texto,sprintf('<td width="15%%" align="center"
bgcolor="#FFFFCC" height="50">Número llamado</td>')];
                    texto=[texto,sprintf('<td width="15%%" align="center"
bgcolor="#FFFFCC" height="50">Operador llamado</td>')];
                    texto=[texto,sprintf('<td width="10%%" align="center"
bgcolor="#FFFFCC" height="50">Inicio de llamada</td>')];
                    texto=[texto,sprintf('<td width="10%%" align="center"
bgcolor="#FFFFCC" height="50">Fin de llamada</td>')];
                    texto=[texto,sprintf('<td width="30%%" align="center"
bgcolor="#FFFFCC" height="50">Tipo de llamada</td>')];
                    texto=[texto,sprintf('<td width="10%%" align="center"
bgcolor="#FFFFCC" height="50">Duración (hh:mm:ss)</td>')];
                    texto=[texto,sprintf('<td width="30%%" align="center"
bgcolor="#FFFFCC" height="50">Costo en US$ (incluye IVA)</td>')];
                    texto=[texto,'</tr>'];

                  for k=1:filas % para cada llamada
                      texto=[texto,'<tr>'];
                      texto=[texto,sprintf('<td width="10%%"
align="center" height="70">%s</td>',datestr(llamadas{k,1},'dd-mm-yyyy
HH:MM:SS'))];
                      texto=[texto,sprintf('<td width="15%%"
align="center" height="70">%s</td>',llamadas{k,2})];
                      texto=[texto,sprintf('<td width="10%%"
align="center" height="70">%d</td>',llamadas{k,3})];
                      texto=[texto,sprintf('<td width="10%%"
align="center" height="70">%s</td>',llamadas{k,4})];
                      texto=[texto,sprintf('<td width="15%%"
align="center" height="70">%s</td>',llamadas{k,5})];
                      texto=[texto,sprintf('<td width="15%%"
align="center" height="70">%s</td>',llamadas{k,6})];
                      texto=[texto,sprintf('<td width="10%%"
align="center" height="70">%s</td>',datestr(llamadas{k,7},'dd-mm-yyyy
HH:MM:SS'))];
                      texto=[texto,sprintf('<td width="10%%"
align="center" height="70">%s</td>',datestr(llamadas{k,8},'dd-mm-yyyy
HH:MM:SS'))];
                      texto=[texto,sprintf('<td width="30%%"
align="center" height="70">%s</td>',llamadas{k,9})];
                      texto=[texto,sprintf('<td width="10%%"
align="center" height="70">%s</td>',llamadas{k,10})];
                      texto=[texto,sprintf('<td width="30%%"
align="center" height="70">%f</td>',llamadas{k,11})];
                      texto=[texto,'</tr>'];
```

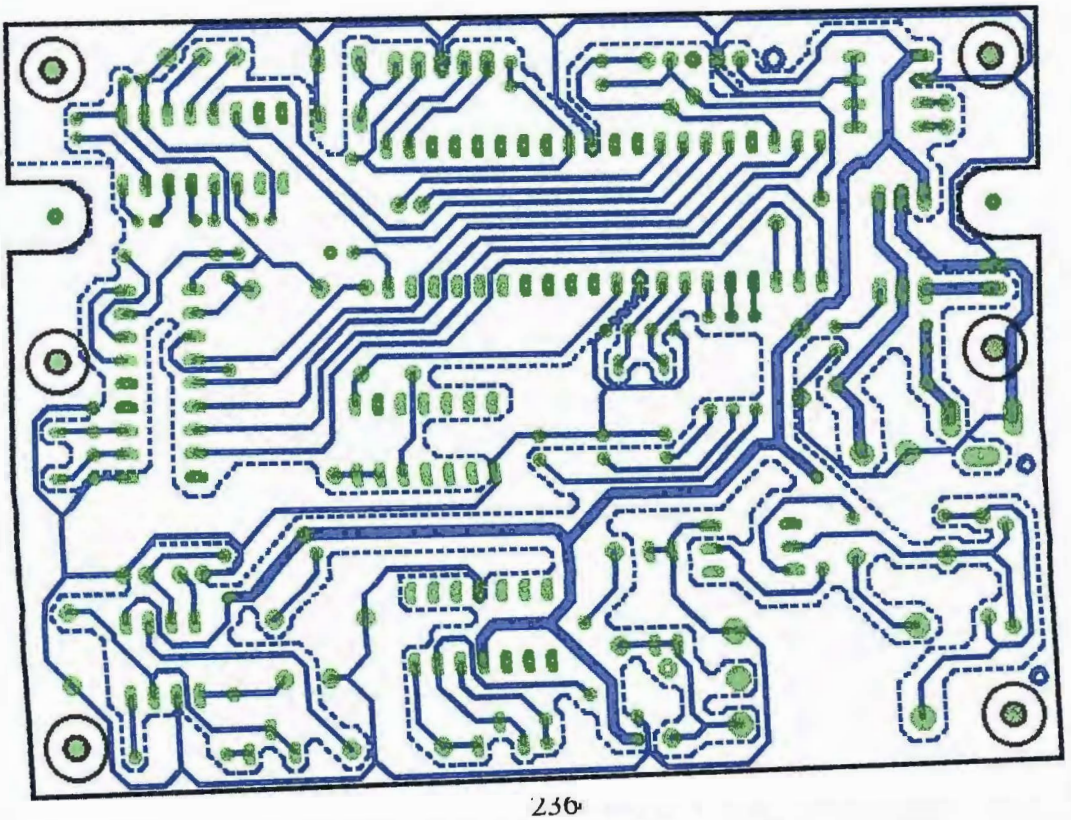
```
                    waitbar(k/filas,h,sprintf('Generando reporte de cada
llamada... %d%%',round(k/filas*100))); % ventana de barra de progreso
                end
            end

