TRABAJO ESPECIAL DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN MODELO DE TELÉFONO PÚBLICO INALÁMBRICO BASADO EN LA TECNOLOGÍA GSM

Presentado ante la Ilustre
Universidad Central de Venezuela
Por el Br. Nunes F., Joao L. G.
para optar al título de
Ingeniero Electricista

Caracas, 2008

TRABAJO ESPECIAL DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN MODELO DE TELÉFONO PÚBLICO INALÁMBRICO BASADO EN LA TECNOLOGÍA GSM

Prof. Guía: Ing. Pedro Pinto
Tutor Industrial: Ing. Mario Giallorenzo

Presentado ante la Ilustre
Universidad Central de Venezuela
Por el Br. Nunes F., Joao L. G.
para optar al título de
Ingeniero Electricista

Caracas, 2008

ÍNDICE GENERAL

CONSTANCIA DE APROBACIÓN

Caracas, octubre de 2008

Los abajo firmantes, miembros del Jurado designado por el Consejo de Escuela de Ingeniería Eléctrica, para evaluar el Trabajo Especial de Grado presentado por el Bachiller Joao Luis G. Nunes Fernandes, titulado:

"DISEÑO E IMPLEMENTACIÓN DE UN MODELO DE TELÉFONO PÚBLICO INALÁMBRICO BASADO EN LA TECNOLOGÍA GSM"

Consideran que el mismo cumple con los requisitos exigidos por el plan de estudios conducente al Título de Ingeniero Electricista en la mención Electrónica, Computación y Control, y sin que ello signifique que se hacen solidarios con las ideas expuestas por el autor, lo declaran APROBADO.

Prof. Rafael Rivero
Jurado

Prof. Carolina Regoli
Jurado

Prof. Pedro P. Pinto
Profesor Guía.

DEDICATORIA

Dedicada con mucho agradecimiento la memoria de la querida y apreciada Profesora Mary Power, ícono representativo de nuestra escuela. Que Dios la tenga en su eterna gloria y que siempre sea recordada por todos, como siempre lo hará el autor de esta obra en el largo camino por la vida que siente que apenas comienza…

RECONOCIMIENTOS Y AGRADECIMIENTOS

Primero que todo, agradezco a mis padres y demás familiares por el apoyo recibido a lo largo de mis estudios, permitiendo así llegar a obtener el título que se me concede en este trabajo, fruto de la dedicación puesta para alcanzar la meta esperada.

Agradecimientos a todos los trabajadores e integrantes del comité de ingenieros de la empresa Electrónica y Telecomunicaciones Generales (ETG) por su valiosa colaboración al haberme dado la oportunidad de desarrollar el trabajo de grado presentado en este documento, especialmente Edith Tadino, Yelitza Espinoza y los Ingenieros Danis Sangoi, Mario Giallorenzo, Raúl Moreno y Francisco Brando por su valiosa colaboración.

Nunes F., Joao Luis G.

DISEÑO E IMPLEMENTACIÓN DE UN MODELO DE TELÉFONO PÚBLICO INALÁMBRICO BASADO EN LA TECNOLOGÍA GSM

Profesor Guía: Prof. Pedro P. Pinto. Tutor Industrial: Ing. Mario Giallorenzo. Tesis. Caracas. U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Opción: Electrónica, Computación y Control. Institución: Electrónica y Telecomunicaciones Generales (ETG). 2008. 98 h. + anexos.

Palabras Clave: Telefonía Pública; Tecnología Celular GSM; Programación Orientada a Objetos; Sistemas Operativos Multitareas.

Resumen. Se plantea el diseño e implementación de un prototipo de teléfono público inalámbrico, desarrollado en las instalaciones de la empresa Electrónica y Telecomunicaciones Generales (ETG), con sede en Caracas. Este modelo de teléfono público opera bajo la red inalámbrica GSM y es tarifado por medio del uso de tarjetas de banda magnética, compartiendo todas las características y similitudes de los teléfonos públicos de esta naturaleza desarrollados anteriormente en el país. Dicho equipo ha sido programado bajo un ambiente de sistema operativo multitareas de tiempo real.

ÍNDICE DE TABLAS

Anexo 3:

Anexo 5:

ÍNDICE DE FIGURAS

SIGLAS

- A: Ampere.
- AC: Corriente Alterna.
- ANSI: American National Standards Institute (Instituto Nacional Americano de Standards).
- C: Capacitancia.
- CANTV: Compañía Nacional de Teléfonos de Venezuela.
- CONATEL: Comisión Nacional de Telecomunicaciones.
- CPU: Central Processing Unit (Unidad de Procesamiento Central).
- DC: Corriente Continua.
- DTMF: Dual Tone Modulation Frequency (Modulación en Frecuencia de Tonos Dobles).
- EGPRS: Enhanced General Packet Radio Service.
- EPA: Enhanced Performance Architecture (Arquitectura de Desempeño Mejorado).
- ETG: Electrónica y Telecomunicaciones Generales.
- F: Farads.
- GPL: General Public License (Licencia Pública General).
- GPRS: General Packet Radio Service.
- GSM: Global System for Mobile Communications.
- HMI: Human-Machine Interface (Interfaz Hombre-Máquina).
- Hz: Hertz.
- I2C: Inter-Integrated Circuit (Circuito Inter-Integrado).
- IEC: International Electrothecnical Comission (Comisión Internacional Electrotécnica).

- ISO: International Standards Organization (Organización Internacional de Standars).
- LCD: Liquid Chrystal Display (Pantalla de Cristal Líquido).
- LED: Light Emitter Diode (Diodo Emisor de Luz).
- MIPS: Millones de Instrucciones Por Segundo.
- OSEK: Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (Sistemas Abiertos e Interfaces para la Electrónica Automotriz).
- OSI: Open Systems Interconection (Interconexión de Sistemas Abiertos).
- PIN: Personal Identification Number (Numero de Identificación Personal).
- R: Resistencia.
- RAM: Random Access Memory (Memoria de Acceso Aleatorio).
- ROM: Read Only Memory (Memoria de Sólo Lectura).
- RTCC: Real-Time Clock Calendar (Reloj Calendario de Tiempo Real).
- SIM: Subscriber Identity Module (Módulo de Identidad de Suscriptor).
- SPI: Serial Peripheral Interface (Interfaz Periférica Serial).
- UART: Universal Asynchronic Receiver Transmitter (Receptor Transmisor Asincrónico Universal).
- V: Volt.
- Ω: Ohm.

INTRODUCCIÓN

El desarrollo de productos y avances tecnológicos competitivos que participen en el mercado requiere de una rápida adaptación de tecnologías, llevando a cabo procesos y métodos que ofrezcan resultados palpables, de alta confiabilidad, cuyas expectativas y exigencias sean cumplidas oportuna y satisfactoriamente. Se necesita aprovechar las ventanas de oportunidad en tiempos viables y con costos rentables. Los procesos involucrados en este tipo de actividad involucran el concurso de múltiples disciplinas que se coordinen en la generación de un nuevo producto, tanto en lo concerniente al área de los equipos de telecomunicaciones como es el caso del trabajo presentado en este documento, así como en otras áreas importantes asociadas a la aplicación de tecnologías de última generación, tales como la instrumentación y automatización.

Históricamente, el desarrollo de software en productos y aplicaciones de relativa sencillez respecto a los complejos sistemas de adquisición o de procesamiento de datos que existen actualmente ha dejado experiencias traumáticas a medida que su complejidad aumenta, debido al gran esfuerzo requerido para la depuración de su funcionamiento, considerando a su vez los problemas asociados a las actualizaciones posteriores de software (firmware), el mantenimiento del sistema y la documentación correspondiente que acarrea dicho proceso haciendo que, en general, la robustez y confiabilidad de un producto definitivo caigan en la incertidumbre.

Existen metodologías, técnicas y herramientas basadas en principios de integración de sistemas, principios de ingeniería de software y de bases de datos, que apoyadas en un sistema operativo multitarea tiempo real (RTOS), permitan generar

aplicaciones confiables, robustas y en lapsos predecibles de tiempo, con ventajas de reutilización de código de programación e independencia de la plataforma de hardware implementada, que puede ser planificado por un equipo de trabajo, llevando la culminación del proyecto planteado a buen término y sin interferencia mutua ni ningún tipo de dependencias entre sus componentes.

Los recursos conceptuales para satisfacer las necesidades de desarrollo de software en sistemas/equipos inteligentes son los siguientes: principios de integración de sistemas que rompan el flujo de información por capas bajo un modelo de programación y organización de la información, principios de ingeniería de software mediante técnicas de programación orientada a objetos y de bases de datos en lo referente a la organización de la información, como ha sido mencionado anteriormente.

Estos recursos fueron adaptados para el presente proyecto a partir de trabajos previos de investigación y desarrollo para implementar software diseñado para manejar un problema genérico de adquisición en tiempo real, múltiples enlaces de comunicación (maestro-esclavo), redireccionamiento lógico y de fuentes de información, así como herramientas de configuración dinámica basadas en scripts, tomando por ejemplo problemas clásicos ampliamente tratados en computadores grandes como mainframes y servidores para aplicaciones en tiempo real. El reto principal del presente trabajo radica la integración práctica de los conceptos y principios mencionados mediante el uso de microcontroladores muy pequeños tales como los de las familias PIC18 y PIC24, manteniendo el enfoque de independencia de plataforma, hardware y RTOS.

Todo lo que ha sido señalado anteriormente ha sido aplicado en el trabajo presentado en este documento al desarrollo de un teléfono inalámbrico GSM construido modularmente con un software de enfoque de programación orientada a

objetos (POO). Como muestra del potencial y los beneficios que conllevan adoptar esta metodología de trabajo, la misma permitirá a futuro con el reemplazo de uno de sus módulos el cambio a teléfono de línea cable con cambios menores en el software.

OBJETIVOS

Objetivo General

Diseñar e implementar un modelo de teléfono público inalámbrico basado en la tecnología GSM bajo un horizonte de metodología de documentación controlada que permita llevar de forma confiable el desarrollo a producto comercial.

Objetivos Específicos

- Obtener la información necesaria correspondiente al desarrollo del equipo telefónico y el módulo de gestión, en lo que respecta a la programación del microcontrolador, dispositivos de comunicación y las tecnologías asociadas.

- Diseñar e implementar un hardware adecuado a los requerimientos del equipo telefónico, sus componentes e interfaces, tomando en cuenta elementos de manufacturabilidad, mantenimiento y control de calidad.

- Implementar un software asociado al microcontrolador a ser utilizado, capaz de manejar las interfaces de comunicación y periféricos requeridos y gestionar correctamente los parámetros y alarmas del sistema bajo un ambiente de sistema operativo multitareas.

- Redactar un manual de funcionamiento para el equipo telefónico y centro de gestión orientado a personal técnico de la empresa con base en los documentos emitidos a lo largo del proyecto.

CAPÍTULO I

PLANTEAMIENTO GENERAL

1. Planteamiento del Problema

A lo largo de la historia de los sistemas telefónicos, el servicio de telefonía pública ha sido uno de los pilares fundamentales dentro del sistema de telecomunicaciones tanto a nivel nacional como internacional. Hasta hace no más de un par de décadas, la telefonía fija era el método de comunicación por excelencia para realizar llamadas telefónicas entre abonados. Desde entonces este sistema ha cedido poco a poco espacio a las redes de telefonía móvil celular y sistemas inalámbricos en general, abarcando actualmente un gran segmento del mercado de las telecomunicaciones, dejando a esta tecnología virtualmente obsoleta.

Debido al gran avance en el desarrollo de estas tecnologías y la implantación de redes celulares a lo largo y ancho del país en los últimos años, las empresas dedicadas a este servicio han sido comprometidas legalmente por parte de la Comisión Nacional de Telecomunicaciones (CONATEL) a incluir en sus redes celulares equipos terminales púbicos que permitan la realización de llamadas telefónicas tarifadas a través de monederos o tarjetas magnéticas, según el Reglamento de Apertura de los Servicios de Telefonía Básica de la República Bolivariana de Venezuela. (Decreto Nº 1.095 del 24 de Noviembre de 2000, publicado en Gaceta Oficial Nº 37.085). Sin embargo, estas empresas han estimado la implantación de estos equipos como una inversión fructífera en el tiempo.

La empresa Electrónica y Telecomunicaciones Generales (ETG) ha dispuesto una serie de recursos humanos y financieros para la realización de estudios de desarrollo y diseño de este tipo de teléfonos públicos, teniendo ya experiencias previas en el mercado de telefonía pública con la Compañía Anónima Nacional de Teléfonos de Venezuela (CANTV) en el diseño y manufactura de equipos terminales públicos de par telefónico, tal y como normalmente han sido diseñados hasta nuestros días. Dicha empresa pretende disponer de equipos de telefonía pública inalámbricos para ser comercializados a las compañías que ofrezcan el servicio de telefonía celular, según sus exigencias y expectativas en el mercado, cumpliendo con la exigencia del ente regulador CONATEL.

Como una solución a la exigencia dispuesta por este ente regulador, se propone el diseño de un teléfono terminal público que cumpla con el propósito arriba citado. Este teléfono dispondrá específicamente de la red GSM para efectuar el servicio de llamadas telefónicas y de manejo de datos por vía TCP-IP para reportes de estado y alarmas del dispositivo telefónico. Dicho teléfono debe cumplir con las exigencias y normas dictadas tanto por el ente regulador como por las compañías que deseen disponer de estos equipos, basados en la experiencia anterior de éstas y de la empresa constructora.

2. Descripción General del Equipo

El equipo en cuestión consiste en un sistema telefónico de uso público que dispone de la red celular GSM para efectuar el servicio de llamadas tarifadas para el cual se concibe. Dicho equipo debe montarse en una pared o pedestal, tal y como se estila para cualquier teléfono público convencional y su funcionamiento debe ser independiente de la central en la cual trabaje, sin requerimientos de interfaces adicionales a nivel del servicio de llamadas de voz. Sin embargo, el software asociado al proceso de llamada debe adaptarse a los requerimientos de la central, habiendo

para ello más de un estándar en el mercado. Adicionalmente, el equipo debe reportar fallas a un centro de gestión por medio del protocolo TCP-IP a una dirección IP fija, la cual es determinada por la empresa de telefonía celular. La Figura 1 muestra la vista externa del equipo telefónico.

Figura 1: Vista externa del equipo telefónico.

El equipo telefónico constará básicamente de un microteléfono convencional compuesto de auricular y micrófono, un teclado para introducir un número telefónico y opcionalmente una pantalla alfanumérica donde se mostrarán mensajes de estado, solicitudes de acciones del usuario o el marcado de tonos que se disquen durante una llamada. Adicionalmente, el equipo tendrá botones para llamar gratuitamente a números de emergencia, servicios al cliente o para controlar el nivel de volumen del teléfono durante una llamada.

Las llamadas de voz que soporte el equipo pueden ser salientes o entrantes. Las llamadas salientes pueden efectuarse mediante el uso de una tarjeta de banda

magnética comercial. Para realizarse una llamada, se debe descolgar el teléfono y haber deslizado la tarjeta magnética a través de un lector accesible al usuario. Posteriormente, el equipo telefónico realizará una llamada a un número de plataforma predeterminado que se encargará de gestionar la llamada tarifada.

Al activarse la llamada, el equipo telefónico enviará el código identificador o PIN de la tarjeta por medio de tonos de marcación multifrecuencial (DTMF) a la central. Esta última avisará al usuario el saldo disponible para efectuar llamadas y solicitará el discado del número de destino, el cual será introducido mediante el teclado alfanumérico enviando tonos DTMF a la plataforma a medida que se marcan los dígitos. Durante la ejecución de la llamada el equipo telefónico permitirá la postmarcación de tonos DTMF.

En algunos casos para zonas de bajos recursos, el equipo podrá disponer de un modo de operación llamado Plan Social, el cual permite efectuar llamadas gratuitas de una duración no mayor de 3 minutos. Pasado este tiempo, el equipo telefónico finalizará la llamada automáticamente. Esta característica podrá ser configurable en el software del equipo, según los requerimientos de la compañía que preste el servicio. En este caso, se podrá discar el número de destino por medio del teclado, mostrando los dígitos marcados en pantalla. Después de un tiempo de espera de 5 segundos sin marcar ninguna tecla se ejecutará la solicitud de llamada al teléfono introducido, pudiendo discar posteriormente tonos DTMF.

Cuando el equipo telefónico reciba una llamada, la misma debe ser avisada mediante una corneta o buzzer, la cual se activará cada vez que se dé un repique. Adicionalmente, la pantalla mostrará el número telefónico que realiza la llamada. Al descolgarse el teléfono, se activará la llamada permitiendo la comunicación entre usuarios. Mientras la llamada esté activa, el usuario del equipo podrá discar tonos DTMF por medio del teclado.

El equipo telefónico estará en la capacidad de reportar su estado, mediante el envío de mensajes de fallas que reportará a una central por medio del protocolo TCP-IP, conociendo previamente una dirección IP y un socket determinado previamente configurados. Para efectos del trabajo de grado, se demostrará la viabilidad de esta característica del teléfono mediante el envío de tramas demostrativas a un PC convencional por medio de un paquete de software conocido, debido a que no existe actualmente un protocolo estándar para el envío de estos mensajes.

El equipo telefónico será alimentado por defecto a través de la línea eléctrica convencional (120V @ 60Hz) o de una batería que mantendrá el equipo en funcionamiento mientras el servicio de energía eléctrica haya sido suspendido. Esta batería podrá ser cargada mientras el equipo esté conectado a la línea eléctrica. El equipo telefónico deberá tener todas las protecciones pertinentes contra sobrevoltajes, sobrecorrientes, rigidez mecánica, rigidez eléctrica y de interferencia electromagnética de tal manera que no genere daños al usuario

A excepción de las tarjetas de circuito impreso que fueron diseñadas y ensambladas en la empresa, los dispositivos y componentes utilizados para la manufactura del equipo telefónico tales como la carcasa externa, teclado, pantalla LCD, auricular, carcasa del lector de tarjetas y demás han sido obtenidos mediante proveedores externos con experiencia en la manufactura de los mismos.

CAPÍTULO II

MARCO TEÓRICO

1. Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que utiliza entes de datos llamados objetos que interactúan entre sí para llevar a cabo una tarea común. Esta metodología de programación ha sido adoptada hace ya varias décadas, aunque sólo hasta la década de 1990 ha tomado gran auge, principalmente en sistemas operativos y redes de comunicaciones.

Mediante esta filosofía, se separan los datos (información) y el código (procesos) de tal manera que los primeros definan al estado del objeto en sí mismo, mientras que el segundo define su comportamiento, es decir, la manera cómo interactúan los datos para un propósito específico del objeto. A partir de este principio se desglosan cuatro conceptos fundamentales, los cuales deben ser considerados para que el paradigma se cumpla a cabalidad. La Figura 2 muestra un bosquejo de objeto.

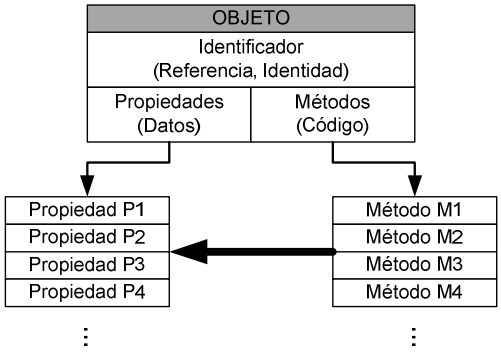


Figura 2: Esquema General de un Objeto

- Estado: El estado del objeto se compone de todos los datos y parámetros requeridos para poder caracterizar al objeto. El estado de un objeto se compone de un conjunto de campos de información llamados propiedades o atributos. Estos atributos definen al objeto y lo diferencian de otros de diferente o incluso de su misma clase.

- Comportamiento: El comportamiento del objeto se compone de un conjunto de rutinas llamadas métodos, los cuales determinan los procesos que éste es capaz de desempeñar. Por medio de estos métodos el objeto es capaz de interactuar a su vez con otros objetos y modificar su estado.

- Identidad: Cada objeto en el universo es único y diferente del resto, y para marcar dicha diferencia se asocia a cada objeto un identificador o ID, con el que puede ser diferenciado, determinando así su papel dentro del programa o tarea.

- Relaciones: Son los nexos que permiten que un objeto pueda interactuar con otros objetos de igual o diferente clase y determinan su papel en la organización del programa.

Esta metodología de programación permite concebir a los objetos como unidades indivisibles, donde su estado y comportamiento están unidos permanentemente como un conjunto. No existe un objeto sin estado, comportamiento, relaciones o identidad propios. Comúnmente se comete el error de crear objetos que sólo contengan información, mientras que otros objetos contienen los métodos requeridos para manejarla. Esto conlleva a realizar una programación estructurada en donde los datos y el código se encuentran desligados y sin relación directa con el único propósito de procesar datos para obtener más datos, perdiéndose así el concepto de objeto.

Existe toda una gama de conceptos asociados a la Programación Orientada a Objetos, los cuales amplían en alcance a los conceptos tradicionales de programación. Entre ellos se encuentran:

- Clase: Conjunto de propiedades de las cuales se compone una familia de objetos determinada.

- Instancia: Asignación de las propiedades de un objeto cuando éste es creado.

- Objeto: Entidad compuesta de propiedades (datos) y métodos (rutinas). Es una instancia a una clase.

- Propiedad: Campo de datos contenido en un objeto o una clase de objetos, cuyo valor puede ser extraído o modificado por un método.

- Método: Algoritmo o rutina asociada a un objeto o una clase de objetos, cuya ejecución es desencadenada producto de la recepción de un mensaje. Los métodos manejan las propiedades de un objeto así como el envío de mensajes a otros objetos.

- Mensaje: Comunicación dirigida a un objeto, la cual ordena la ejecución de un método determinado bajo ciertos parámetros según el evento que lo haya producido.

- Evento: Suceso del sistema, el cual es manejado por el sistema, enviando un mensaje al objeto apropiado.

Algunas de las propiedades de la Programación Orientada a Objetos son:

- Abstracción: Es una característica que permite que los métodos y propiedades inherentes a un objeto sean programados sin revelar sus detalles intrínsecos. Todo objeto puede actuar como un ente de un sistema que realiza una tarea determinada modificando su estado sin la necesidad de intervenir externamente en su comportamiento. Este concepto se divide en dos ramas: la abstracción de procedimientos, en donde los procedimientos (métodos) que ejecuta un objeto pueden ser programados modularmente ignorando los detalles menores. Derivado a esto, existe la abstracción de datos, en donde la representación de los datos que administra un objeto es conocida por el propio objeto y se puede revelar mediante el uso de métodos convenientes. Toda abstracción de datos conlleva siempre a una abstracción de procedimientos.

- Encapsulamiento: Permite llevar a todos los objetos de una misma entidad al mismo nivel de abstracción, aumentando la cohesión de los componentes del sistema.

- Principio de Ocultación: Se basa en impedir que otros objetos modifiquen intencionalmente el estado de un objeto determinado, permitiendo así permanecer aislado del resto del sistema. Bajo este principio, sólo los métodos son capaces de modificar el estado de un objeto, impidiendo así efectos secundarios y la corrupción de información. Sólo en algunos casos específicos se le permite al resto del sistema modificar directamente el estado de un objeto.

- Polimorfismo: Permite identificar propiedades o métodos de objetos de clases diferentes bajo un mismo nombre, sin que sus comportamientos tengan que ser iguales.

- Herencia: Todo objeto puede heredar las propiedades y métodos pertenecientes a objetos de otras clases sin la necesidad de ser redefinidos, permitiendo así la creación de objetos con propiedades y métodos comunes con propósitos o características particulares. Esta propiedad favorece el encapsulamiento y el polimorfismo. Esta herencia puede ser simple o múltiple.

- Transportabilidad: Se dice que un código es transportable cuando se logra que éste funcione para diversas aplicaciones o plataformas haciendo cambios en los campos de estado de los objetos que lo componen sin alterar sus métodos, es decir, hacer cambios de configuración (datos) sin alterar el programa (código).

La Programación Orientada a Objetos goza de las siguientes ventajas:

- Facilita la escritura de código más simple, concreto y organizado.
- Reduce el tamaño de memoria de programa requerida.
- Aumenta la reutilización y mantenimiento de código, permitiendo que el mismo sea transportable para varias aplicaciones.

La Programación Orientada a Objetos presenta sin embargo las siguientes desventajas:

- Curvas de Aprendizaje Largas.
- Dependencia del lenguaje de programación.
- Problemas en la determinación de clases, propiedades y métodos consistentes.
- Afecta el desempeño y la velocidad de procesamiento del sistema.

2. Modelos Estructurados de Objetos

Para cualquier desarrollo de software industrial o comercial debe existir una estructuración sólida de los procesos que integran el sistema. Esta estructuración debe permitir la transportabilidad y jerarquización de éstos bajo una estructura coherente. Uno de los modelos más populares es el presentado por la ISO (International Standards Organization) llamado OSI (Open Systems Interconection), el cual ha sido implementado en diversos protocolos industriales y de comunicación a nivel mundial. A partir de este modelo se ha derivado otro más simplificado, orientado para sistemas embebidos de menor complejidad llamado modelo EPA (Enhanced Performance Architecture). Ambos modelos serán introducidos a continuación.

2.1. Modelo de Capas ISO/OSI

El modelo OSI (Open Systems Interconnection) permite conectar sistemas abiertos independientemente de su complejidad con el fin de intercambiar información. Se compone de siete capas independientes con diferentes atributos y funciones, definidas bajo la norma ISO 7498. La Figura 3 muestra un esquema general de la estructura OSI.

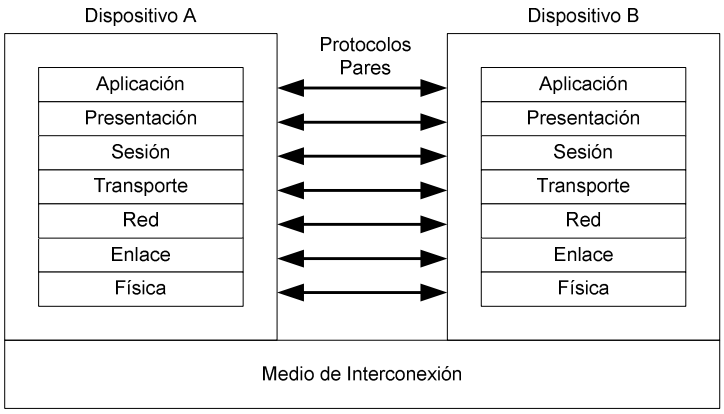


Figura 3: Estructura del modelo de capas ISO/OSI.

El modelo de capas ISO/OSI se compone de las siguientes capas:

- Capa Física: Comprende las interfaces de hardware que componen el sistema. Incluye conexiones, entradas, salidas y dispositivos físicos que hacen posible la comunicación tales como cables, conectores o antenas. En esta capa pueden intervenir señales digitales o analógicas de cualquier naturaleza.

- Capa de Enlace: Es la encargada de gestionar, verificar y filtrar el flujo de información proveniente de la capa física. También determina errores de transmisión de datos o de colisión y de prestar servicios específicos a la capa de red, entregando o recibiendo tramas de información verificadas previamente.

- Capa de Red: Es la encargada de determinar el destino de los paquetes de información en el proceso de comunicación. En algunas aplicaciones, esta capa es muy sencilla o incluso puede llegar a ser inexistente.

- Capa de Transporte: Traduce los paquetes de información provenientes de la capa de red para entregarlos a la capa de sesión y viceversa. La capa de transporte es la que independiza la información de la tecnología o protocolo de comunicación utilizado. Al efectuarse una transmisión, la capa de transporte recibe los datos de la capa de sesión, los divide y empaqueta para así pasarlos a la capa de red. Al recibir datos, los paquetes pasan de la capa de red a ser codificados nuevamente para ser tratados por la capa de sesión. Esta capa debe asegurarse que los datos empaquetados lleguen a destino en el orden correcto.

- Capa de Sesión: Es un añadido de la capa de transporte. Dentro de sus funciones está la sincronización, control de intercambio de información y la administración del canal para casos de canales half-duplex. En cierto tipo

aplicaciones se puede prescindir de esta capa cuando no sea necesaria la supervisión del canal.

- Capa de Presentación: Se encarga de codificar datos de la capa de aplicación en datos que puedan ser transmitidos por el canal en capas inferiores y a su vez traducir los paquetes crudos de información en información inteligible en forma de comandos o tramas a la capa de aplicación.

- Capa de Aplicación: Esta capa contiene las aplicaciones del sistema, las cuales son las que entran en contacto directo con el usuario. Esta capa es la encargada de administrar la información y solicitudes provenientes de otros nodos de comunicación y de enviar los comandos o archivos necesarios por medio del canal a petición del sistema, un usuario o ante la ocurrencia de un evento.

2.2. Modelo de Capas EPA

El modelo EPA (Enhanced Performance Architecture) es un derivado simplificado del modelo ISO/OSI y promovido por la IEC (International Electrothecnical Comission), el cual comprende fundamentalmente las capas Física, Enlace, Aplicación y de Usuario. Por lo general, se implantan subcapas dentro de estas cuatro mencionadas que complementan el modelo de acuerdo a las exigencias del ambiente de comunicación. Por ejemplo, es común añadir una subcapa de transporte dentro de la capa de enlace cuando se manejan protocolos de transmisión de gran cantidad de datos. Mediante este modelo se prescinde de la capa de sesión, ya que no hay que establecer rutas para los mensajes ni paquetes de información. La Figura 4 ilustra el conjunto de capas que conforman el modelo.

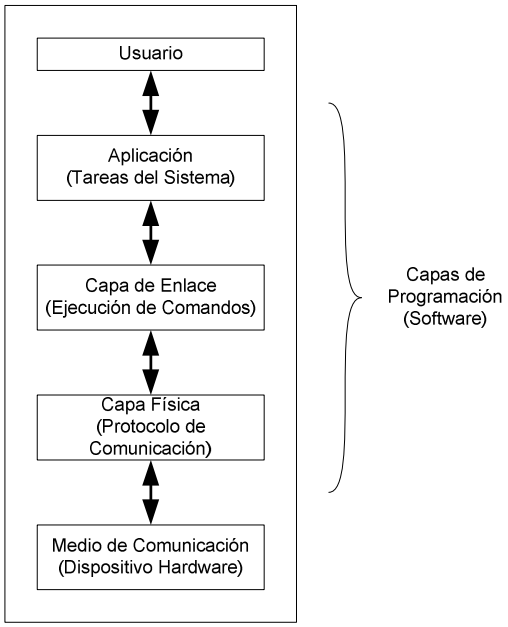


Figura 4: Estructura del modelo de capas EPA.

3. Señales Multifrecuenciales DTMF

El sistema de señales Multifrecuenciales DTMF (Dual Tone Modulation Frequency) se compone de grupos de dos frecuencias enviadas simultáneamente a través de un canal de comunicaciones. Cada una de las señales emitidas tiene forma sinusoidal con frecuencias de rangos diferentes pertenecientes a dos grupos de cuatro (4) frecuencias definidas, llamados Grupo Superior y Grupo Inferior. Estas frecuencias han sido seleccionadas bajo el criterio de que la suma o diferencia de las frecuencias de cada par de tonos correspondientes a un código cualquiera no puede dar como resultado una frecuencia dentro de las seleccionadas ni pueden existir armónicos entre ellas. Esta serie de códigos admite hasta un máximo de 16 combinaciones con valores del 0 al 9, asterisco (*), numeral (#) y las letras de la A a la D. La Tabla 1 y la Tabla 2 muestran respectivamente los grupos de frecuencias y los códigos asociados a cada par de frecuencias respectivamente.

Grupo	Frecuencia en Hz			
Grupo Superior	1209	1336	1477	1633
Grupo Inferior	697	770	852	941

Tabla 1: Grupos de Frecuencias de las señales DTMF.

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

Tabla 2: Códigos asociados a pares de frecuencias de las señales DTMF.

CAPÍTULO III

MÓDULOS Y PERIFÉRICOS DEL SISTEMA

El hardware diseñado para el teléfono terminal público se compone de tres tarjetas independientes con funciones y propósitos diferentes. Por medio de una serie de conectores con entradas y salidas seleccionadas estratégicamente, estos módulos se acoplan para formar un producto determinado, ya sea el presentado en este documento o cualquier otro del que se desee disponer para proyectos futuros. Estos módulos están diseñados para ser reutilizables e intercambiables entre productos diferentes, reduciendo el tiempo de diseño de hardware y facilitando la reparación de cada módulo. Se describen a continuación las características de los módulos del sistema. La Figura 5 muestra un diagrama de bloques de los módulos que lo componen. Posteriormente, la Figura 6 y la Figura 7 muestran la disposición de los componentes del equipo.

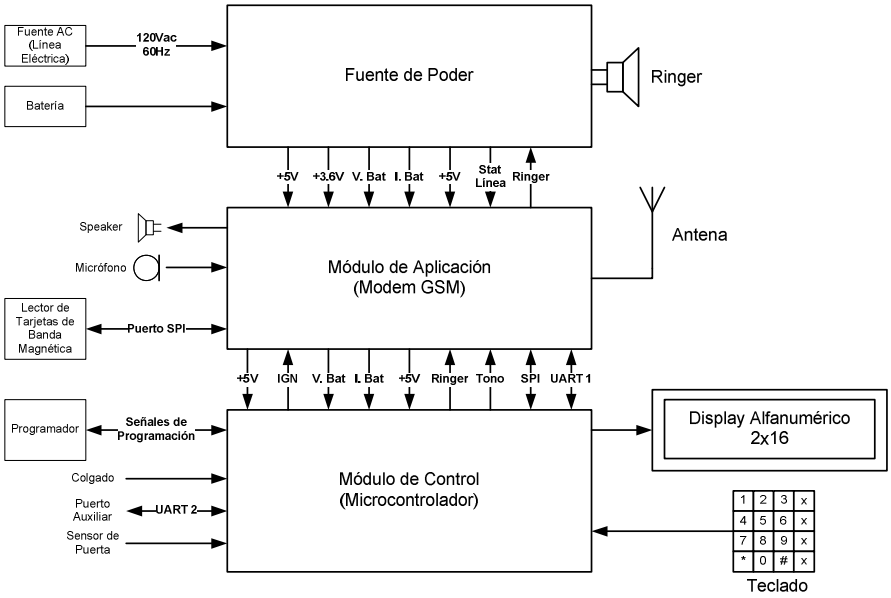


Figura 5: Diagrama de Bloques del Hardware del Sistema.

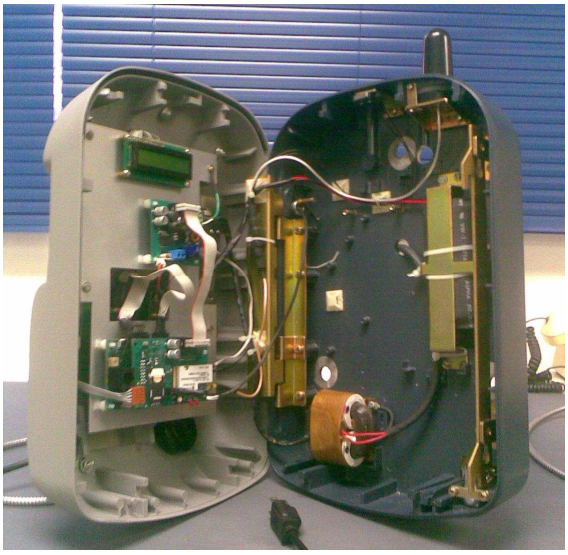


Figura 6: Distribución interna de elementos del teléfono público

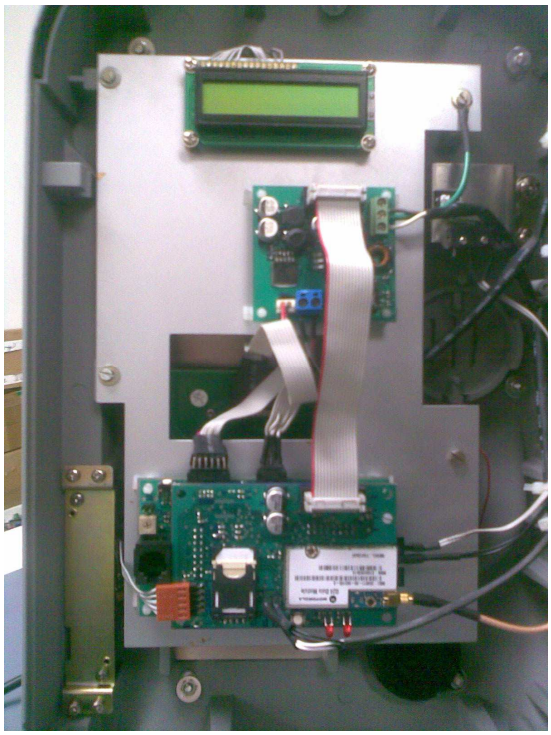


Figura 7: Distribución interna de tarjetas de circuito impreso del teléfono público

1. Módulo de Control

El módulo de control contiene los dispositivos e interfaces periféricas de uso más frecuente que se puedan implementar en cualquier proyecto. Dispone de un microcontrolador maestro de la familia PIC24, memoria EEPROM I2C externa, entradas/salidas digitales y osciladores para el microcontrolador y de reloj de tiempo real (RTCC). Adicionalmente, se han habilitado 2 puertos UART, un puerto SPI, y un puerto I2C para conexiones con otros módulos externos. La memoria incluida en el módulo hace uso del puerto I2C habilitado. El microcontrolador puede ser programado mediante el uso de un conector destinado para tal fin. La Figura 8 muestra un bosquejo del módulo de control. Posteriormente se presenta el circuito impreso ensamblado en la Figura 9.

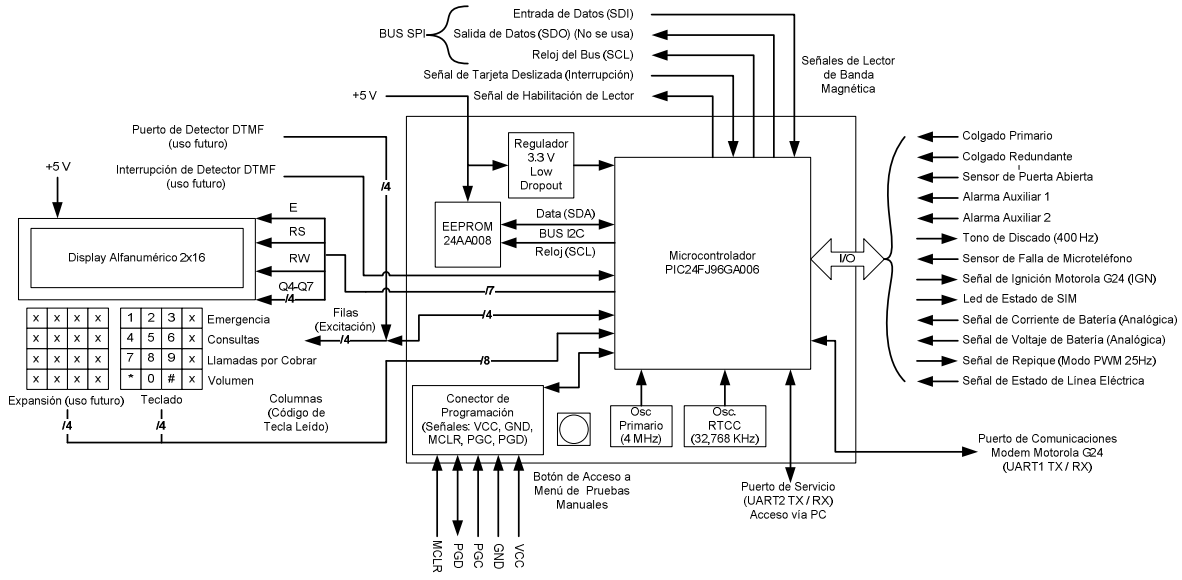


Figura 8: Diagrama de Bloques del Módulo de Control.

Figura 9: Circuito Impreso del Módulo de Control.

1.1. Microcontrolador PIC24

Se ha dispuesto de un microcontrolador PIC24FJ96GA006 de la compañía Microchip como elemento de control del sistema. Los microcontroladores pertenecientes a esta familia presentan una Arquitectura Harvard de 16 bits, anteriormente implementada en la serie de microcontroladores dsPIC, prescindiendo de las ventajas del tratamiento digital de señales. La Figura 10 muestra la disposición física de los pines del microcontrolador. Dicho controlador se ha programado mediante el dispositivo ProMate3 (PM3) de la casa Microchip.

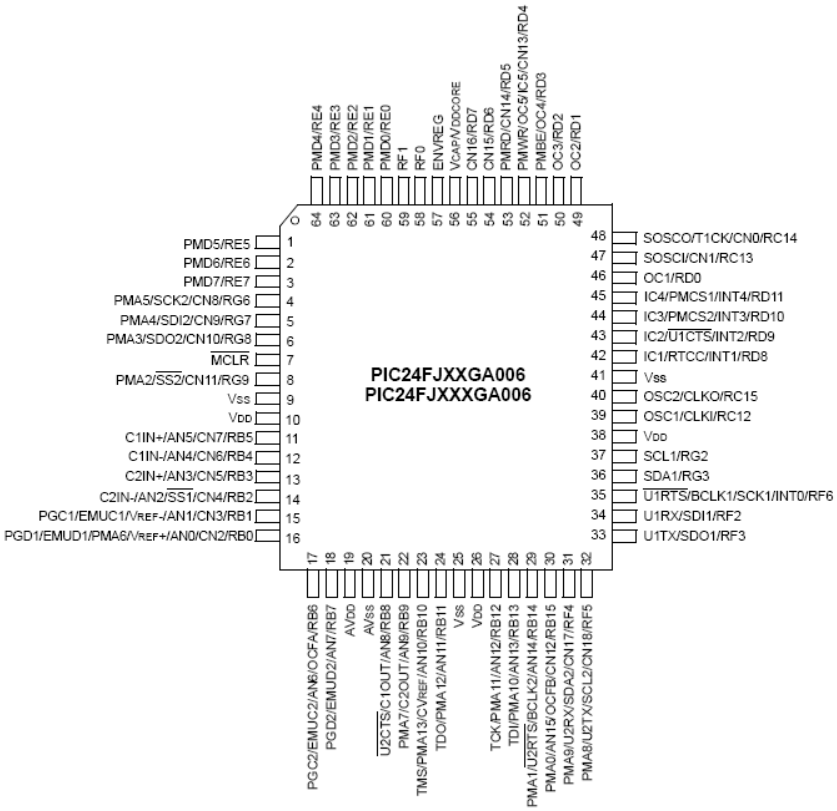


Figura 10: Disposición de pines del microcontrolador PIC24 de 64 pines.

Características Generales:

- Memoria RAM de 8kBytes, direccionada linealmente.
- Memoria ROM de 96kBytes, separada en páginas de 64kBytes para lectura de datos.
- 16 registros de trabajo de 16 bits cada uno.
- Cinco (5) timers de 16 bits, convertibles a dos (2) de 32 bits.
- Hardware de Multiplicación de 17x17 bits y de división de 32 y 16 bits.
- Set de Instrucciones adaptado para lenguajes de alto nivel (ANSI C).
- Desempeño de hasta 16MIPS (Millones de Instrucciones Por Segundo).
- Selección de Oscilador por Software.

- Alimentación de 3 a 3.6 Volt.
- Vectores de Interrupción Independientes y con hasta 8 niveles de prioridad programables.

1.1.1. Unidad de Control (CPU)

La unidad central de procesamiento de los microcontroladores de la familia PIC24 presenta una arquitectura tipo Harvard de 16 bits, que procesa instrucciones a una rata fija de hasta 16MIPS. Cada instrucción emplea dos (2) ciclos de reloj para ser ejecutada, salvo las instrucciones de salto fijo o condicional, las cuales pueden durar dos (2) o tres (3) ciclos de máquina respectivamente. Cada instrucción tiene un tamaño de 4 bytes, por lo cual pueden almacenarse hasta 4 mil instrucciones dentro de la memoria de código del microcontrolador.

La arquitectura de esta familia contiene 16 registros de trabajo de 16 bits cada uno (W0 hasta W15) ubicados al inicio de la memoria RAM, siendo el último de éstos (W15) el registro pila de datos para llamados de subrutinas e interrupciones.

El set de instrucciones de la familia PIC24 se ha ampliado para adaptarse a compiladores de alto nivel como ANSI C. Este set de instrucciones permite la ejecución de instrucciones de hasta tres (3) operandos, instrucciones de salto directo y condicional generalizado, direccionamiento de memoria inmediato, directo, indirecto y por registros. El direccionamiento de memoria ROM está dividido en páginas de 64kBytes, que pueden ser seleccionados con el registro PSVPAG.

La familia de microcontroladores PIC24 contiene un hardware de multiplicación de 17x17 bits con y sin signo que ejecuta este tipo de instrucciones en un ciclo de máquina. A su vez contiene un hardware para división de 32x16 bits por medio de un algoritmo iterativo empleando la instrucción REPEAT.

1.1.2. Entradas/Salidas

Los pines de Entrada/Salida de los microcontroladores de la familia PIC24 presentan diversas funcionalidades de acuerdo con los requerimientos de la aplicación a diseñar para buses de comunicación, comparadores y entradas analógicas entre otras funciones. Estos pines soportan corrientes de hasta 20mA por pin y pueden configurarse como entradas schmitt-trigger con mayor inmunidad al ruido que las entradas convencionales o salidas simples o de drenador abierto (open drain) en los casos que la tensión de las interfaces sea diferente de los 3.3V.

1.1.3. Buses de Comunicación

Los microcontroladores de la familia PIC24 incluyen en su circuitería una serie de interfaces de comunicación para periféricos. A continuación se enumeran los puertos habilitados para este propósito y los dispositivos que manejan:

- UART (Universal Asynchronic Receiver-Transmitter). Puerto UART1 para comunicación con el módem GSM y UART2 como puerto de servicio.
- I2C (Inter-Integrated Circuit). Puerto I2C1 para comunicación con una memoria EEPROM externa.
- SPI (Serial Peripheral interface). Puerto SPI2 para extracción de datos del lector de tarjeta magnética.
- IrDA (InfraRed Data Association) (incluido dentro de la circuitería de la interfaz UART). No implementado.

1.1.4. Reloj-Calendario de Tiempo Real (RTCC)

Los microcontroladores de la familia PIC24 contienen un hardware de Reloj-Calendario de Tiempo Real con programación de interrupción de alarma, liberando

recursos de timers y código de programación, actuando como un módulo independiente en el sistema. Este módulo permite registrar la hora del sistema con la finalidad de emitir los reportes periódicos necesarios al centro de gestión.

1.1.5. Programación del Microcontrolador

La tarjeta de control contiene un conector del tipo RJ-11 de 6 terminales para la programación del microcontrolador. Para programar el microcontrolador se utilizan las señales de Alimentación (VCC y GND), la señal de reset MCLR y las señales de bus de comunicación PGD y PGC de datos y reloj respectivamente. A través de un programador adecuado (Programador ProMate 3 de la compañía Microchip) se puede acceder a la memoria de código del dispositivo.

1.2. Interfaz HMI: Conjunto Teclado-Pantalla

La interfaz hombre-máquina implementada para el desarrollo de la aplicación consiste en un dispositivo de entrada (teclado matricial de 4 filas por 4 columnas) y otro de salida (pantalla alfanumérica de 16 filas por 2 columnas) los cuales permiten la ejecución de llamadas telefónicas salientes, en conjunto con el lector de tarjetas y el gancho de colgado, y el ingreso al menú de pruebas manuales.

El teclado matricial está conectado a 4 pines de salida del microcontrolador, los cuales excitan las filas de éste por medio de niveles de tensión altos (“1” lógico) una fila a la vez, dejando las restantes 3 filas en nivel bajo. Por medio de otros 4 pines de entrada se consigue leer la muestra de una tecla presionada, barriendo las columnas del teclado. Con estos códigos de fila y columna se calcula la posición relativa de la tecla en la matriz, para así obtener una muestra de tecla presionada.

Por medio de un algoritmo de eliminación de rebotes, se confirman las teclas que han sido presionadas para ser gestionadas por la aplicación. Los pines de salida (filas de teclado) se comparten con las entradas de botón de pruebas manuales, falla de microteléfono, puerta abierta y alarma auxiliar. La Figura 11 muestra las conexiones asociadas al teclado matricial.

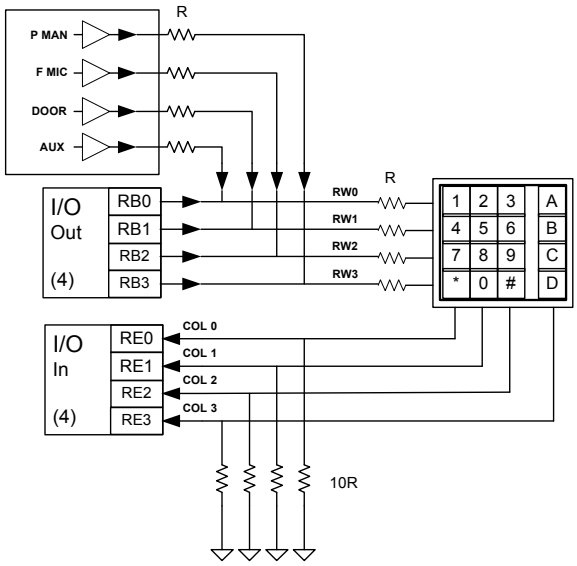


Figura 11: Conexiones del Teclado Matricial.

La pantalla implementada en la aplicación del teléfono público está programada para utilizar un bus de datos de 4 bits, con tres (3) señales de control adicionales: Selección de Registro (RS), Lectura/Escritura (RW) y Habilitación de Comando (E). La señal RS permite elegir entre escribir o leer memoria de pantalla (presentación de caracteres de pantalla) o memoria de caracteres especiales (8 primeros caracteres del set). La señal RW permite la lectura o escritura de datos, ya sea de presentación de caracteres en pantalla o de lectura de cursor o datos. La señal E avisa al controlador de pantalla que se debe ejecutar una instrucción, previamente programados los pines de bus de datos, RS y RW. La Figura 12 muestra las conexiones de la pantalla alfanumérica LCD.

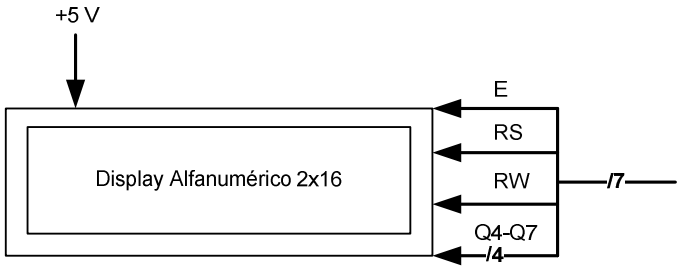


Figura 12: Conexiones de la Pantalla LCD para 4 bits de puerto.

1.3. Memoria EEPROM Externa

Para almacenar los datos que deben quedar permanentemente registrados en el teléfono en caso de ausencia de electricidad o cuando se le aplique mantenimiento se dispone de una memoria EEPROM 24AA008 de 8 kbits, que utiliza un bus I2C para el intercambio de datos con el microcontrolador. En esta memoria se almacenan datos importantes del teléfono tales como:

- Serial Electrónico.
- Periodo de Reporte.
- Umbrales de Alarma.
- Password de Menú de Pruebas Manuales.
- Números de Plataforma y de Centro de Gestión.
- Versión de Parámetros.
- Fecha de Inicialización.
- Banderas de Modo de Operación.
- Código de Redundancia Cíclica (CRC).

Estos datos pueden ser configurables en el menú de pruebas manuales o cada vez que éste se reporte ante el centro de gestión. A través del cálculo de un código CRC se corrobora la fidelidad del contenido total de la memoria contenida en la

memoria externa. En caso de que la memoria sea nueva o esté corrupta, se cargarán una serie de parámetros por defecto para permitir que el teléfono no pierda por completo su funcionalidad antes de ser revisado por personal técnico.

1.4. Puerto Serial Auxiliar

El puerto serial auxiliar es una vía de comunicación alterna que permitirá a personal autorizado configurar parámetros de teléfono y obtener registros de su estado para fines de depuración y recarga de software, carga de parámetros y comportamiento. Para poder acceder a este puerto se deberá disponer de un computador con puerto serial y una interfaz de conversión de niveles de voltaje por medio de un chip MAX232 o similar.

2. Módulo de Aplicación

La tarjeta de aplicación se compone de una serie de dispositivos que caracterizan a la aplicación propiamente dicha, diferenciándola del resto de productos que se deseen implementar o desarrollar. Básicamente, esta tarjeta está constituida por el módem GSM que permite la comunicación con la red inalámbrica (Modem Motorola G24) con sus interfaces de puerto serial, antena, encendido y de tarjeta SIM (Subscriber Identity Module), conexiones con el puerto SPI del microcontrolador para extracción de datos de lector de tarjeta y la circuitería de filtrado para auricular y microteléfono. La Figura 13 muestra un bosquejo de la interconexión de los módulos que componen la tarjeta de aplicación. Posteriormente en la Figura 14 se muestra el circuito impreso correspondiente.

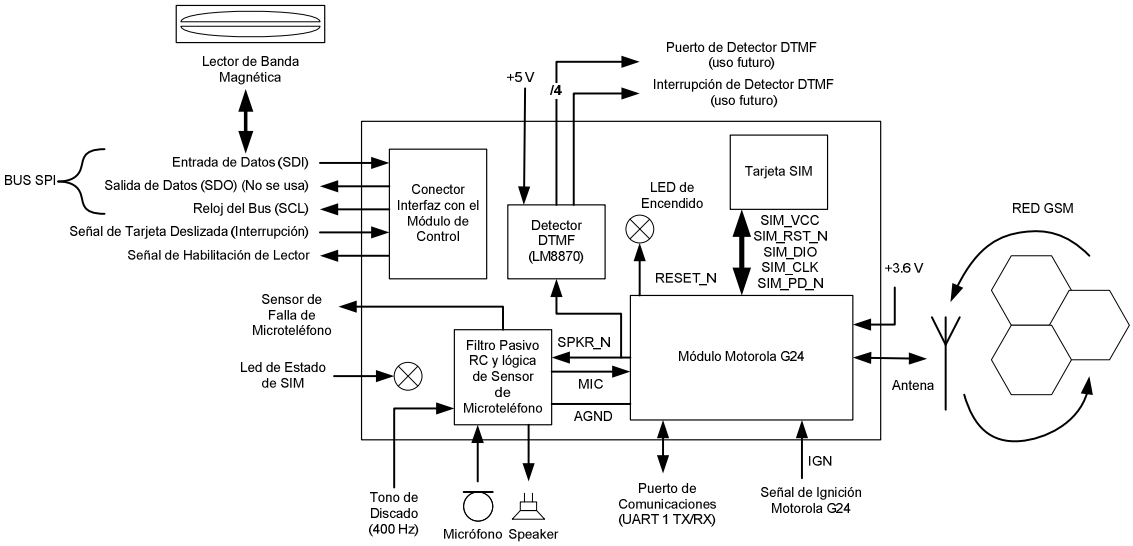


Figura 13: Diagrama de Bloques de la Tarjeta de Aplicación.

Figura 14: Circuito Impreso del Módulo de Comunicación GSM.

2.1. Módulo Motorola G24

El módulo Motorola G24 es uno de los dispositivos de comunicación celular más recientes desarrollados por la casa Motorola. Este módulo está diseñado para

utilizar la red GSM con manejo de cuatro bandas: 850/900/1800/1900MHz con multislot GPRS/EGPRS, pudiéndose comunicar con cualquier red GSM que disponga de servicios de voz y datos. Este módulo dispone de las características fundamentales del núcleo de un equipo celular GSM, transformándolo en un dispositivo independiente de alto desempeño que es capaz de establecer enlaces de voz y datos a través de la red GSM. El módulo G24 dispone de un conjunto de comandos AT, los cuales permiten que sea controlado a través de un puerto serial por el microcontrolador PIC24. En la Figura 15 se muestra la apariencia del módulo Motorola G24.

Figura 15: Módulo Motorola G24

Este módulo dispone de una interfaz de tarjeta SIM (Subscriber Identity Module) con el cual se asocia una línea GSM que contiene un número telefónico y un código de identificación (PIN) de cuatro (4) dígitos tal y como cualquier teléfono celular convencional y que puede ser configurado por medio del menú de pruebas manuales. La interfaz entre el módulo G24 y la tarjeta SIM se compone de las señales SIM_VCC, SIM_RST_N, SIM_DIO, SIM_CLK y SIM_PD_N.

El módulo Motorola G24 permite la marcación de tonos DTMF durante la ejecución de una llamada de voz, sin embargo carece de un hardware adecuado para su detección. Por tanto, se implementará a futuro un hardware detector de tonos

DTMF que por medio de una interfaz adecuada con el microcontrolador PIC24 y con la interfaz de audio del módulo G24 complemente esta desventaja.

2.2. Circuitería de Audio y Micrófono

La circuitería de audio contenida en la tarjeta de aplicación está constituida por un filtro pasivo RC, en donde intervienen las señales de speaker SPKR_N del módulo G24, la tierra analógica (AGND) y un terminal proveniente del microcontrolador que ejerce la función de emitir un tono de 400Hz al descolgar simulando el tono de marcado, ausente en los teléfonos celulares convencionales y cada vez que se pulsa una tecla durante la realización de una llamada saliente. El circuito de Micrófono se compone también de un filtro pasivo RC con una salida que permite al microcontrolador PIC24 detectar fallas de desconexión del auricular en el teléfono. El control de volumen y de silenciado (mute) de audio y micro puede ser controlado por el propio módulo G24 a través de comandos AT.

2.3. Interfaz de Lector de Tarjeta

La interfaz de lector de tarjeta está constituida por un chip dedicado a la extracción de datos de tarjetas por medio de un cabezal magnético, el cual emite una señal de interrupción al microcontrolador cada vez que se deslice una tarjeta. Posteriormente, los datos son extraídos a través de un bus SPI del microcontrolador para ser procesados. Las dimensiones de la tarjeta magnética vienen dadas por la norma ISO 7810, mientras que el formato de datos correspondiente se describe en la norma ISO 7811-3. En la Figura 16 se muestra el circuito impreso del lector de tarjetas magnéticas.

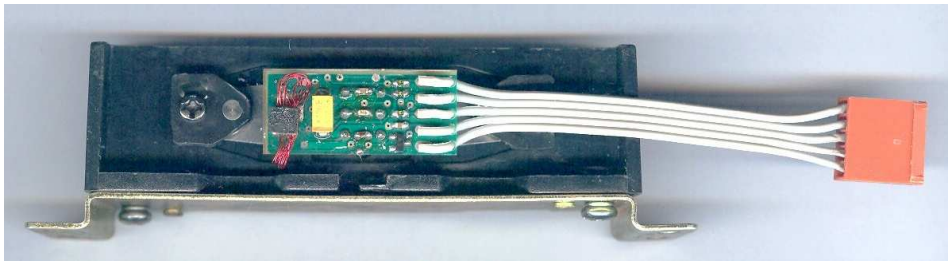


Figura 16: Circuito del lector de tarjeta magnética.

3. Módulo de Fuente de Alimentación

El equipo telefónico dispone de una fuente regulada que admite alimentación desde la línea eléctrica (120V, 60Hz) a través de un transformador reductor protegido por un fusible convencional o por medio de baterías. La Figura 17 muestra una foto del circuito impreso ensamblado.

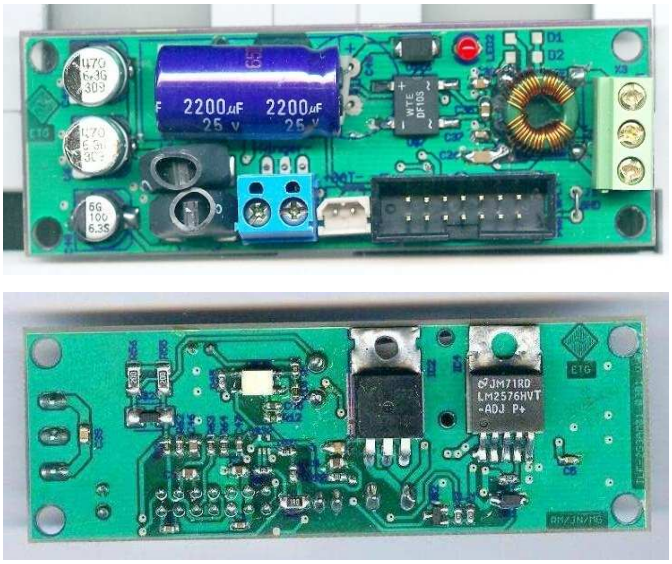


Figura 17: Circuito Impreso de la Fuente de Poder.

Dicha fuente arroja tensiones de +3.6V y +5V por medio de reguladores independientes para alimentar al módulo Motorola G24 y a los periféricos del sistema respectivamente. Una señal digital permite al microcontrolador detectar la presencia

de alimentación por medio de la línea eléctrica, acompañado de un LED indicador con el mismo propósito. Este circuito permite que la pila sea recargada una vez que la red eléctrica se restablezca. Se ha añadido una interfaz de repique para que por medio del microcontrolador PIC24 se avisen llamadas entrantes. La Figura 18 muestra un diagrama de bloques constitutivo del circuito de fuente de alimentación.

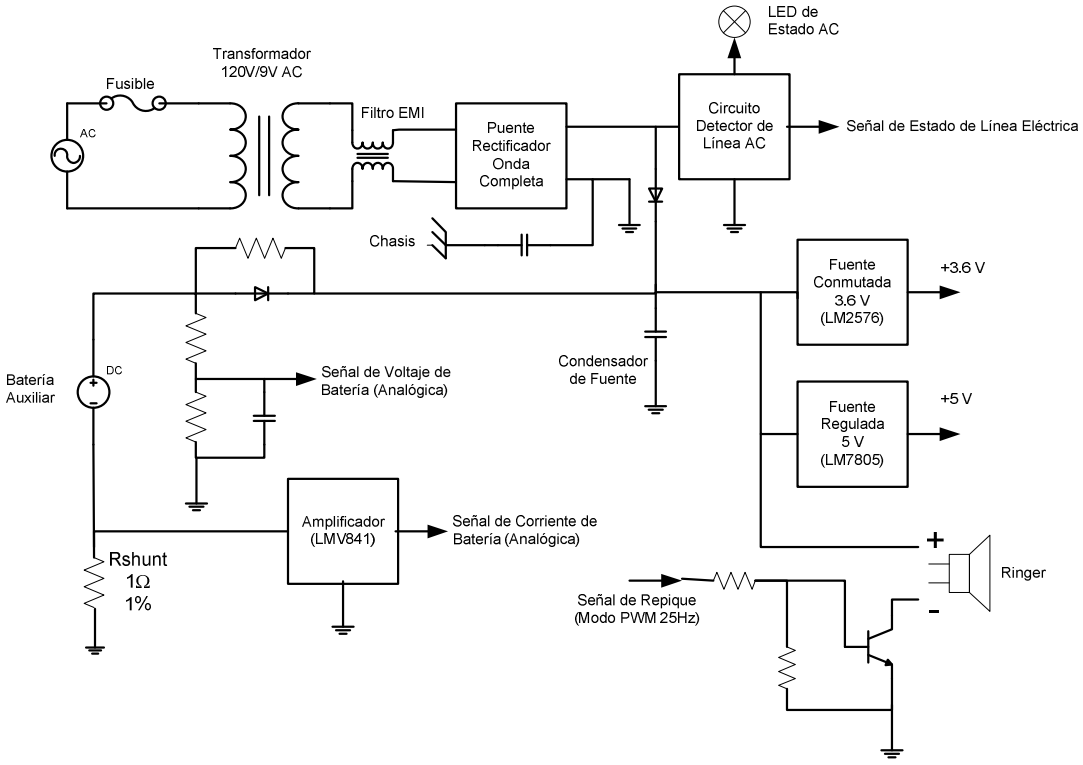


Figura 18: Diagrama de Bloques del Circuito de Alimentación.

3.1. Diseño de la Fuente Regulada del Módulo GSM

La fuente de alimentación para el módulo Motorola G24 es una fuente conmutada step-down de 3,6V de salida. Dicha fuente fue diseñada para soportar picos de corriente de hasta 2,0A, la cual ocurre durante la transmisión de ráfagas de datos. Esta fuente se alimenta del circuito rectificador de onda completa conectado a

la línea eléctrica (120V, 60Hz) a través de un transformador reductor de 120V a 9V. En caso de falla en la línea eléctrica, una batería auxiliar suministrará energía a la fuente.

Esta fuente alimenta a todas las interfaces analógicas y digitales del dispositivo y al Amplificador de Radio Frecuencia interno del módulo G24. Por esta razón, cualquier degradación en el desempeño de la fuente ya sea por ruido, transitorios o pérdidas, puede afectar a las interfaces del módulo y por ende el desempeño del mismo. Especial cuidado debe tomarse durante la transmisión de ráfagas de datos, donde se generan los picos de corriente antes mencionados, ya que en este momento las fluctuaciones de tensión de la fuente suelen ser mayores y podrían afectar el desempeño del módulo Motorola G24. En la Figura 19 se ilustra el comportamiento que presenta la fuente de alimentación durante la transmisión de datos.

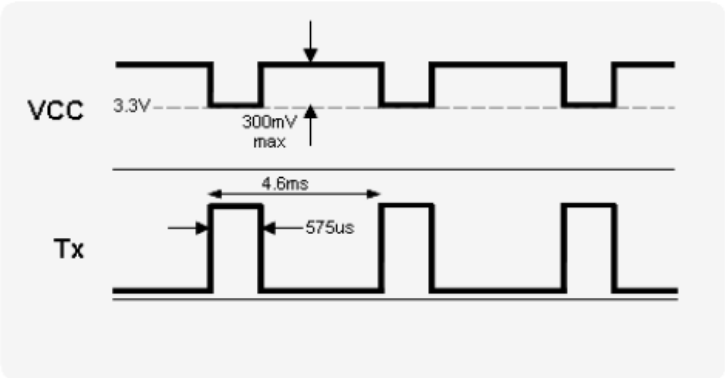


Figura 19: Comportamiento de la fuente de alimentación durante una transmisión de datos.

Se recomienda que la caída de voltaje de la fuente durante la transmisión de datos no exceda los 300mV, medidos desde el conector del módulo. En general, bajo estas condiciones el voltaje de alimentación no debe ser nunca menor a los 3.3 V, o de lo contrario el módulo detectará una caída de voltaje excesiva y se apagará

automáticamente. Se dispuso una serie de condensadores de filtrado, los cuales se muestran en la Tabla 3 para el diseño de la fuente para el módulo G24. Estos condensadores deben ubicarse cerca de los pines de alimentación del módulo.

Condensador	Uso	Descripción
1000 uF	Picos de Corriente de Transmisión.	Minimiza la caída de voltaje provocada por los picos de corriente.
10nF, 100nF	Ruido provocado por señales digitales.	Filtra ruidos provocados por señales digitales.
8.2pF, 10pF	Bandas GSM 1800/1900 MHz	Filtra señales EMI.
33pF, 39pF	Bandas GSM 850/900 MHz	Filtra señales EMI.

Tabla 3: Condensadores de filtrado para fuente del módulo Motorola G24.

3.2. Fuente Regulada para periféricos

Se ha dispuesto de una fuente regulada lineal a partir de un regulador comercial LM7805 de 5V de salida, el cual alimenta a los periféricos del Módulo de Control y la circuitería de lector de tarjetas de banda magnética. La energía necesaria para la operación de esta fuente proviene del mismo circuito de puente rectificador que alimenta al regulador del módulo Motorola G24. La batería auxiliar funcionará como fuente alternativa en caso de falla en la línea eléctrica.

3.3. Sensores de Corriente y Tensión de Batería

Con la finalidad de medir la tensión y corriente de la batería se dispone de dos (2) circuitos transductores cuyas salidas se conectan a dos entradas analógicas que permitirán determinar por medio del microcontrolador el estado de la batería. Se

plantea a futuro considerar una alarma de falla de energía cuando la batería presente bajo voltaje o se comporte irregularmente.

3.4. Circuito de Repique

Se incluye adicionalmente en el circuito de alimentación una interfaz para manejar un buzzer o corneta piezoeléctrica por medio de un pin del microcontrolador, la cual es activada por medio de un transistor con una duración de 1 segundo cada vez que se reciba un repique de parte del módulo GSM. Este sonido es modulado a una cadencia de 25Hz con el fin de simular el efecto del repique de los teléfonos convencionales.

CAPÍTULO IV

METODOLOGÍA DE PROGRAMACIÓN

1. Descripción General

A lo largo de los últimos años, la complejidad de los sistemas computacionales y electrónicos orientados a aplicaciones industriales y multimedia actuales conlleva a una mejor administración de los recursos de hardware y procesos asociados a los mismos donde la complejidad de los procesos está continuamente en crecimiento. El incremento del poder del sistema no siempre requiere microcontroladores más rápidos. Otro camino consiste en utilizar una metodología de manejo de procesos más eficiente.

Una solución elegante se basa en la implementación de un sistema multitareas. Por medio de esta metodología de programación se mejora en gran medida la gestión de los dispositivos físicos disponibles (hardware), los procesos de cómputo asociados a los mismos (programas o software) y de los datos y resultados que maneje o reporte (flujo de información). Lo más importante de este tópico se relaciona con el problema de integrar muchos procesos en una aplicación de cierta complejidad. El problema de concurrencia de procesos se resuelve utilizando un ambiente de sistema operativo multitareas en tiempo real.

El comportamiento de este tipo de sistemas es el equivalente a tener varios microcontroladores independientes funcionando simultáneamente comunicándose entre ellos y apoyados en un controlador maestro llamado gestor de tareas (Scheduler). Bajo este modelo, cada microcontrolador estaría a cargo de un proceso

de cómputo independiente o de un dispositivo de entrada/salida con uno o varios buffers de datos asociados que permitirían el intercambio de información entre las tareas que componen el sistema. El uso de esta metodología facilitaría el desarrollo de aplicaciones complejas, partiéndolas en pequeñas piezas de hardware en un conjunto estructurado.

En la realidad, la implantación de este tipo de sistemas sería costosa y conllevaría al uso de un espacio físico mucho mayor, además de aumentar la probabilidad de fallas de hardware. Un perfecto equivalente es hacer que un solo microcontrolador se encargue de todos los procesos de cómputo y de manejo de hardware de la aplicación por medio de un método llamado Cambio de Contexto. Por medio de este cambio de contexto se logra que todos los pequeños procesos o tareas se ejecuten repartiéndose el tiempo equitativamente entre ellos, guardando adecuadamente el estado del microcontrolador para cada proceso dado, es decir, asociar a cada uno un stack o espacio de memoria suficientemente holgado para guardar el valor de los registros críticos de cada proceso (contador de programa, registros de cálculo, contadores, punteros, etc.) que compartan las tareas y procesos asociados. Esta medida impide que la información procesada por cada tarea y los registros de hardware del proprio microcontrolador se corrompan inconvenientemente al hacerse el cambio de contexto entre tareas.

Para que esta metodología se cumpla es necesaria una aplicación dentro del programa que administre la gestión de los procesos, interrumpiendo la ejecución de una tarea, guardando su contexto y restaurando el contexto de otra que seleccione el gestor de procesos o Scheduler para ser ejecutada de acuerdo a los requisitos del sistema. La prioridad de cada tarea depende de la importancia del proceso que lleve a cabo, la complejidad del mismo y la eventualidad o frecuencia de ejecución entre otras posibles condiciones. Como en todo proceso de cómputo, el cambio de contexto realizado por el gestor de tareas requiere de un tiempo para ser ejecutado. A este

tiempo se le llama latencia del kernel y se define como el tiempo requerido por el microcontrolador para salvar el contexto de una tarea, seleccionar la tarea siguiente que se deba ejecutar, restaurar su contexto y ponerla en marcha. Normalmente, se selecciona el microcontrolador y se configura el sistema operativo que se vaya a utilizar para que dicha latencia sea menor al 5% del tiempo de cómputo disponible.

Por ejemplo, la ejecución de un proceso de recepción por puerto serial podría ser de alta prioridad, ya que no se sabe con certeza en qué momento podría recibirse información, por lo cual se debe condicionar al gestor de tareas a que actúe inmediatamente activando dicha tarea ante una llamada o interrupción del puerto. Por el contrario, un proceso de cálculo de checksum o de CRC (Código de Redundancia Cíclica) de un segmento de memoria requeriría de un procesamiento complicado y de tiempo prolongado, lo cual monopolizaría al microcontrolador, impidiendo la ejecución de otros procesos. Esto se evitaría asignando una baja prioridad a esta última tarea para permitir que otras más críticas como la primera sean ejecutadas luego de lo cual se podrá retornar a la primera sin inconveniente. La Figura 20 muestra el desarrollo de un proceso donde concurren tres tareas en un sistema operativo.

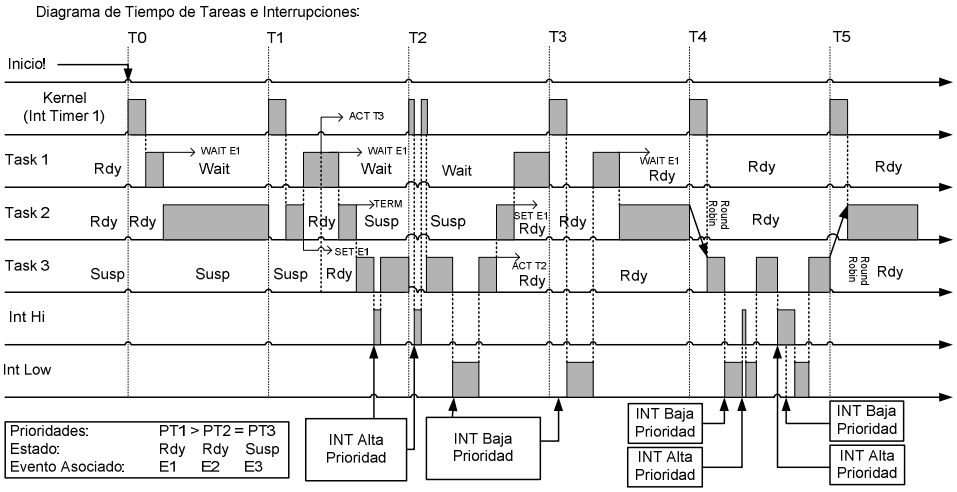


Figura 20: Diagrama de Tiempo de tareas e interrupciones en un sistema multitareas.

Cuando un sistema multitareas se rige bajo esta dinámica se dice que es un kernel de derecho preferencial (Preemptive Kernel), mientras que en el caso de que no existan procesos prioritarios se estaría en presencia de un kernel cooperativo (Cooperative Kernel). Bajo este último modelo la gestión de activación o desactivación de las tareas podría ser gestionada por ellas mismas sin la necesidad de un Scheduler o administrador pero requiere mayor atención del programador sobre la lógica global del sistema.

2. Kernel PICos18 y PICos v1.1

Los kernels PICos18 y PICos son sistemas operativos multitareas abiertos al público desarrollados por la empresa francesa Pragmatec bajo la norma OSEK/VDX para ser usados en los microcontroladores de las familias PIC18 y PIC24 de la casa Microchip respectivamente, considerando las diferencias de arquitectura entre ambas familias pero bajo una filosofía de trabajo común. Estos sistemas operativos han sido programados bajo el paquete de software MPLAB 7.62 de la casa Microchip.

El kernel PICos es un sistema operativo de derecho preferencial, es decir, las tareas que componen la aplicación que se desea desarrollar tienen prioridades diferentes en el sistema. La prioridad de una tarea puede ser reasignada en tiempo de corrida por ella misma o por cualquier otra tarea. Este kernel soporta mensajes entre tareas mediante la señalización de eventos y programación de alarmas temporales o de contadores manejados por el administrador o Scheduler del sistema. Mediante esta política de funcionamiento, cualquier tarea puede ponerse en espera de uno o más eventos definidos, siempre y cuando exista otra tarea o ente que los señalice de acuerdo a la lógica que se desee implementar en el sistema.

La latencia de la ejecución del Scheduler del kernel PICos18 es de 50us a 40MHz. El kernel PICos para microcontroladores de la familia PIC24 es capaz de

ejecutar este proceso en un tiempo incluso mucho menor (12,5us aproximadamente), gracias a que tiene un código más compacto con arquitectura mejorada (16 bits en vez de 8 bits como en el PIC18) y una velocidad de cómputo mayor. Por otro lado, la eficiencia en la ejecución de código de los microcontroladores de la familia PIC24 y una estructura de tareas mejorada dentro de su software son ventajas críticas respecto al kernel PICos18. Para el desarrollo del proyecto se optó por el kernel PICos para microcontroladores PIC24 debido a las ventajas de desempeño y costo que ofrece esta familia de microcontroladores respecto a la familia PIC18.

La tarea administradora o Scheduler se ejecuta periódicamente por medio de la interrupción de **Timer 1**, cuando lo solicite cualquier tarea al señalizar eventos o cuando se modifique la prioridad de una tarea. A la periodicidad de la activación de este administrador (latencia del kernel) se la llama **TickTime**, la cual puede tomar valores de 1 hasta 10ms dependiendo de las características de la aplicación, optando por un tiempo de 4ms en el caso del desarrollo de este trabajo. Esta interrupción periódica permite que varios procesos de la misma prioridad se repartan el tiempo de ejecución cuando tengan que ejecutarse simultáneamente.

Esta interrupción de Timer 1 suele programarse como de alta prioridad, exceptuando los casos donde el cambio de contexto de tareas provoque la corrupción de datos o comportamientos erráticos en el sistema. Es conveniente colocar cualquier otra interrupción como de menor prioridad con el fin de evitar problemas de cambio de contexto entre tareas o corrupción de datos, ya que no se guarda el contexto de la tarea corriente cuando ocurre una interrupción de mayor prioridad que la de Timer 1.

Durante el cambio de contexto, el kernel guarda en stack de software de la tarea corriente la dirección de la instrucción siguiente por medio del Contador de Programa (**PC**), el byte bajo del registro de Status (**SR**), los registros de trabajo W0 a W13, los registros de stack y paginado (W15 y W14 respectivamente) y el bit de

prioridad de interrupción 3 del registro de control de núcleo (**CORCON**), llamado **IPL3,** el cual controla la suspensión de o activación interrupciones anidadas.

2.1. Estructura de Tareas del Sistema

En el sistema multitareas PICos se definen las tareas, sus prioridades, los códigos de error que reporten las funciones que las gestionen y los modos de operación bajo los cuales se determina el comportamiento de la aplicación. Algunos de los tipos de estructuras que contempla el sistema multitareas para declarar tareas son los siguientes:

- **TaskType**: Describe el identificador de una tarea. Este identificador equivale a un número de 16 bits sin signo. De acuerdo a la arquitectura del sistema multitareas, este identificador sólo puede tomar valores **desde 1 hasta 15**, con lo cual se puede decir que el sistema acepta hasta 15 tareas en la misma aplicación y todas deben tener identificadores diferentes. El identificador 0 no se debe usar, ya que para las variables de estado del kernel es indicio de que una tarea con este identificador no está activa o no existe.

- **TaskRefType**: Puntero a una variable de tipo **TaskType**.

- **TaskStateType**: Estado de una tarea. Es un número de 8 bits sin signo, que toma los valores de la tabla para registrar el estado de una tarea. A continuación en la Tabla 4 se enumeran en la tabla los distintos valores que pueden adquirir las variables de este tipo.

Identificador	Valor Asociado
SUSPENDED (Suspendida)	0x00
READY (Lista)	0x01
RUNNING (Corriendo)	0x02
WAITING (En espera)	0x04
RR_RUNNING (Corriendo bajo Round Robin, es decir, con otras tareas de igual prioridad)	0x08

Tabla 4: Identificadores de Estado de las Tareas.

- **TaskStateRefType:** Puntero a una variable de tipo **TaskStateType**.

- **StatusType:** Código de Retorno de las funciones API del sistema multitareas. Equivale a un número de 8 bits sin signo. Los valores posibles para variables de este tipo se muestran a continuación:
  - **E_OK**: 0x00 (Proceso realizado satisfactoriamente)
  - **E_OS_ACCESS**: 0x01 (Error de acceso a recurso del sistema)
  - **E_OS_CALLEVEL**: 0x02 (Error de nivel de llamada a función)
  - **E_OS_ID**: 0x03 (Identificador de tarea o recurso desconocido)
  - **E_OS_LIMIT**: 0x04 (Error de límite de tareas)
  - **E_OS_NOFUNC**: 0x05 (Recurso del sistema inactivo)
  - **E_OS_RESOURCE**: 0x06 (Error de negociación de recurso)
  - **E_OS_STATE**: 0x07 (Estado de recurso inválido)
  - **E_OS_VALUE**: 0x08 (Valor de parámetro inválido)
  - **INVALID_TASK**:0x10 (Identificador de Tarea inválido)

Cada tarea pasa por las cuatro fases definidas por su variable de estado de tipo **TaskStateType**. La Figura 21 muestra gráficamente las transiciones por las que debe pasar una tarea para alcanzar cada uno de los estados definidos anteriormente.

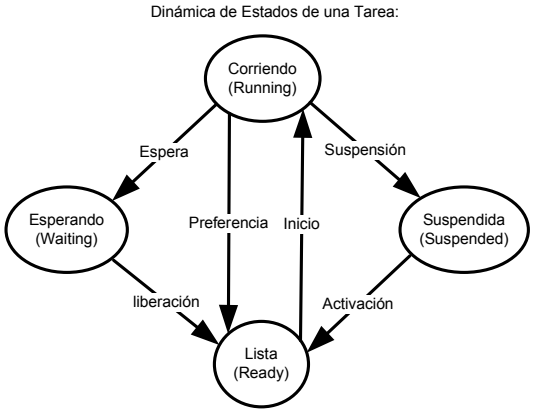


Figura 21: Dinámica de Estados de una Tarea.

De acuerdo a esta gráfica, una tarea que esté ejecutándose (**Running**) sola o junto con otras tareas de igual prioridad (**RR_Running**) puede ser suspendida de la aplicación (**Suspended**), haciendo que el administrador o Scheduler considere a la tarea como no existente o puede ser colocarla en espera (**Waiting**) de un evento determinado, es decir, el administrador colocará la tarea en espera hasta que alguna otra señalice que su proceso ha sido ejecutado, que el sistema haya llegado a un estado crítico o después de transcurrido un tiempo, momento cuando se solicite de su servicio. Por otro lado, una tarea será colocada por el Scheduler como lista (**Ready**) después de la ocurrencia de un evento esperado, la eliminación de su suspensión o ante la activación de un proceso más importante, tomando en cuenta las prioridades de las tareas que componen la aplicación.

Cuando existen varios procesos de la misma prioridad listos para ejecutarse (**Ready**), el administrador reparte el tiempo de ejecución de cada proceso permitiendo que trabajen en paralelo bajo un algoritmo llamado **Round Robin**. Por medio de este algoritmo, el administrador o Scheduler dosifica el tiempo de ejecución en turnos por igual para cada tarea que tenga que ejecutarse. El tiempo de ejecución asignado a cada tarea depende del TickTime que se le asigne al sistema. Es importante recordar que mientras una tarea se mantenga corriendo, las tareas con prioridad menor **nunca**

podrán ejecutarse hasta que la primera detenga su ejecución producto de la espera por un evento o por ser suspendida. Bajo estas circunstancias, si la tarea detiene nunca su ejecución se dice que la misma está **monopolizando el sistema**.

Las funciones que permiten controlar la interacción entre el sistema operativo y las tareas del sistema son las siguientes:

- **DeclareTask(TaskIdentifier)**: Declara el nombre de una tarea en tiempo de compilación.

- **SetPriority (unsigned char new_prio, TaskType TaskID)**: Asigna la prioridad **new_prio** a la tarea de identificador **TaskID.**

- **GetPriority (unsigned char *the_prio, TaskType TaskID)**: Retorna la prioridad de la tarea de identificador **TaskID** en la variable apuntada por **the_prio.**

- **StatusType GetTaskID (TaskRefType TaskID)**: Retorna el identificador de la tarea en ejecución en la variable apuntada por **TaskID.**

- **GetTaskState (TaskType TaskID, TaskStateRefType State)**: Retorna el estado de la tarea de identificador **TaskID** en la variable apuntada por **State.**

- ActivateTask (TaskType TaskID): Activa la tarea cuyo identificador viene dado por **TaskID**. Si la tarea a activar es de mayor prioridad la misma pasará a ejecutarse.

- **TerminateTask (void)**: Suspende la tarea que está corriendo. Para reactivarla es necesario que otra tarea la active usando la función **ActivateTask.**

- **ChainTask (TaskType TaskID)**: Suspende la tarea en ejecución y coloca como lista la tarea con identificador **TaskID**, encadenando los procesos.

2.2. Tabla Descriptora de Tareas

El administrador o Scheduler de tareas dispone de un conjunto de tablas de memoria para gestionar los estados y prioridades de las tareas así como los eventos asociados a las mismas. A continuación se describen cada una de las tablas y registros de acuerdo a su función para el sistema operativo PICos de la familia PIC24.

Como elemento principal de la gestión de tareas del sistema se dispone de la **Tabla Descriptora de Tareas**, en donde toda tarea para ser reconocida por el kernel debe ir acompañada de un bloque de control llamado descriptor de tarea. Esta estructura de control es generada en memoria flash ROM en la tabla **TCB_const_list** estáticamente en tiempo de compilación sin posibilidad de modificarse dinámicamente durante la ejecución del programa. Durante la inicialización del kernel, se escribe en RAM una tabla imagen de la tabla descriptora que puede ser modificada durante la ejecución del programa llamada **TCB_list**.

Esta estructura describe las características más importantes de que debe disponer el kernel para inicializar una tarea o acceder a sus parámetros de funcionamiento, tales como la dirección inicial de la tarea y su stack, su prioridad, su estado al arrancar el sistema multitareas y su identificador o ID entre otros. Cada tarea dentro de una aplicación es reportada al kernel por medio de esta lista descriptora de tareas. En dicha lista se definen las propiedades de cada tarea para uso del kernel. La estructura asociada a este descriptor es del tipo **TCB** (proveniente de las siglas de Task Control Block), cuyo tipo de puntero asociado es **ptrTCB**. La declaración de esta estructura se muestra en la Figura 22.

Task Control Block	
Campo	Descripción
*Stack_register	Registro de Stack (W15)
*Frame_register	Registro de Página (W14)
void(*StartAddress)(void)	Dirección Inicial de Tarea
*StackAddress	Dirección de Stack
StackSize	Tamaño de Stack
TaskID	Identificador de Tarea
Priority	Prioridad de tarea
EventWaited	Evento Esperado
EventReceived	Evento Recibido
State	Estado de Tarea
Type	Tipo de Tarea
Time	Contador local de Tarea
kernelState_copy	Copia de Estado de Kernel
*next	Siguiente Tarea

Figura 22: Estructura de datos para el descriptor de una tarea.

Dentro de los elementos que componen el descriptor de una tarea se encuentran:

- **Registro de Stack** (Stack_register): Este registro es temporalmente guardado en la tabla descriptora de tareas cada vez que se requiere detener su ejecución. El mismo contiene la dirección del puntero de pila (Stack Pointer) de donde se extraen y se almacenan las variables locales de la tarea. Al reactivarse la tarea, el puntero de pila es recargado con este valor para que la tarea siga su ejecución habitual.

- **Registro de Página** (Frame_register): Este registro es temporalmente guardado en la tabla descriptora de tareas cada vez que se requiere detener su ejecución. Cuando la tarea es reactivada, el puntero de pila es recargado con este valor para que la tarea siga su ejecución.

- **Dirección Inicial de Tarea** (StartAddress): Este campo contiene la dirección inicial del código de ejecución de la tarea. Cada vez que la ejecución de una tarea sea suspendida mediante la función **TerminateTask(),** al ser reactivada por el sistema multitareas o por alguna otra tarea mediante la función **ActivateTask(task_ID)**, éste reorganiza el stack de la misma y la relanza haciendo una llamada a subrutina teniendo como parámetro la dirección dada por este campo. Este campo no debe ser modificado en tiempo de ejecución.

- **Dirección de Stack** (StackAddress): Este registro contiene la dirección inicial del stack de software asociado la tarea. Esta dirección no debe ser modificada en tiempo de ejecución.

- **Tamaño de Stack** (StackSize): Este campo contiene el tamaño del stack de software de la tarea. Es de vital importancia para el buen funcionamiento del sistema multitareas, ya que por medio de este campo se verifica el desbordamiento de stack, conllevando la inmediata suspensión de la tarea.

- **Identificador de Tarea** (TaskID): En este campo se establece el identificador de la tarea. Todas las tareas deben tener identificadores diferentes para evitar problemas de concurrencia entre el kernel y las mismas y no pueden ser modificados en tiempo de ejecución.

- **Prioridad de Tarea** (Priority): Aquí se establece la prioridad de ejecución de la tarea. Todas las tareas deben tener una prioridad dada, pudiendo existir más de una

tarea con la misma prioridad. En el caso de que coexistan 2 o más tareas con la misma prioridad y que estén activas, el sistema multitareas se encargará de repartir los tiempos de ejecución de cada tarea aplicando la política de **Round Robin**, en donde se reparte por tiempos iguales la ejecución de las tareas, siendo ejecutadas en tiempos iguales de un periodo de tick, cadencia bajo la cual periódicamente interviene el kernel para realizar los cambios de contexto correspondientes. Este campo puede ser accedido o modificado por medio de las funciones **GetPriority** y **SetPriority** respectivamente, permitiendo manipular las prioridades de las tareas en tiempo de ejecución. La prioridad de una tarea será mayor mientras éste número lo sea también.

- **Eventos Esperados** (EventWaited): En este campo se registran los eventos por los cuales una tarea se pone en espera. Posteriormente, el Scheduler o administrador determina su activación al señalizarse la ocurrencia de al menos uno de los eventos. Por medio de la función **WaitEvent** se puede programar una tarea para suspender su ejecución a la espera de uno o más eventos. Cada tarea sólo puede programarse para esperar por sus propios eventos y no podrá colocar a ninguna otra tarea en espera de ningún evento. Estos eventos son señalizados por medio de máscaras de bits bajo valores de potencias de 2: 1, 2, 4,… hasta el valor 8000 hexadecimal. Bajo este formato y considerando que este es un campo de 2 bytes, cada tarea puede señalizar hasta 16 eventos diferentes.

- **Eventos Recibidos** (EventReceived): Este campo registra la ocurrencia de un evento señalizado por el administrador de tareas o por cualquier otra tarea mediante la función **SetEvent** Los eventos pueden ser activados por otras tareas o mediante **alarmas temporales** y **contadores**. La función **GetEvent** reporta los eventos ocurridos para una determinada tarea, es decir, devuelve el campo de esta tabla asociado a una tarea específica. Cada tarea podrá tener asociados hasta un máximo de 16 eventos correspondientes a los bits de una palabra de 2 bytes. Estos eventos son

señalizados por medio de máscaras de bits bajo valores de potencias de 2: 1, 2, 4,… hasta el valor 8000 hexadecimal.

- **Estado de la Tarea** (State): En este campo se almacena el **estado** de cada tarea (nibble bajo). El Scheduler modifica el campo de estado de cada tarea, de tal manera que para colocar una tarea en espera (**WAITING=0x08**) o ejecución (**RUNNING=0x04**) coloca los valores **WAITING+READY=0x0A** y **RUNNING+READY=0x06** respectivamente. Para suspender una tarea (**SUSPENDED**), el Scheduler limpia el nibble bajo del registro correspondiente (State=**0x00**) y para colocarla como lista para ejecución (**READY**) coloca el valor **0x02**. Contiene además el número de veces acumuladas que ha sido llamada una tarea a activarse (nibble alto) por medio de la función **ActivateTask** antes de que ocurra una suspensión por medio de la función **TerminateTask**. La función **ActivateTask** incrementará el contador de activaciones acumuladas mientras que **TerminateTask** se encargará de decrementar este contador y reiniciar la tarea sin suspenderla en caso de que existan activaciones acumuladas y, en caso contrario, suspender la tarea definitivamente cuando ya no existan más.

- **Tipo de tarea** (Type): En este campo se define el tipo de función que ejerce la tarea dentro del sistema. Este es un campo meramente informativo, sin ninguna repercusión en el funcionamiento de la tarea. Este campo puede tomar los siguientes valores:

  - **BASIC** (0x02): Tarea básica del sistema.
  - **EXTENDED** (0x01): Tarea extendida del sistema.

Existe un identificador de ROUND_ROBIN (0x04) que el Scheduler añade o sustrae de este campo sin modificar los otros anteriormente mencionados. Este identificador le permite determinar al Scheduler si, en dado caso, la tarea en cuestión

debe correr en tiempo compartido con alguna otra de su misma prioridad bajo el modo Round Robin.

- **Contador de Tiempo** (Time): Este campo pretende ser de uso futuro del sistema. Entre otras posibles funciones, en éste se almacenará la cuenta o tiempo de alarma asociada a la tarea en cuestión cuando haya sido programada.

- **Copia de Estado del Kernel** (kernelState_copy): En este campo almacena una copia temporal del estado del kernel cuando se ejecuta una rutina de servicio de interrupción (ISR). Como media de protección, las funciones **EnterISR()** y **LeaveISR()** han sido creadas para efectuar la carga y restauración del estado del kernel durante una interrupción y deben ser colocadas al principio y al final de la rutina de servicio.

- **Siguiente Tarea** (next): Puntero a la siguiente tarea del sistema con una prioridad inmediatamente inferior. Este campo es de uso exclusivo del kernel para determinar el orden de prioridad de las tareas del sistema.

Cada tarea se describe a sí misma con las propiedades definidas en esta tabla. El sistema multitareas se encargará de gestionar la activación o suspensión de tareas de acuerdo a las prioridades asignadas, los eventos en espera o el estado de cada una cada vez que se ejecute la rutina de interrupción de Timer 1, cuando se alteren prioridades, estados o se indiquen o esperen eventos en el sistema. La Figura 23 muestra un bosquejo de la organización de las tareas del sistema.

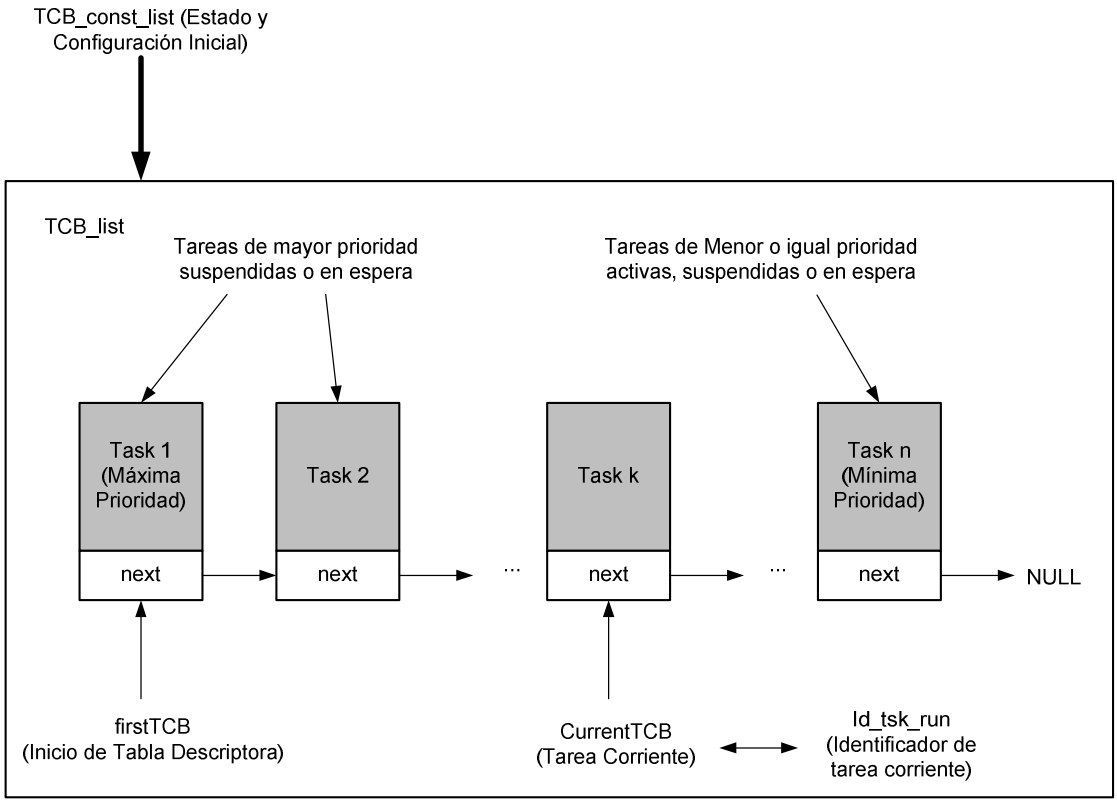


Figura 23: Funcionamiento de la Tabla Descriptora de Tareas.

2.3. Manejo de Eventos

El manejo de eventos constituye una característica esencial en la dinámica de cualquier sistema multitareas, permitiendo la puesta en espera de cualquier tarea de uno o varios eventos simultáneos. Para el manejo de eventos se destacan los siguientes tipos de variables:

- **EventMaskType**: Tipo de variable que indica los eventos en espera o señalizados por las tareas. Son equivalentes a números de 8 bits sin signo, tomando en cuenta que los mismos deben ser declarados como máscaras de bits independientes, es decir, estas máscaras deben tomar los valores hexadecimales **0x01,**

0x02, 0x04,… hasta 0x8000. Debido a esta disposición, cada tarea podrá manejar un máximo de 16 eventos, cada uno correspondiente a un bit de una palabra de 16.

- **EventMaskRefType**: Puntero a una variable de tipo **EventMaskType**.

Se describen a continuación las funciones que desempeñan la gestión de eventos del sistema:

- **SetEvent (TaskType TaskID, EventMaskType Mask)**: Señaliza la ocurrencia de un evento declarado bajo la máscara **Mask** esperado por la tarea con el identificador **TaskID** en la Tabla Descriptora de la Tareas.

- **ClearEvent (EventMaskType Mask)**: Limpia la máscara del evento declarado bajo la máscara **Mask** recibido por la tarea en progreso en su Tabla Descriptora.

- **GetEvent (TaskType TaskID, EventMaskRefType Mask)**: Obtiene las máscaras de los eventos recibidos por la tarea de identificador **TaskID** de la tabla de Ocurrencia de Eventos en la variable cuyo puntero es **Mask.**

- **WaitEvent (EventMaskType Mask)**: Coloca a la tarea corriente en espera por el evento o eventos cuyas máscaras vienen dadas por **Mask**, señalizándolo convenientemente en la tabla de Señalización de Espera de Eventos. Esta función elimina cualquier evento cuya espera que se haya señalizado anteriormente.

La Figura 24 describe el funcionamiento de dos tareas ante la espera y la ocurrencia de un evento, destacando la programación de su espera y la señalización de la posterior ocurrencia del mismo.

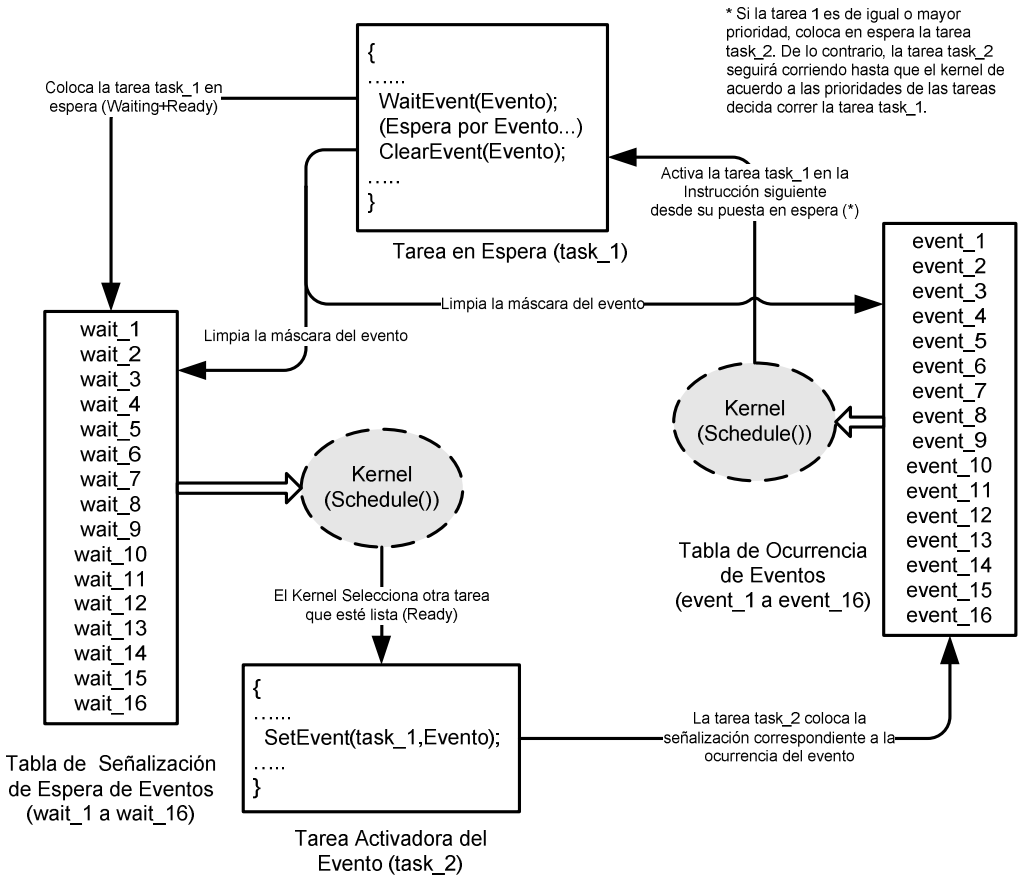


Figura 24: Descripción de la dinámica del sistema multitareas en el manejo de eventos.

En la gráfica arriba presentada, función **WaitEvent** coloca a una tarea en espera (**task_1**) ante la ocurrencia del evento cuya máscara se pasa como parámetro (**Evento**). Posteriormente, alguna otra tarea (**task_2**) por medio de la función **SetEvent** o por medio de una alarma, colocará en la tabla de ocurrencia de eventos la señalización correspondiente activando el bit correspondiente indicado por la máscara **Evento**. El kernel por medio de la función **Schedule()** seleccionará dependiendo de la prioridad de cada tarea la activación de una de ellas, dentro de las cuales estará **task_1**. Finalmente, la función **ClearEvent** se encargará de limpiar las banderas asociadas al evento ocurrido en ambas tablas.

La arquitectura del sistema multitareas también permite que una tarea espere por más de un evento a la vez, despierte de la ocurrencia de uno o más de ellos y seleccione la acción a realizar de acuerdo al evento que la haya despertado. A continuación en la Figura 25 se muestra un código simplificado para programar una tarea ante este tipo de situaciones.

```
EventMaskType EventQuerry;

{
……
  WaitEvent(Evento_1 | Evento_2);

  GetEvent(Task_ID,&EventQuerry);

  if (EventQuerry & Evento_1){
    ClearEvent(Evento_1);
    ….. (Código asociado a la ocurrencia de Evento_1)
  }

  if (EventQuerry & Evento_2){
    ClearEvent(Evento_2);
    ….. (Código asociado a la ocurrencia de Evento_2)
  }
…..
}
```

Figura 25: Programación de una tarea ante la espera de más de un evento.

Para este tipo de situaciones, se debe declarar una variable de tipo **EventMaskType** para leer la máscara o máscaras asociadas a los eventos ocurridos por medio de la función **GetEvent**. En esta función, el parámetro **Task_ID** normalmente se suele pasar como el identificador de la tarea corriente, aunque puede ser el de cualquier otra tarea. En ese caso no se deben colocar las instrucciones **ClearEvent** para no afectar la dinámica del sistema.

2.4. Manejo de Alarmas

El sistema operativo multitareas PICos permite la programación de alarmas de tiempo y contadores capaces de señalizar eventos a cualquier tarea o incluso sacarla de una suspensión.

Las estructuras de datos asociadas a contadores y alarmas se describen a continuación:

- **TickType**: Tipo de variable representativa de cuentas o umbrales de alarma. Equivale a un número entero de 16 bits sin signo.

- **TickReftype**: Puntero a variable de tipo **TickType**.

- **AlarmType**: Tipo de variable para identificadores de alarma. Equivale a un número entero de 8 bits sin signo.

- **AlarmRefType**: Puntero a variable de tipo **AlarmType**.

- **AlarmBaseType**: Estructura de parámetros internos de una alarma.

- **AlarmBaseRefType**: Puntero a variable de tipo **AlarmBaseType**.

- **Counter**: Estructura de parámetros para contadores y timers de alarma.

- **RefCounter**: Puntero a variable de tipo **Counter**.

- **AlarmObject**: Estructura de Parámetros de estado y control de una alarma.

Toda estructura de datos de tipo **AlarmBaseType** contiene los parámetros y umbrales de funcionamiento internos de un contador cualquiera del sistema. Esta estructura de datos está compuesta por los siguientes componentes:

- **MaxAllowedValue**: Variable de tipo **TickType** que contiene el valor de la cuenta máxima permitida que puede ser programada. Toda cuenta a ser programada en un contador deberá ser menor o igual a este valor.

- **ticksPerBase**: Variable de tipo **TickType** que registra el número de cuentas o ticks que serán añadidos cada vez que el contador sea incrementado. Hasta los momentos no ha sido implementada en la programación, fijando los incrementos de todos los contadores en uno (1).

- **minCycle**: Variable de tipo **TickType** que registra la cuenta mínima a la que puede ser programado cualquier contador.

Las estructuras de datos de tipo **Counter** contienen los siguientes elementos:

- **Base**: Variable de tipo **AlarmBasetype** que presenta los umbrales de los valores de cuenta con los cuales un contador cualquiera podrá ser programado.

- **CounterValue**: Variable de tipo **TickType** que almacena la cuenta actual del contador. Este campo comprende valores entre 0 y 65535 y actúa de forma cíclica, es decir, al incrementarse una cuenta cuyo valor es el máximo posible (65535) ésta pasa a ser cero (0) y se pierde el acarreo producto del incremento realizado.

- **Tick**: Variable de tipo **TickType** aún no implementada.

Dentro de las estructuras de datos correspondientes a esta parte del sistema multitareas, las entidades de mayor funcionalidad dentro del mismo son las alarmas. Éstas permiten administrar y controlar lapsos de tiempo que deban ser programados o cuentas sucesivas de eventos dentro de cualquier aplicación para activar tareas o señalizar cualquier evento aunque, sin embargo, las mismas deban siempre tener un

contador asociado. La Figura 26 muestra un esquema de la estructura de una alarma del sistema.

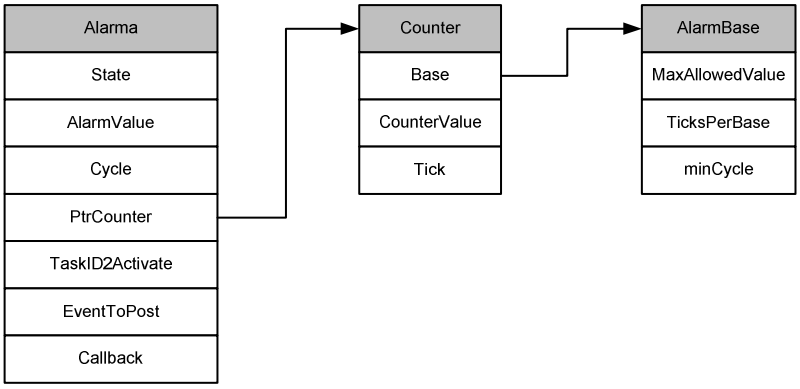


Figura 26: Estructura General de una Alarma.

Las estructuras de datos de alarmas **AlarmObject** se componen de los siguientes elementos:

- **State**: Variable de 8 bits sin signo que reporta si una alarma está encendida (ON=1) o apagada (OFF=0).

- **AlarmValue**: Variable de tipo **TickType** que almacena el valor que deberá alcanzar el contador asociado a una alarma para ejecutar la acción correspondiente, ya sea señalizar un evento o activar una tarea.

- **Cycle**: Variable de tipo **TickType** que contiene el número de cuentas necesarias para la activación de la alarma cuando la misma es programada para que ocurra cíclicamente. Cuando este campo se programa como cero (0) la alarma se programará para ser activada una sola vez y será apagada.

- **PtrCounter**: Variable de tipo **RefCounter** que contiene la dirección del contador asociado a la alarma, el cual almacenará la cuenta o lapso de tiempo

transcurrido para la ocurrencia del evento de alarma. Varias alarmas pueden tener asociadas el mismo contador.

- **TaskID2Activate**: Campo que contiene el identificador de la tarea a ser activada.

- **EventToPost**: Campo que contiene la máscara de Evento a ser señalizado. Cumple con las mismas exigencias que para la declaración de cualquier evento del sistema. En caso de que este campo sea nulo, el sistema multitareas asumirá que la tarea ha sido suspendida y en vez de la señalización de evento el mismo procederá a activarla.

- **CallBack**: Rutina asociada sin pase ni recepción de parámetros que será ejecutada cada vez que se dé la condición de alarma. Para inhabilitar esta opción se deberá colocar un valor nulo en este campo.

Todo contador debe ser declarado en la lista de contadores **Counter_list** del archivo **taskdesc.c**. Esta lista de contadores permitirá llevar la cuenta de los eventos que ocurran en el sistema que así lo amerite, mediante el uso de la instrucción **IncCounter**, pasando como parámetro el índice del contador de esta lista que se desee incrementar.

A su vez, toda alarma debe ser declarada en la lista de alarmas **Alarm_list** del archivo **taskdesc.c**, su identificador debe ser declarado en el archivo **define.h** y debe tener un contador asociado para poder funcionar. Las alarmas de esta lista son las encargadas de señalizar a las tareas por medio de la comparación entre la cuenta almacenada y los umbrales programados la ocurrencia de un evento de límite de cuenta y permitir que las tareas ejecuten las acciones correspondientes.

Una alarma también puede ser programada para activarse por lapsos de tiempo, colocando en el campo **PtrCounter** de la alarma respectiva la dirección del contador de kernel (&Counter_kernel). Este contador se incrementa cada periodo de tick, actualizando posteriormente el estado de las alarmas, la señalización de eventos asociados para finalizar ejecutando el gestor de tareas o Scheduler, lo cual genera un efecto de programación de un timer. El tiempo para el cual la alarma haya sido programada será siempre múltiplo del periodo de tick del sistema multitareas. En la Figura 27 se muestra un diagrama de interrelación entre alarmas y contadores.

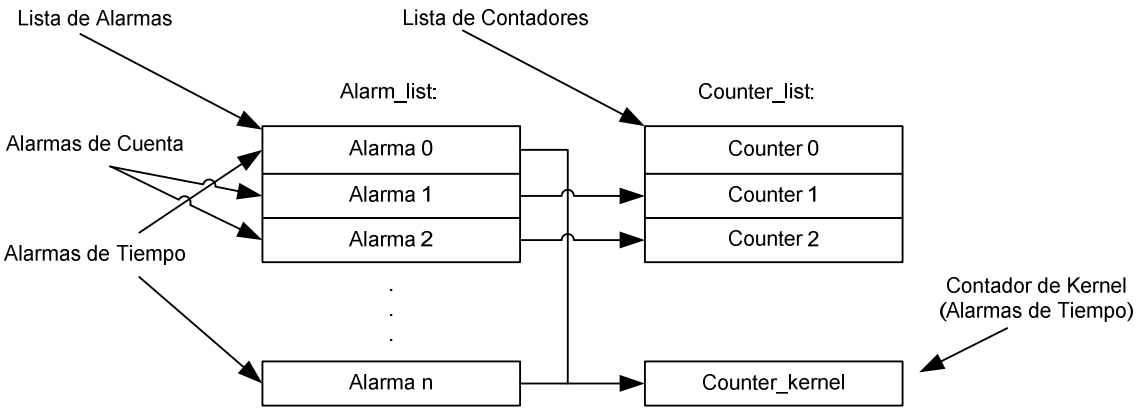


Figura 27: Diagrama de interrelación entre alarmas y contadores.

Las funciones para el manejo de alarmas del sistema multitareas son las siguientes:

- **SetRelAlarm (AlarmType ID, TickType increment, TickType cycle)**: Programa la alarma de identificador **ID**, especificando el tiempo **increment** para un primer tiempo de espera y un tiempo **cycle** para una activación cíclica de la tarea asociada. Ambos tiempos se basan en ciclos de kernel (Ticks).

- **SetAbsAlarm (AlarmType ID, TickType start, TickType cycle)**: Programa la alarma de identificador **ID**, especificando el tiempo **start** para un primer tiempo de espera y un tiempo **cycle** para una activación cíclica de la tarea asociada. Ambos tiempos se basan en ciclos de kernel (Ticks).

- **GetAlarm (AlarmType ID, TickRefType Tick)**: Retorna en la variable apuntada por **Tick** los ciclos de kernel que restan para que la alarma de identificador **ID** sea activada.

- **CancelAlarm (AlarmType ID)**: Desactiva la alarma de identificador **ID**. La alarma detendrá su funcionamiento y ningún evento será señalizado por el kernel a ninguna tarea.

- **GetAlarmBase (AlarmType ID, AlarmBaseRefType Info)**: Retorna la dirección de la lista de parámetros base de la alarma de identificador **ID** en la variable **Info**.

- **IncCounter (AlarmType ID)**: Incrementa la cuenta de la alarma de identificador **ID** cuándo ésta sea programada en modo contador.

- **GetCounterValue (AlarmType ID, TickRefType tick)**: Retorna en la variable apuntada por **Tick** el número de cuentas que lleva la alarma de identificador **ID** cuándo ésta se programa en modo contador.

Para la programación de la alarmas deben usarse las instrucciones **SetRelAlarm** o **SetAbsAlarm**. Ambas funciones permiten programar la alarma pasada como parámetro para un primer periodo de tiempo y posteriormente para una activación cíclica de la misma cada vez que el contador asociado alcance un valor prefijado, el cual será invariante a menos que la alarma sea reprogramada. La

diferencia entre ambas funciones radica en que la primera programará la alarma para ser activada por primera vez después de ocurrido un cierto número de pasos, mientras que la segunda la programará cuando el contador asociado alcance la cuenta especificada, sin importar que haya sobrepasamiento del contador. Toda alarma debe ser cancelada por medio de la instrucción **CancelAlarm** antes de ser reprogramada en el caso de que ésta haya sido habilitada.

A continuación en las figuras 6.2 y 6.3 se muestran dos diagramas de flujo que muestran los métodos implementados para gestionar las alarmas de tiempo y de contadores respectivamente. Es de interés recordar que la gestión de alarmas de tiempo es realizada internamente en la rutina de servicio de interrupción de Timer 1 (interrupción del Kernel), mientras que las de contadores es realizada internamente por la función **IncCounter**, siendo pasado como parámetro para ésta última el identificador de alarma correspondiente**.** La Figura 28 y la Figura 29 muestran el tratamiento dado por el kernel PICos para las alarmas de tiempo y contadores en general.

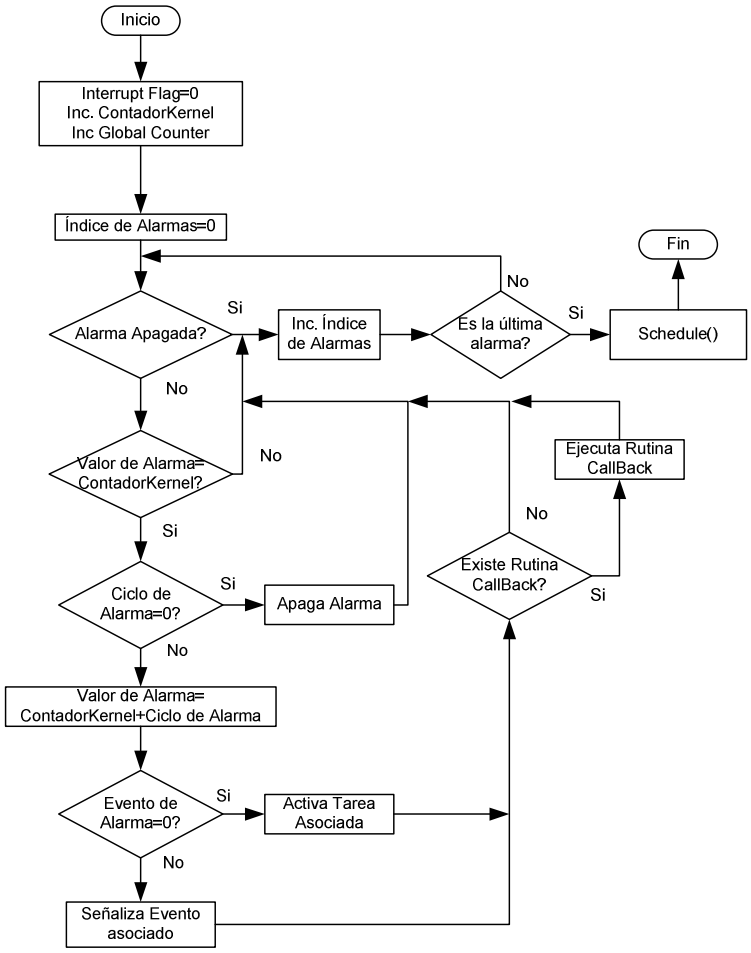


Figura 28: Diagrama de Flujo de gestión de alarmas de tiempo basadas en el Contador de Kernel.

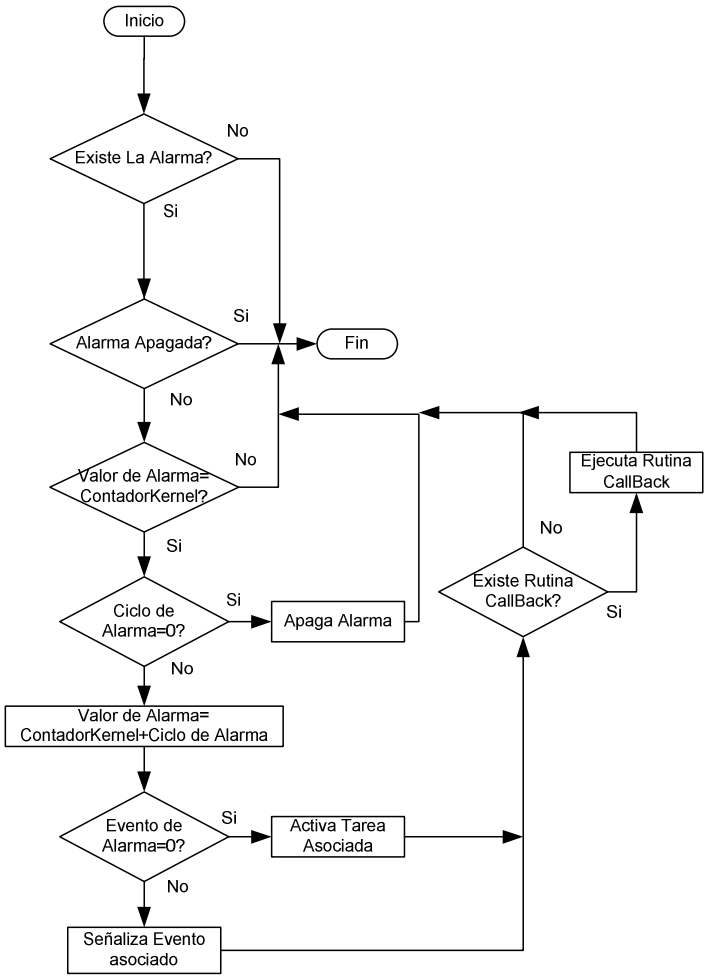


Figura 29: Diagrama de Flujo de verificación de condición de alarma de un contador al ser incrementado.

3. Mensajes entre Tareas (MailBoxes)

El Sistema Operativo Multitareas PICos carece de un medio de envío de mensajes formal. La complejidad de programación y funcionalidades del teléfono público GSM aquí implantado es alta debido a los múltiples procesos que se llevan en paralelo. En muchos casos se hace necesario condicionar unas tareas al estado de otra mediante algún tipo de sincronización. Otros sistemas operativos multitareas tiempo real disponen de estas herramientas, pero el sistema PICos carece de ellas.

Por tal motivo, se le añadió un conjunto de rutinas para administrar los mensajes entre tareas, partiendo de que éstas pueden ser activadas por medio de eventos. Por medio de dos (2) eventos dedicados a la señalización de mensajes, se logra que las tareas detecten mensajes recibidos y puedan entonces ejecutar los procesos que le sean asignados por alguna otra tarea. El evento **MailEvent** señaliza a una tarea que un mensaje ha sido recibido, mientras que el evento **MailAckEvent** le permite saber al destinatario que el mensaje ha sido recibido con éxito.

Cada mensaje contiene un conjunto de parámetros que recibe la tarea ejecutora para poder llevar a cabo el proceso asignado por éste. El elemento intermediario para que la recepción del mensaje y el pase de parámetros se lleve a cabo es llamado **MailBox** y se compone de una lista de estructuras que contienen un identificador (MailBoxID), una tarea fuente, una tarea destino, una bandera de actividad que señala el estado del mensaje (recibido o no recibido), el comando que se debe ejecutar y los parámetros intrínsecos asociados al mensaje. Un comando puede carecer de parámetros o puede tener más de un parámetro, dependiendo de las exigencias del comando solicitado. La Figura 30 esquematiza una estructura de maiboxes.

MailBox ID	Tarea Fuente	Tarea Destino	Bandera de Actividad	Comando	Parámetros
MailBox1	Tarea 1	Tarea 2	0	Comando 1	Parámetros 1
MailBox2	Tarea 2	Tarea 1	1	Comando 2	Parámetros 2
MailBox3	Tarea 3	Tarea 2	0	Comando 3	Parámetros 3
⋮	⋮	⋮	⋮	⋮	⋮
MailBoxN	Tarea X	Tarea Y	1	Comando C	Parámetros N

Figura 30: Estructura de MailBoxes del Sistema.

Existen un conjunto de funciones determinado que permiten manejar los eventos de mensaje y el pase de parámetros correspondiente. La función **WaitMail** programa a una tarea para esperar por un evento de mensaje con la posibilidad de esperar por algún otro evento externo (ExternalEvent). La función **SendMail** carga en un MailBox determinado los parámetros de un mensaje, congela el buzón activando la bandera de actividad y señaliza a la tarea destinataria el evento de mensaje (MailEvent). La tarea destinataria puede verificar por medio de la función **TestMail** cual fue el buzón activado, recoger los parámetros y señalizar la recepción bajando esta bandera de actividad por medio de la función **AckMail**. En dado caso, si la tarea solicitadora necesita esperar la confirmación de mensaje recibido, puede ser puesta en espera por medio de la función **WaitAckMail**. La Figura 31 muestra el proceso de envío y recepción de mensajes entre tareas.

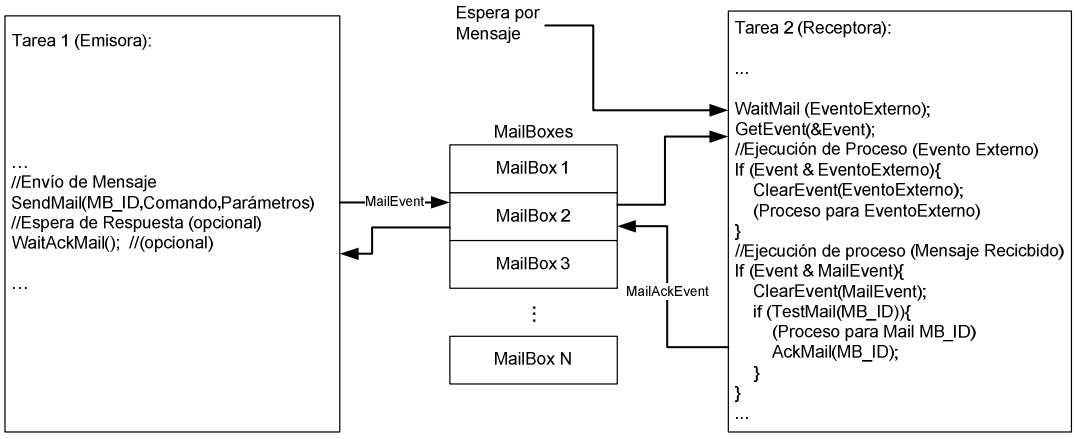


Figura 31: Proceso de envío y recepción de mensaje entre tareas.

4. Tablas Descriptoras de Dispositivos (DDT)

Dentro del sistema multitareas existen diversos componentes de hardware que entran en juego y que interactúan entre sí para conformar así una aplicación. Para poder independizar el comportamiento de cada dispositivo se ha considerado una metodología de trabajo en la cual se declaran un conjunto estructurado de datos y rutinas que controlan a todos y cada uno de los dispositivos físicos del sistema. Para ello, se emplea el modelo de capas EPA y la conformación de objetos asociados a los dispositivos entra/salida. Estos objetos para dispositivos entrada/salida se conformaron a partir de lo que denominaremos DDT (tablas descriptoras de dispositivos o "device data table").

Las DDT almacenan todas las variables de estado y de control de un dispositivo físico determinado. Estas estructuras cumplen con las características fundamentales de un objeto: Identificador, Propiedades y Métodos. Toda DDT presenta un header o cabecera. Este header contiene un identificador de dispositivo (DeviceID), tipo de dispositivo (DeviceType), un campo de banderas y de Control y Estado, los cuales pueden ser diferentes entre dispositivos, un campo de código de error, la dirección del vector de funciones del dispositivo (handler) y una cola de parámetros que son usados por el handler cuando se ejecuta una función determinada. El campo de Estado presenta las banderas de encendido (On), ocupación (Busy) y el identificador de tarea propietaria (CallerID), pudiendo contener otros campos adicionales. La Figura 32 muestra una estructura detallada de una Tabla Descriptora de Dispositivo.

A través de esta filosofía de programación se han diseñado drivers para buses de comunicación UART, SPI, I2C, Pantallas alfanuméricas, Reloj de Tiempo Real interno del microcontrolador, entradas y salidas digitales y convertidor A/D interno.

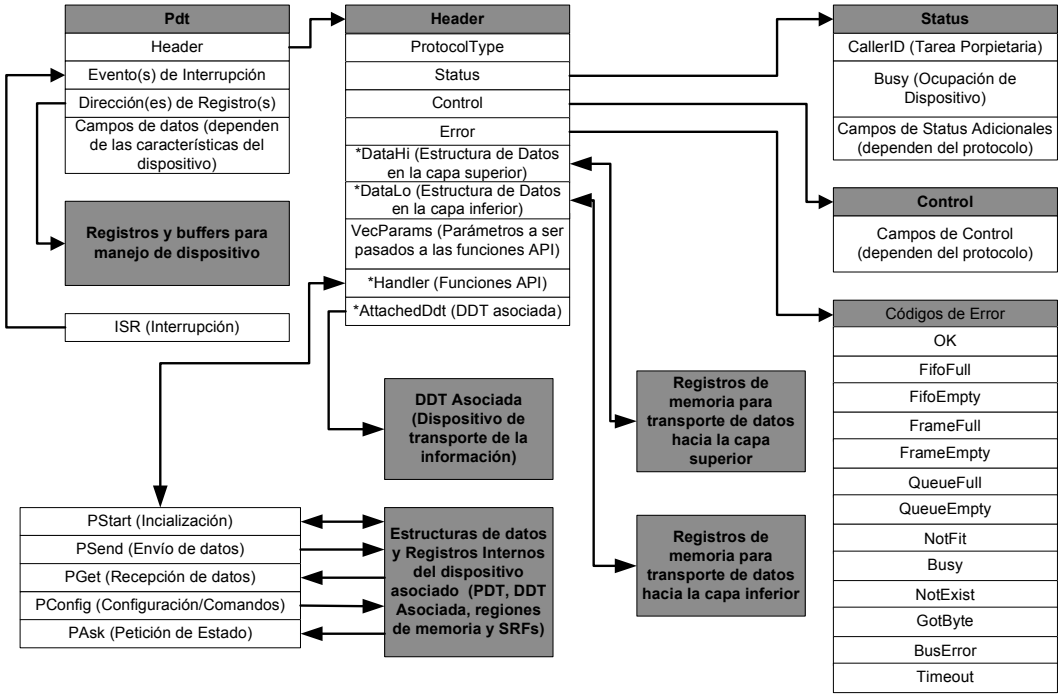


Figura 32: Estructura General de una Tabla Descriptora de Dispositivo.

Todo handler o manejador de una DDT contiene las siguientes funciones:

- Start(&DDT): Función de inicialización de dispositivo. Esta función inicializa los registros, interrupciones y puertos del dispositivo para su correcto funcionamiento. No existe pase de parámetros para esta función.

- Send(&DDT, &Trama, Modo): Función encargada de la transmisión de datos. Esta función pasa como parámetros la dirección de la trama de datos a enviar y el formato o modo de operación bajo el cual son enviados estos datos.

- Get(&DDT, &Trama, Modo): Función encargada de la recepción de datos. Esta función pasa como parámetros la dirección de destino de los datos recibidos y el formato o modo de operación bajo el cual deben ser recibidos estos datos. En algunos

casos la función del campo Modo puede variar, pudiendo ser un evento, identificador o contador.

- Config(&DDT, Propiedad, Valor): Función de configuración de dispositivo. Esta función permite modificar una o más propiedades de la DDT, afectando su comportamiento. Generalmente se usa para configurar registros de control o estado de dispositivos.

- Ask(&DDT, Propiedad): Función de solicitud de estado de dispositivo. Esta función devuelve el estado o modo de configuración del dispositivo almacenado en su DDT.

- Take(&DDT): Función de solicitud de recurso. Esta función verifica si el dispositivo está siendo usado por alguna tarea y en caso de estar libre asocia el dispositivo a la tarea que hizo el llamado.

- Release(&DDT): Función de liberación de recurso. Esta función libera el recurso anteriormente solicitado. Sólo se podrá liberar el recurso si la tarea que libera el recurso es la misma que la solicita.

5. Tablas Descriptoras de Protocolo (PDT)

Para el manejo de protocolos de comunicación se ha dispuesto de otra clase de objetos con un conjunto de estructuras de datos llamadas Tablas Descriptoras de Protocolo (PDT o “Protocol Descriptor Table”), las cuales se concentran las funciones y parámetros de los protocolos mediante los cuales el sistema se comunica. La estructura de este descriptor es muy parecida a la de una DDT, con la salvedad de que no contiene recursos de hardware en sí mismo, sino que éste los solicita para

llevar a cabo el proceso correspondiente. Una PDT representa la capa siguiente en el modelo EPA, la cual evidentemente se encarga de procesos de mayor nivel de abstracción en lo que a flujo de información se refiere.

Todo conjunto de datos que llegan por un puerto en general tienen una estructura donde sólo una parte es el dato o información requerida. En general dicho conjunto viene acompañado de indicador de cantidad de datos, tipo de datos, los datos en sí mismos (información), validación de error (por ejemplo código CRC o checksum). Para la DDT son solo datos, la PDT se encarga de analizar la estructura de los mismos, validarla y extraer la información buscada sin importar la DDT empleada (UART, SPI, I2C, USB, etc). Toda PDT tiene un campo donde se le configura la DDT asociada. La generalidad de parámetros de enlace permite tratar tramas de datos provenientes de cualquier dispositivo con tan sólo cambiar la DDT asociada. Diferentes PDT permiten implantar diferentes protocolos de comunicaciones (MODBUS, Fieldbus, CAN, protocolos propietarios, etc).

Toda PDT presenta un header o cabecera. Este header contiene un identificador (ProtocolType), un campo de banderas y de Control y Estado, los cuales pueden ser diferentes entre dispositivos, un campo de código de error, la dirección del vector de funciones del dispositivo (handler) y una cola de parámetros que son usados por el handler cuando se ejecuta una función determinada. El campo de Estado presenta una bandera de ocupación (Busy) y un identificador de tarea propietaria (CallerID), pudiendo contener otros campos adicionales. La Figura 33 muestra una estructura detallada de una Tabla Descriptora de Protocolo

La implementación de este tipo de estructuras ha servido para la programación de protocolos de comandos AT, manejo de memoria EEPROM y lector de tarjetas magnéticas.

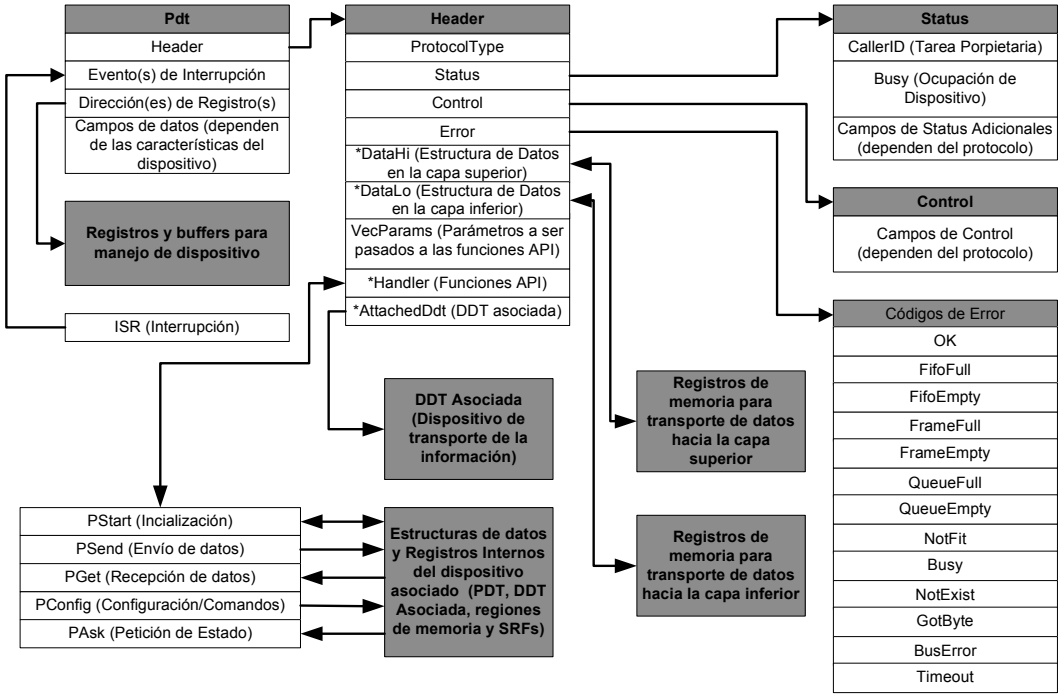


Figura 33: Estructura General de una Tabla Descriptora de Protocolo.

El handler de toda PDT contiene las siguientes funciones:

- PStart(&PDT): Función de inicialización de parámetros de protocolo. Esta función inicializa los registros de memoria, toma de recursos e interrupciones necesarias para el correcto funcionamiento del protocolo. No existe pase de parámetros para esta función.

- PSend(&PDT, &Trama, Modo): Función encargada de la codificación de datos a transmitir por medio de la DDT asociada. Esta función pasa como parámetros la dirección de la trama de datos a enviar y el formato o modo de operación bajo el cual son enviados estos datos.

- PGet(&PDT, &Trama, Modo): Función encargada de decodificar los datos recibidos por la DDT asociada. Esta función pasa como parámetros la dirección de

destino de los datos recibidos y el formato o modo de operación bajo el cual deben ser decodificados estos datos. En algunos casos la función del campo Modo puede variar, pudiendo ser un evento, identificador o contador.

- PConfig(&PDT, Propiedad, Valor): Función de configuración de protocolo. Esta función permite modificar una o más propiedades de protocolo de la PDT, afectando su comportamiento. Generalmente se usa para configurar registros de control o estado o incluso para generar tramas de comunicación con fines de establecer las configuraciones deseadas para dispositivos que se comuniquen mediante el protocolo descrito en la PDT.

- PAsk(&PDT, Propiedad): Función de solicitud de parámetros de protocolo. Esta función devuelve parámetros de estado o modo de configuración del protocolo almacenado en la PDT. A su vez, esta función puede generar tramas de comunicación para solicitar estado de algún dispositivo que se comunique mediante el protocolo descrito en la PDT.

6. Estructuras Utilitarias

6.1. Estructuras de Datos First In - First Out (FIFO)

Una estructura First In – First Out almacena datos (bytes) de forma consecutiva, de tal manera que sean extraídos en la misma secuencia de cómo fueron guardados. Este tipo de estructuras permite el anidamiento de datos que deben ser procesados en secuencia sin perder la información ni el orden entre los mismos como por ejemplo una secuencia de datos ingresados por teclado.

Para establecer una estructura que presente este comportamiento se dispone de un buffer para almacenar los datos que se ingresan (vector **Data**) de tamaño fijo,

en conjunto con dos punteros, uno de los cuales permite llevar la secuencia de los caracteres que ingresan al FIFO (**P_in**) y otro que permite la extracción de los mismos (**P_out**) guardando la misma secuencia. Adicionalmente, un contador (**Count**) permite llevar la cuenta de la cantidad de datos válidos almacenados en el FIFO. La Figura 34 muestra un bosquejo de la estructura de datos FIFO y los campos que la contemplan.

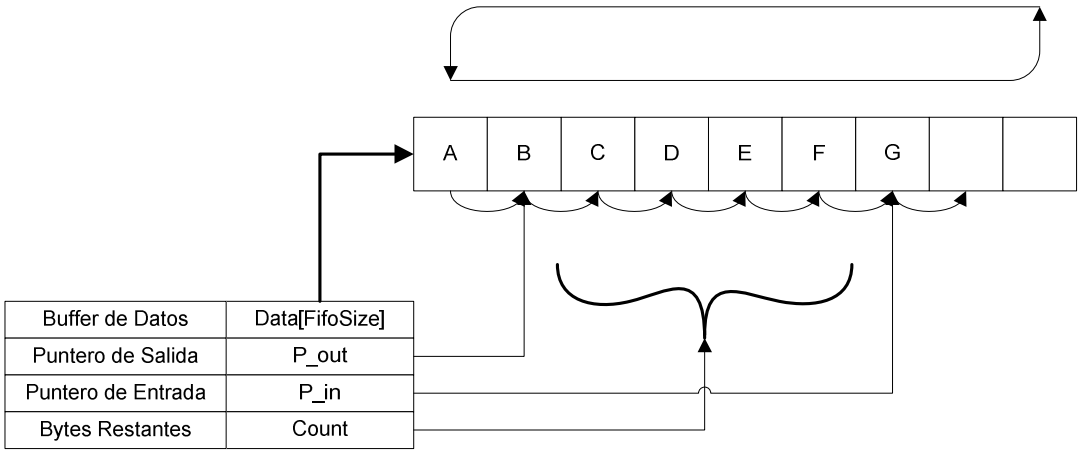


Figura 34: Esquema General de una estructura First In-First Out (FIFO).

El puntero de entrada de datos **P_in** se incrementa cuando se almacena un dato, mientras que el puntero de salida de datos **P_out** se incrementa al extraerse un dato. En el caso de que alguno de estos punteros alcance el fin del buffer de datos, el mismo se inicializa de nuevo para apuntar al primer campo del mismo, generando así un efecto de carrusel. El contador de datos **Count** se incrementa o decrementa cada vez que se guarda o extrae un dato del FIFO. Por medio de este contador se puede determinar si el FIFO se ha llenado o está vacío, casos en los cuales ambos punteros de entrada y salida toman el mismo valor, evitando así ambigüedades en el comportamiento de la estructura. En caso de que el FIFO se llene o se vacíe, la estructura no soportará el ingreso o extracción de datos dependiendo del caso, para lo

cual las funciones y procedimientos que manejan estas estructuras arrojarán los códigos de error correspondientes.

6.2. Estructuras de Trama de Datos (Frame)

Toda estructura de Trama (Frame) almacena una secuencia de datos provenientes de un proceso de comunicación cualquiera entre capas de un protocolo o medio de transporte de hardware o software.

Este tipo de estructuras consisten en un buffer de datos de tamaño fijo (vector **Data**) acompañado de un índice o contador (**Index**), el cual apunta al espacio siguiente a ser leído o escrito en la trama. Este contador cumple simultáneamente la función de poder registrar la cantidad de datos almacenados en una trama en el caso de que requiera ser enviada o leída por cualquier función. La punta de la trama almacenada se hallará siempre en el primer campo (byte) del buffer, a diferencia de la estructura FIFO. Esta estructura posee un campo adicional (**Next**) que permite enlazar varias tramas en secuencia en el caso de que la cantidad de información de la trama exceda el tamaño de una sola estructura. La Figura 35 muestra un esquema que describe la estructura y comportamiento de una trama de datos o Frame.

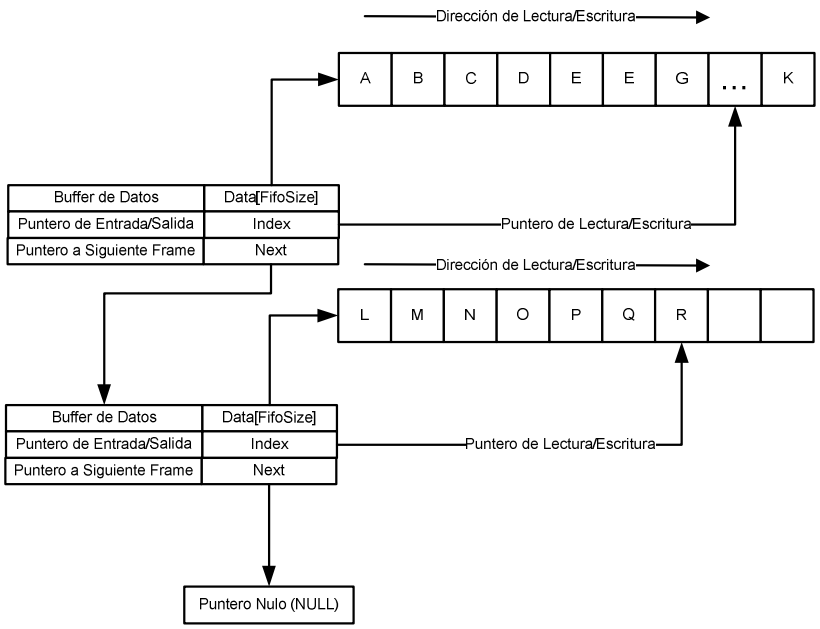


Figura 35: Esquema General de una estructura Trama (Frame).

7. Tareas del Sistema

Hasta ahora se ha mostrado la implantación básica de la plataforma de software que contempla el sistema operativo multitarea tiempo real y los diferentes drivers de dispositivos (adquisición de puntos digitales de entrada, puntos digitales de salida, puntos analógicos de entrada, uart, spi, i2c, eeprom, display, teclado, etc.). Como ya se dijo fueron realizados con la metodología de separación de capas considerando el modelo EPA y la filosofía de objetos formando las DDTs correspondientes para cada dispositivo. Ahora toca avocarse desde el punto de vista de sistemas a organizar los distintos procesos en forma estandarizada mediante el uso de Tareas que correrán paralelamente apoyadas en el sistema operativo PiCos.

La programación del Teléfono Público se divide en siete (7) tareas independientes con funciones específicas. Cada tarea dispone de dispositivos y procedimientos que ésta puede controlar para cumplir su función. A continuación se

describe cada tarea del sistema. La Figura 36 muestra un esquema de la distribución de tareas en el sistema y los enlaces correspondientes.

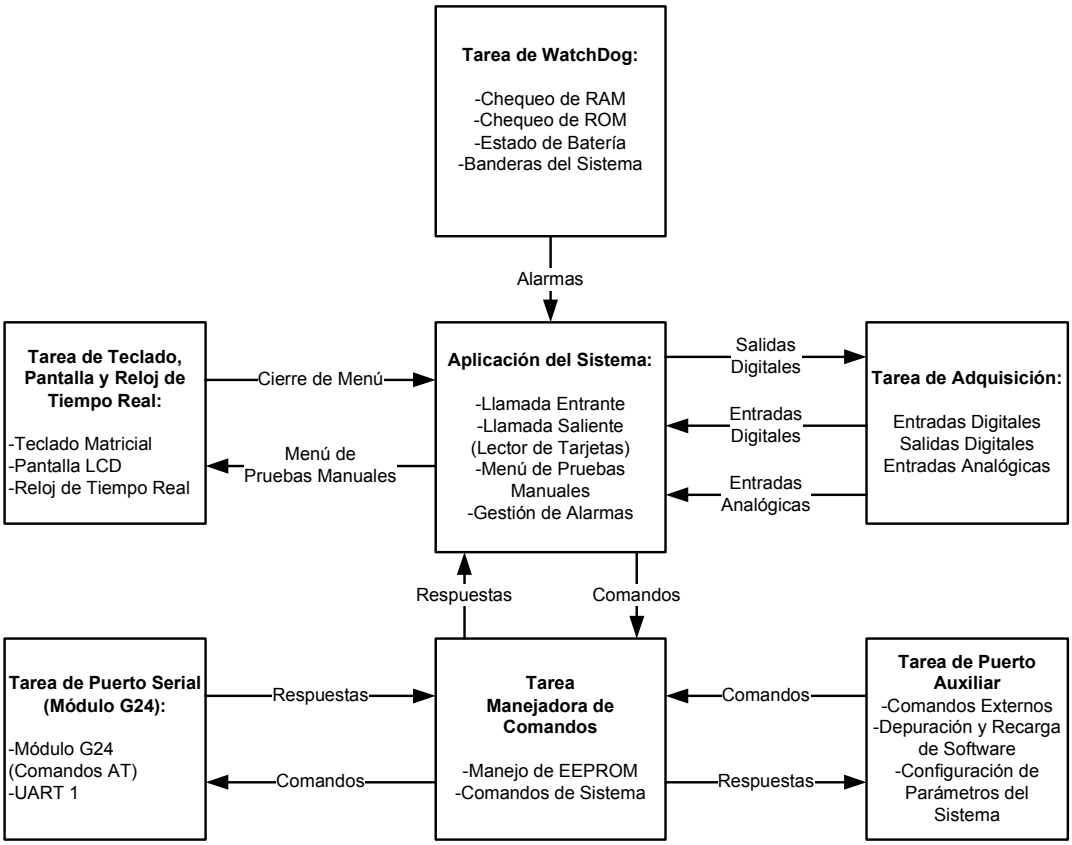


Figura 36: Diagrama de Tareas del Sistema.

7.1. Tarea de Aplicación

La tarea de aplicación es la tarea maestra del sistema, la cual mediante una máquina de estados controla el estado del teléfono durante la realización de llamadas. Por medio de mensajes dirigidos al manejador de comandos, la aplicación es capaz de delegar funciones a otras tareas.

7.2. Manejador de Comandos

La tarea manejadora de comandos permite al sistema gestionar las funciones inherentes a la aplicación, tales como la inicialización de dispositivos, gestión de eventos de llamada con la aplicación y otras tareas, activación de modos de operación del dispositivo, etc. Utiliza la DDTs de bus I2C en conjunto con la PDT de Memoria EEPROM para carga de parámetros del Teléfono.

7.3. Tarea de Adquisición de Datos

La tarea de adquisición de datos configura las entradas analógicas y digitales y las salidas digitales y actualiza su estado periódicamente a una rata de 40ms. Por medio de las DDTs de entradas digitales, Salidas Digitales y Entradas Analógicas se actualizan en bases de datos el estado de cada entrada o salida física del microcontrolador.

7.4. Tarea de Perro Guardián (Watchdog)

Esta tarea verifica el estado de las memorias ROM y RAM del microcontrolador y las alarmas del sistema. Para verificar el estado de la memoria ROM se efectúa un cálculo de CRC comparándolo con el valor calculado en una primera iteración al arrancar el sistema, mientras que para la memoria RAM se realiza una prueba de escritura/lectura. Ambos procesos se realizan periódicamente. Las alarmas del sistema se guardan en la base de datos del teléfono para ser posteriormente procesadas.

7.5. Tarea de Comunicación (Módulo GSM)

La tarea de comunicación es la encargada de interactuar con el módulo GSM Motorola G24 mediante su PDT, la cual procesa el protocolo de comandos AT bajo el cual funciona. Esta PDT toma como recurso de comunicación el puerto serial UART1 del microcontrolador. La tarea de aplicación se comunica con el módulo GSM a través de mensajes entre tareas durante los procesos de llamada saliente y entrante. La Tabla 5 enumera los comandos AT utilizados para la aplicación de teléfono público. Los comandos en negrita son propietarios de Motorola.

Comandos de interfaz UART	
&K	Controla la configuración de pines de control del puerto serial.
E	Controla el eco de datos enviados por el puerto serial.
V	Configura códigos de error en las respuestas.
Comandos de Llamada	
+CLIP	Configura la respuesta del módulo para llamadas entrantes.
+VTD	Configura duración de tonos DTMF.
+VTS	Envía una trama de tonos DTMF.
A	Atiende a una llamada entrante.
D	Realiza una llamada de voz o datos.
H	Termina la llamada activa.
Comandos de interfaz de audio	
+CALM	Configura volumen de repique.
+CLVL	Configura volumen de speaker.
+CMUT	Configura mute de microteléfono.
+MAFEAT	Configura cancelación de eco y sidetone (Modo Avanzado).
+MAMUT	Configura mute de canal activo (Modo Avanzado).
+MAPATH	Configura canal de audio activo (Modo Avanzado).
+MAVOL	Configura volumen de canal activo (Modo Avanzado).
+MMICG	Configura ganancia de microteléfono,
S94 / S96	Supresión de eco de voz / Reducción de ruido / Sidetone
Comandos de Repique	
+CALM	Control de Activación de repique.
+CRSL	Selección de tipo de repique.
+CRTT	Selección de ringtone y modo.
Comandos de red GSM	
+CGMI	Identificador de Compañía OEM: Motorola.
+CGMM	Reporte de bandas activas de la red GSM.

+CPIN	Envía el número de PIN de la tarjeta SIM GSM.
+CREG	Solicitud de estado de registro a la red GSM.
+CSQ	Reporta la potencia de la red GSM.
+GMR	Reporte de versión de firmware de Módulo G24.
+TPIN	Reporta el número de intentos restantes para enviar un PIN.
S97	Reporta la conexión física de antena.
Comandos de stack TCP-IP	
+MIPCALL	Realiza un llamado a la red para obtener una dirección IP.
+MIPCLOSE	Cierra un canal TCP-IP.
+MIPFLUSH	Limpia el stack TCP-IP.
+MIPOPEN	Abre un canal TCP-IP.
+MIPPUSH	Envía los datos del stack TCP-IP a la red GSM.
+MIPRTCP	Avisa la recepción de datos en el stack TCP-IP.
+MIPSEND	Coloca datos en el stack TCP-IP.
+MIPSETS	Configura el tamaño del stack TCP-IP.
+MIPSTAT	Avisa el estado de una comunicación TCP-IP.
+MPING	Inicia un proceso de PING a una dirección IP fija
+MPINGSTAT	Reporta el estado de un intento de PING

Tabla 5: Lista de Comandos AT utilizados.

7.6. Tarea de Interfaz HMI Teclado/Pantalla y RTCC

Esta tarea controla los dispositivos de teclado, pantalla y reloj de tiempo real. Esta tarea se ejecuta periódicamente refrescando la pantalla a partir de la información contenida en un buffer de datos controlado por su DDT y base de datos de pantallas correspondiente cada 200ms.

El teclado es escaneado cada TickTime del sistema (4ms). Cuando se ha presionado una tecla, se procede a ejecutar un algoritmo que espera un número determinado de muestras consecutivas para validar una tecla. Una vez detectada una tecla, ésta se almacena en una cola de datos tipo FIFO para que durante el próximo refrescamiento de pantalla la tarea tome la decisión adecuada según lo que indique la base de datos de pantallas. Esta base de datos contiene las rutinas que toman las decisiones pertinentes según la lógica del menú que sea programada. Estas rutinas permiten la edición de variables del sistema y el despliegue de pantallas mediante el

teclado. La Figura 37 muestra un diagrama de bloques de los componentes de la interfaz HMI.

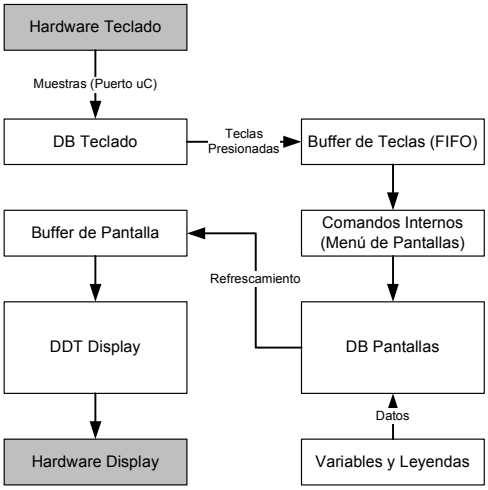


Figura 37: Componentes de la Interfaz HMI.

7.7. Tarea de Puerto Auxiliar

La tarea de Puerto Auxiliar se ha creado con la finalidad de poder configurar o diagnosticar los parámetros internos del sistema a través del puerto serial UART 2 con el uso de la DDT correspondiente. Se planea a futuro utilizar este canal como una "Puerta Trasera" para espiar el funcionamiento del dispositivo y así depurar posibles fallas en su funcionamiento.

8. Menú de Pantallas

Para el manejo de la pantalla LCD se dispone de una base de datos llamada **ScreenDB**. Esta base de datos almacena el estado de la pantalla, el cual se compone de los siguientes elementos:

- Vector de pantallas **ScreenVector** (**Vector**): Contiene la información que se debe presentar en la misma.

- Buffer o imagen de pantalla **ScreenFrame** (**Screen**): Aquí se escriben los datos a mostrar al usuario.

- Identificador de pantalla actual (**ActualScreenID**): Elemento del vector de pantallas a ser mostrado.

- Estado de líneas de la pantalla (**LineStatus: 2 variables**): Se usa para los casos en que se muestren mensajes en marquesina y mensajes titilantes.

- Posición del cursor de pantalla (**CursorPos**): Variable que contiene la posición donde debe aparecer el cursor cuando éste se desee representar.

- **AuxCounter** y **AuxPointer**: Variables temporales que permiten el almacenamiento de datos asociados a las pantallas.

La Figura 38 muestra un bosquejo de la base de datos de pantalla.

ScreenDB (Base de Datos de Pantalla)

u_byte ***Vector** (&ScreenVector)	Dirección de Vector de Pantallas
PtrFrame **Screen** (&ScreenFrame)	Dirección de Frame de Pantalla
u_byte **ActualScreenID**	Identificador de Pantalla Actual
word **LineStatus [2]**	Variables de Estado para líneas de pantalla
u_byte **CursorPos**	Posición de Cursor
u_byte **AuxCounter**	Variable auxiliar para rutinas de pantalla
void* **AuxPointer**	Puntero auxiliar para rutinas de pantalla

Figura 38: Base de Datos de Pantallas.

El Vector de Pantallas **ScreenVector** (campo **Vector**) almacena la programación de pantallas en memoria ROM que serán utilizadas en la aplicación

programada. El frame o buffer de pantalla **ScreenFrame** (campo **Screen**) se encarga de almacenar en sus primeros 32 caracteres una imagen de la información que vaya a ser presentada en pantalla. Por medio de la función **RefreshScreen** se actualizan los datos de este buffer en la pantalla y la posición del cursor (0x00 a 0x0F en la línea superior y 0x40 a 0x4F en la segunda línea), mientras que la función **ProcessScreen** se encarga de cargar los datos a presentar en el buffer de pantalla de acuerdo al número de pantalla seleccionado dado por el campo **ActualScreenID** de la base de datos. Ambas funciones son ejecutadas por la tarea de manejo de pantalla a una rata de tiempo fija preestablecida dentro de la misma. En caso de que la función **ProcessScreen** no sea ejecutada antes de un refrescamiento de pantalla, las variables en RAM no serán actualizadas en la pantalla.

Las variables de estado de línea (**LineStatus**) son manipuladas por estas funciones cuando se programan mensajes de marquesina o mensajes titilantes, siendo éste un medio idóneo para establecer o verificar la posición de la misma o si ésta ha terminado de desplazarse cuando dicha variable de estado alcance el valor -16, es decir, cuando el mensaje reinicie su recorrido de derecha a izquierda.

La Figura 39 muestra a continuación de forma gráfica el comportamiento de estas estructuras y funciones.

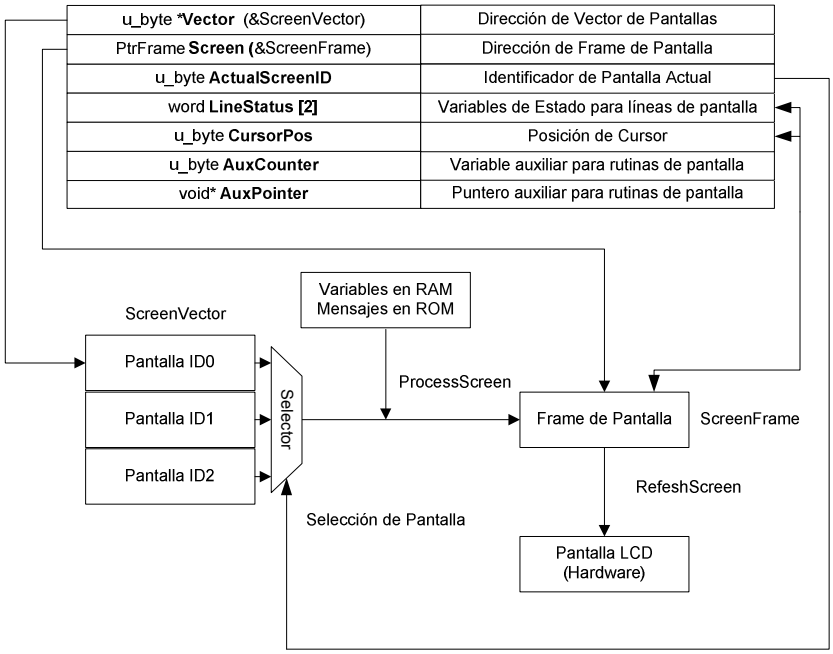


Figura 39: Conexiones entre los elementos de la Base de Datos de Pantalla.

8.1. Vector de Pantallas

El vector de pantallas de esta base de datos se fundamenta en el uso de un conjunto vectorizado de estructuras prefijadas (templates) con un conjunto de características que determinen su comportamiento. La estructura de pantallas **ScreenVector** almacena el conjunto de pantallas que serán necesarias durante la ejecución de la aplicación. Cada elemento de este vector contiene la información de datos y código asociados a cada una de las pantallas a ser presentadas tal y como se muestra en la Figura 40. El orden de las pantallas no debe corresponder necesariamente con el orden de los identificadores de las mismas, ya que éstos últimos son los que determinan su jerarquía.

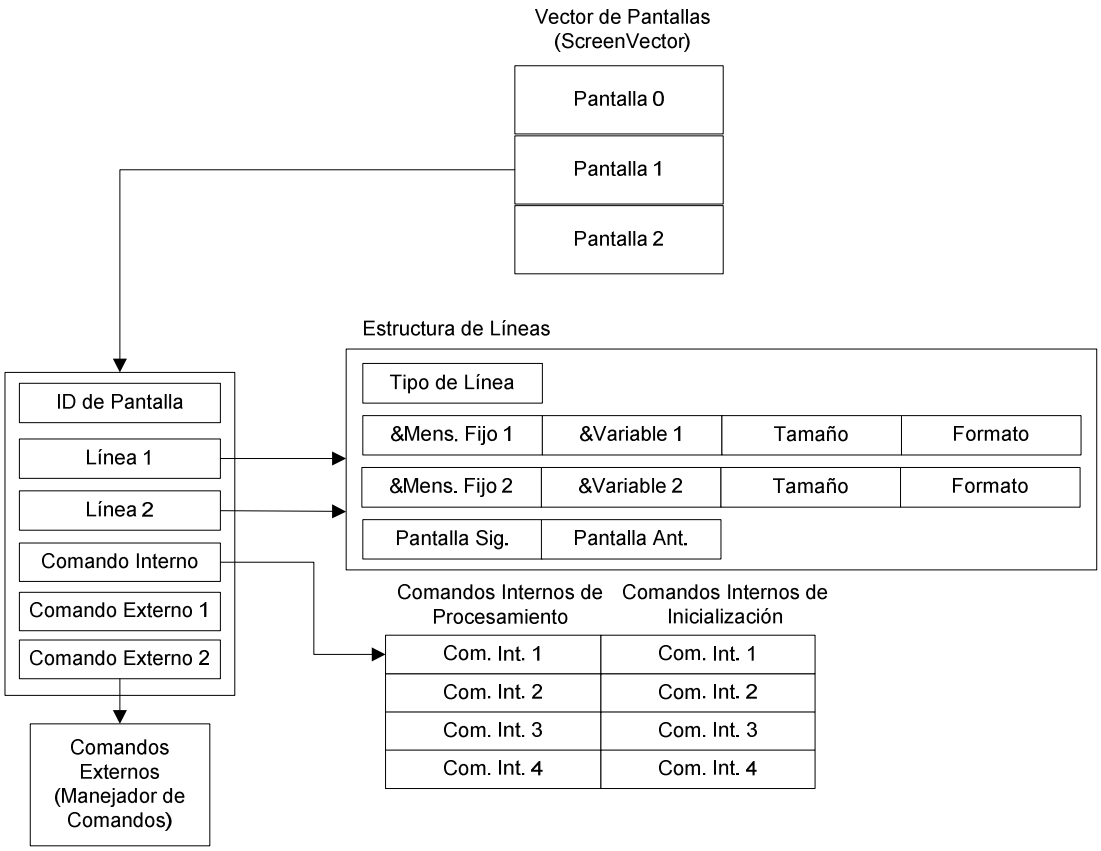


Figura 40: Estructura del vector de pantallas y vectores de rutinas asociadas.

El vector de pantallas se compone de una lista de objetos genérica donde no se programan pantallas con funcionalidades o comportamientos particulares. En vez de eso, se contemplan una serie de opciones dentro de un estándar de plantillas definidas o tipos posibles que pueden ser referenciados dentro del vector. Luego el programador solo llena las plantillas y define cuantas son y cómo se relacionan, sin la necesidad de escribir código de maneje su comportamiento.

La información que contiene este vector consta de direcciones de variables y mensajes fijos, donde los primeros se leen en memoria RAM y pueden presentarse en diversos formatos (ASCII, decimal, binario, hexadecimal entre otros), mientras que

los mensajes fijos se alojan en memoria ROM y siempre son representados en formato ASCIIZ (ASCII terminado en carácter nulo). Cada pantalla consta de 2 líneas y cada línea es capaz de representar hasta 2 mensajes fijos y 2 variables de manera intercalada.

8.2. Rutinas de Manejo de Pantallas

A cada elemento de este vector se le puede asociar un código que permita la operación de un menú de pantallas. El código asociado a cada pantalla se compone de un par de rutinas para inicialización y de ejecución de comandos respectivamente, donde estos comandos pueden provenir de acciones de un puerto de comunicaciones, una acción de teclado o por eventos de tiempo entre otros y es ejecutada por medio de la función **ExecIntCommand()**, la cual es ejecutada cada vez que se realice un refrescamiento de pantalla. La rutina asociada a la inicialización permite preparar las variables del sistema para ejecutar los comandos posteriormente y puede ser llamada también por medio de la función **ExecStartIntCommand()**. Dentro de estas rutinas se puede hacer uso de las variables **AuxPointer** y **AuxCounter** de la base de datos de pantalla como datos temporales.

Para efectos del manejo y despliegue de datos en un menú de pantallas, se han programado una serie de funciones dedicadas para este fin. Dentro de ellas, la función **SetScreen(Identificador de Pantalla)** permite seleccionar la pantalla que se desee mostrar dentro conjunto dado por el Vector de Pantallas pasando como parámetro su identificador correspondiente. Esta función actualiza la base de datos inicializando las variables de identificador y estado de las líneas de pantalla y ejecuta la función de inicialización correspondiente.

De la misma manera, las funciones **SetNextScreen()** y **SetPrevScreen()** se encargan de seleccionar las pantallas siguiente y anterior, cuyos identificadores se

hallan en la estructura de datos asociada a la pantalla actual dentro vector de pantallas. La función **GetActualScreenAddress** permite ubicar dentro del vector de pantallas la estructura de datos de la pantalla actual, retornando el índice donde se encuentra, el cual no se corresponde necesariamente con el identificador de pantalla.

8.3. Tipos de Formato para Líneas de Pantalla

Existen 5 tipos básicos de línea, las cuales pueden ser programadas independientemente del orden o selección de éstos al programar cualquier pantalla:

- Mensaje Vacío (**LineSpaces**): Este mensaje muestra solamente espacios vacíos en la línea programada. Este tipo de programación es ideal para ser usado cuando se deseen mostrar mensajes sencillos de una sola línea.

- Mensaje Fijo (**FixedTitle**): Este tipo de mensaje debe ser almacenado en memoria ROM en formato ASCIIZ (cadena terminada en caracter nulo) y de no más de 16 caracteres sin incluir el caracter nulo.

- Mensaje Parpadeante (**Blinking**): Este tipo de mensaje tiene formato similar al de Mensaje Fijo (16 caracteres en ROM en formato ASCIIZ) e incluye la cualidad de presentarse de forma parpadeante en pantalla.

- Mensaje Marquesina (**Marquesina**): Este tipo de mensaje debe ser guardado en formato ASCIIZ en ROM y puede tener un tamaño mayor al ancho de pantalla de 16 caracteres. El mensaje aparecerá desplazándose de derecha a izquierda en cualquiera de las líneas. En caso de que el mensaje termine de desplazarse, éste volverá a aparecer de la misma manera antes mencionada indefinidamente. Para verificar si una marquesina se ha desplazado completamente en la pantalla se puede verificar la variable **LineStatus** de la base de datos de pantalla asociado a la línea

correspondiente cuando alcance el valor dieciséis negativo (-16), momento en que la marquesina se recarga.

- Conjunto de Textos y variables (**TextVar**): Este tipo de mensaje es el más versátil y puede ser usado para mostrar cualquier tipo de información guardada en memoria RAM para reporte visual o para motivos de depuración de aplicación. Este mensaje consta de 2 campos almacenados en ROM con formato ASCIIZ a manera de títulos y 2 campos variables intercalados con los fijos, tal como se muestra en la figura 1. Los campos almacenados en ROM pueden proceder de cualquier campo de una estructura en ROM o de leyendas independientes también en ROM, mientras que los campos variables pueden ser programados en los campos Formato para ser caracterizados bajo las siguientes representaciones:

- Formato decimal de datos sin signo (**LCDUDecType**).
- Formato decimal de datos con signo (**LCDSDecType**).
- Formato binario de datos (**LCDBinType**).
- Formato Hexadecimal de datos (**LCDHexType**).
- Formato Ascii de datos (**LCDAsciiType**).
- Formato de Hora HH:MM (am/pm) (**LCDHora**).
- Formato de Hora Militar HH:MM (**LCDHoraMilitar**).
- Formato de Hora HH:MM:SS (am/pm) (**LCDHoraMinSeg**).
- Formato de Hora Militar HH:MM:SS (**LCDHoraMilitarMinSeg**).
- Formato de Fecha DD/MM/AA (**LCDFecha**).
- Formato de Día de Semana (**LCDDiaSemana**).

8.4. Propiedades de Formato de Líneas de Pantalla

Los formatos arriba mostrados presentan las siguientes propiedades:

- Los mensajes fijos pueden usarse para justificar los espacios entre campos variables por medio de la inserción de espacios.

- Para programar los mensajes Fijos (**FixedTitle**), marquesinas (**Marquesina**) y titilantes (**Blinking**) sólo se debe colocar en el campo de Mensaje Fijo 1 la dirección de la cadena que debe ser declarada en formato ASCIIZ en ROM. El compilador se encarga de insertar el caracter nulo al final de la declaración de la cadena.

- Los datos a ser mostrados en los formatos decimal, binario y hexadecimal dependen del tamaño de los mismos, ya sean de 8 bits (**LCDByteType**), 16 bits (**LCDWordType**), o 32 bits (**LCDLongType**). Esta propiedad deberá ser colocada en el campo de Tamaño. El número de caracteres que ocupa el campo de datos de acuerdo a la programación de cada formato se muestra en la Tabla 6. Para formatos numéricos decimales (**LCDUDecType y LCDSDecType**) se debe agregar el número de dígitos a presentar en pantalla sin contar el signo en el campo de tamaño, sumándole éste número en dicho campo al identificador (ejemplo: LCDWordType+4).

	LCDByteType	**LCDWordType**	**LCDLongType**
LCDUDecType	3 (XXX)	5 (XXXXX)	10 (XXXXXXXXXX)
LCDSDecType	4 (±XXX)	6 (±XXXXX)	11 (±XXXXXXXXXX)
LCDBinType	8	16	(no se usa)
LCDHexType	2	4	8

Tabla 6: Tamaño máximo ocupado por los campos de datos numéricos en pantalla.

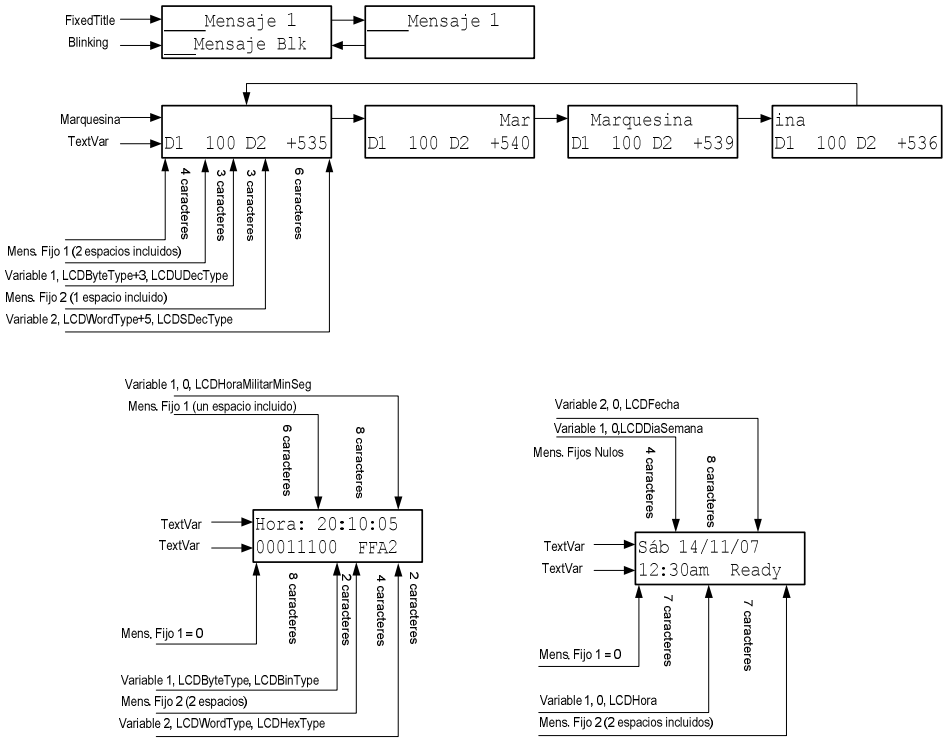
- Los datos representados en formato **AsciiType** van siempre acompañados por el tamaño de la cadena en bytes ubicado en el campo Tamaño. Este formato de datos es conveniente para insertar por ejemplo códigos o passwords través de teclado.

- Para obviar campos de líneas tipo **TextVar** se coloca la dirección de variable o de mensaje como nula. De esta manera no se colocará ningún carácter de estos campos en el buffer.

- Los datos de formatos de fecha, hora y día de semana deberán ser almacenados en estructuras de tipo **RTCCStruct** en formato BCD. Su representación siempre tendrá tamaño fijo, por lo tanto el campo de tamaño no es tomado en cuenta.
  - LCDHora: HH:MM (am/pm) (7 caracteres).
  - LCDHoraMilitar: HH:MM (5 caracteres).
  - LCDHoraMinSeg: HH:MM:SS (am/pm) (10 caracteres).
  - LCDHoraMilitarMinSeg: HH:MM:SS (8 caracteres).
  - LCDFecha: DD/MM/AA (8 caracteres).

- Todo dato de formato de día de semana viene representado por las cadenas LeyendaDom a LeyendaSab y su tamaño dependerá del tamaño de estas leyendas, las cuales pueden ser modificadas a gusto del programador. Se recomienda conservar el orden de estas leyendas al efectuarse cualquier modificación.

A continuación en la Figura 41 se muestran algunos ejemplos de pantallas programadas con sus respectivas características y propiedades tal y como deben ser programadas en el vector de pantallas.




```
{73,
   {FixedTitle,
   &Mensaje1,0,0,0,
   0,0,0,0,
   FixedTitle,
   &MensajeBlk,0,0,0,
   0,0,0,0
   },74,84,PaseMenuID,0}
```

```
{18,
   {Marquesina,
   &MensHora,&HoraActual,0,LCDHoraMilitarMinSeg,
   0,0,0,0,
   TextVar,
   0,&Var1,LCDByteType+3,LCDUDecType,
   &MenEspacios,&Var2,LCDWordType,LCDHexType,
   },32,18,Exec1ID,ComExternoID2}
```

```
{18,
   {Marquesina,
   &MensajeMarquesina,0,0,0,
   0,0,0,0,
   TextVar,
   &MensajeD1,&Var1,LCDByteType+3,LCDUDecType,
   &MensajeD2,&Var2,LCDByteType+5,LCDSDecType,
   },32,18,Exec1ID,ComExternoID2}
```

```
{40,
   {Marquesina,
   0,&Hora2,0,LCDDiaSemana,
   0,&Hora2,0,LCDFecha,
   TextVar,
   0,&Hora2,LCDByteType+3,LCDHora,
   &MensReady,&Var2,LCDWordType,LCDHexType,
   },41,42,Exec1ID,ComExternoID2}
```

Figura 41: Ejemplos de algunos templates con sus características fundamentales.

CONCLUSIONES

Dentro de los puntos concluyentes a los que se pudo llegar, producto de los resultados obtenidos en el desarrollo de este trabajo se enumeran los siguientes puntos:

- Se obtuvo un prototipo de teléfono público inalámbrico con las funcionalidades mínimas exigidas para su correcto funcionamiento.

- El uso de un Sistema Operativo Multitareas en Tiempo Real (RTOS) y el enfoque de programación orientada a objetos facilita el desarrollo aplicaciones donde concurren diversos procesos simultáneos (diagnóstico de fallas, procesos de comunicación, modos de operación y comportamiento, algoritmos de manejo de datos, etc.).

- Se logró verificar que es factible la realización de procesos de comunicación para envío de reportes y alarmas de teléfonos públicos a un centro de gestión por medio de la red TCP-IP, con las consiguientes mejoras asociadas de costo y tiempo empleado para su realización respecto a otros estándares como por ejemplo el estándar DTMF.

- La filosofía de programación implementada para el equipo telefónico facilita la readaptación de su funcionamiento para distintas empresas o consumidores finales, tanto en lo referente al proceso de llamadas como para la emisión de reportes y alarmas al centro de gestión.

Por otro lado, mediante el uso de la metodología de programación orientada a objetos se obtuvieron los siguientes resultados:

- Fácil expansión de nuevos elementos del sistema, donde sólo la estructura de datos (Tabla Descriptora de Dispositivo, Tabla Descriptora de Protocolo, Punto de Base de Datos y Manejador asociado (Handler) son requeridos.

- Fácil procesamiento de los objetos. El conocimiento de cómo procesar un dispositivo determinado está contenido en su manejador (handler).

- Fácil depuración del software y pruebas debido al encapsulamiento de los manejadores de dispositivos y organización de la información.

- Redirecionamiento lógico de los dispositivos al cambiar los punteros de las tablas descriptoras, es decir, el código es reentrante.

- El código específico de la aplicación es independiente de las interfaces que interactúan con los otros elementos del sistema.

- Provee herramientas de interfaz genéricas para sistemas de telecomunicaciones o industriales pequeños o medianos.

- Reutilización de código y fácil control de la documentación.

- Reducción de código de Inicialización (alrededor del 40%).

- Reducción de código de procesamiento debido al encapsulamiento de objetos y estandarización de tareas (aproximadamente 50%)

- Permite la organización del trabajo con múltiples programadores en paralelo ya que las estructuras de datos son estandarizadas.

- Reducción del esfuerzo en horas/hombre en el desarrollo de aplicaciones en general.

- Reducción perceptible en el tiempo dedicado a la implementación de menú de pantallas respecto a metodologías convencionales.

RECOMENDACIONES

Con la finalidad de mejorar las prestaciones del producto final desarrollado en este trabajo de grado y basado en los resultados obtenidos, se proponen las siguientes recomendaciones:

- Readaptar a futuro el funcionamiento del equipo telefónico para diversos tipos de plataforma telefónica existentes en el mercado, como por ejemplo la plataforma Calling Card. Actualmente no existen en Venezuela plataformas de este tipo implementadas para la red GSM.

- Ampliar la gama de fallas que el equipo telefónico pueda diagnosticar, dependiendo de los requerimientos que impongan las empresas que los deseen adquirir.

- Mejorar la lógica de detección, procesamiento y envío de alarmas y reportes al centro de gestión para diversos escenarios, tales como una falla de energía eléctrica o falla de red inalámbrica.

- Mejorar la robustez del equipo telefónico mediante pruebas de campo, con la finalidad de mejorar su desempeño durante el proceso de llamadas así como en la emisión de reportes y alarmas al centro de gestión.

- Agregar una funcionalidad de recarga de software (bootloader) y de grabación de tarjetas SIM en memoria EEPROM (funcionalidades obligatorias para teléfonos públicos en la actualidad).

BIBLIOGRAFÍAS

Tesis.

González Gómez, Aroom Rafael. Diseño e implantación de funciones transportables y multiplataforma para protocolos de comunicaciones industriales: Metodología / Eliécer Henríquez; Mario Giallorenzo (Tesis). --Barquisimeto: Universidad Nacional Experimental Politécnica Antonio José de Sucre, 1998.

Loseto Cantone, Michele. Diseño de software para instrumentación inteligente en un ambiente multitarea-tiempo real: Metodología / Omar Escalona; Mario Giallorenzo (Tesis). --Caracas: Universidad Simón Bolívar, 1996.

Normas.

ISO/IEC (7810 : 1995). Identification Cards – Physical Characteristics. – New York: International Organization for Standarization.

ISO (7811-3 : 2004). Identification Cards – Location of Embossed Characters. – New York: International Organization for Standarization.

ISO (7498-1 : 1994). Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model . – New York: International Organization for Standarization.

OSEK/VDX standard for the automotive industry (www.osek-vdx.org).

Manuales.

Motorola Developer's guide: G24 AT Commands Reference Manual. Manual Nº 6889192V28-F. 31 de Mayo de 2007.

G24 and G24-L Changes Document. Rev. 0.2. Mayo de 2007.

Motorola Developer's guide: G24 Module Hardware Description. Manual Nº 6802984C05-A. 25 de septiembre de 2007.

Libros.

Angulo U., José María y otros. dsPIC: Diseño práctico de aplicaciones. Madrid: Editorial McGraw Hill, 2006.

Overland, Brian. C in plain english. New York: MIS Press, 1995.

Jacobson, Ivar y otros. Object-Oriented Software Engineering. Harlow, England: Addison Wesley, 1998.