            texto=[texto,'</table></p>']; % termina la tabla de todas
las llamadas

            texto=[texto,'</body></html>'];

            fwrite(id_reporte,texto,'uchar'); % se escribe el archivo
nuevo
            fclose(id_reporte); % cierra el archivo que se acaba de
escribir
            close(h); % cierra la ventana de la barra de progreso
            uiwait(msgbox('El reporte que se presenta a continuación
está almacenado en C:\Reportes','Aviso','help'));
            pause(0.5); % pausa de 1/2 segundo
            eval(sprintf('web file:///%s -browser',nombre_reporte)); %
abre el reporte en página web
        end

    end % fin de switch
end % fin de if de error de intervalo de tiempo
```

Anexo F: CIRCUITO IMPRESO

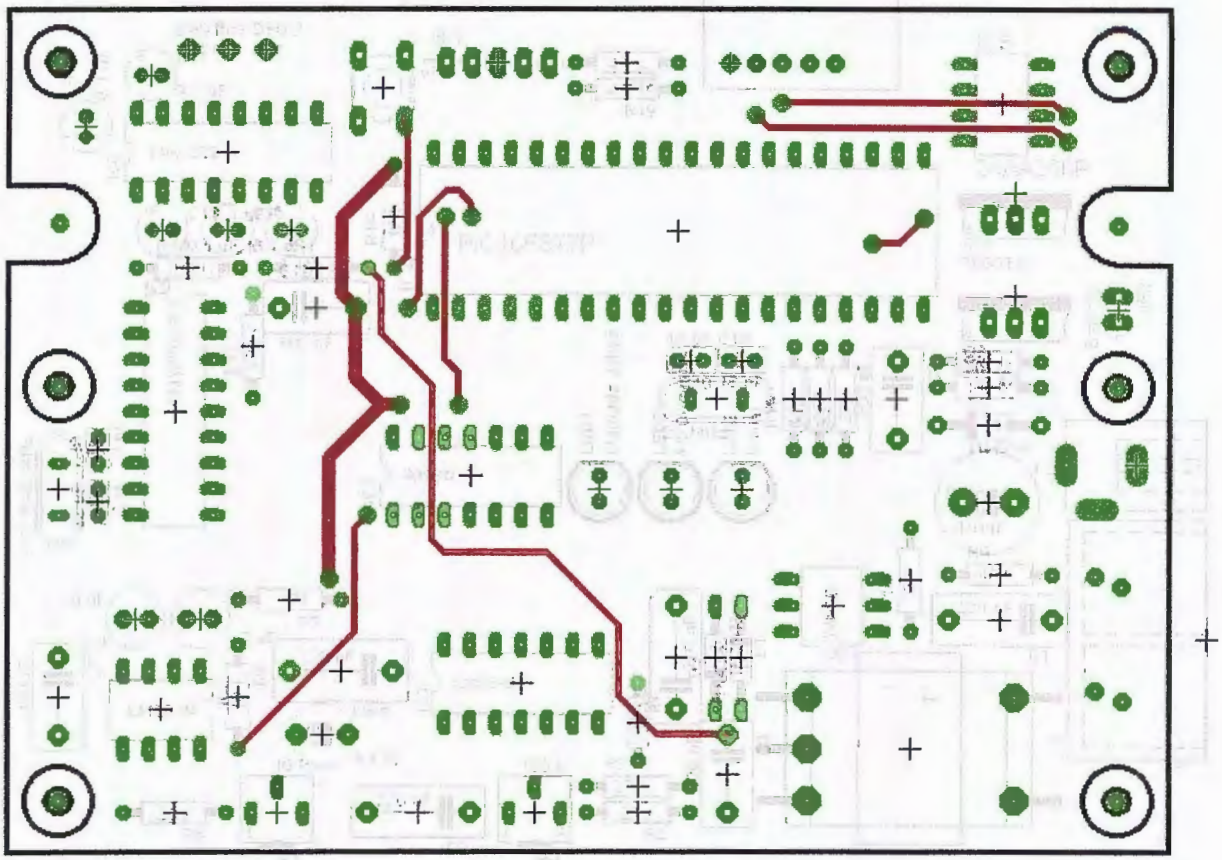
arriba



abajo



arriba



Anexo G: BASE DE DATOS

Tabla ABONADO			
Clave primaria	**Campo**	**Tipo**	**Longitud**
	Nombre	Texto	50
	Apellido	Texto	75
	Tipo_identificacion	Texto	9
*	Identificacion	Texto	50
	Direccion	Texto	100

Tabla LLAMADAS			
Clave primaria	**Campo**	**Tipo**	**Longitud**
	Toma_datos	Fecha/Hora	
	Abonado	Texto	50
*	Telefono_A	Número	Entero largo
	Operador_A	Número	Entero largo
	Telefono_B	Texto	50
	Operador_B	Texto	110
*	Inicio_llamada	Fecha/Hora	
	Fin_llamada	Fecha/Hora	
	Tipo_llamada	Texto	105
	Duracion_llamada	Texto	50
	Tarifa	Número	Simple

Tabla OPERADOR			
Clave primaria	**Campo**	**Tipo**	**Longitud**
*	Id	Autonumérico	Entero largo
	Nombre	Texto	110
	Fijo	Texto	2

Tabla PLAN_NUMERACION			
Clave primaria	**Campo**	**Tipo**	**Longitud**
*	Lote_Inicio	Número	Doble
	Lote_Fin	Número	Doble
	Tipo	Texto	255
	Id_Operador	Número	Entero largo

Tabla TARIFA			
Clave primaria	**Campo**	**Tipo**	**Longitud**
*	Id_Operador	Número	Entero largo
*	Fecha_modificacion	Fecha/Hora	
	Inicio_horario_pleno	Texto	5
	Inicio_horario_reducido	Texto	5
	Tarifa_plena_local	Número	Simple
	Tarifa_plena_nacional	Número	Simple
	Tarifa_reducida_local	Número	Simple
	Tarifa_reducida_nacional	Número	Simple
	Tarifa_internacional	Número	Simple
	Tarifa_a_celular	Número	Simple

Relaciones entre las tablas: