INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA

Y ELÉCTRICA

“DISEÑO DE UN CIRCUITO ELECTRÓNICO PARA UNA COMUNICACIÓN DE UN DISPOSITIVO USB CON UNA BASE DE DATOS EN FORMATO VBA PARA EL REGISTRO DE LLAMADAS TELEFÓNICAS EN UN CALL CENTER”

MEMORIA DE EXPERIENCIA PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMUNICACIONES

Y ELECTRÓNICA

PRESENTA

JOSÉ DE JESÚS MOLINA VALENCIA

ASESOR

M. EN C. ERIC GÓMEZ GÓMEZ

MÉXICO, D.F. JUNIO DE 2014

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELECTRICA

UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

T E M A D E T E S I S

QUE PARA OBTENER EL TITULO DE **INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**
POR LA OPCIÓN DE TITULACIÓN **MEMORIA DE EXPERIENCIA PROFESIONAL**
DEBERA(N) DESARROLLAR **C. JOSÉ DE JESÚS MOLINA VALENCIA**

"DISEÑO DE UN CIRCUITO ELECTRÓNICO PARA UNA COMUNICACIÓN DE UN DISPOSITIVO USB CON UNA BASE DE DATOS EN FORMATO VBA PARA EL REGISTRO DE LLAMADAS TELEFÓNICAS EN UN CALL CENTER"

DESCRIBIR EL DESARROLLO E IMPLEMENTACIÓN DE UN CIRCUITO ELECTRÓNICO CON EL PROPÓSITO DE OPTIMIZAR EL REGISTRO DE ACTIVIDADES DE UN CALL CENTER.

- SEMBLANZA LABORAL
- DESCRIPCIÓN DEL PROYECTO
- GENERALIDADES DEL MICROCONTROLADOR PIC18F4550
- DESARROLLO E IMPLEMENTACIÓN DEL PROYECTO
- CONCLUSIONES

MÉXICO D.F. A 03 DE JUNIO DE 2014

A S E S O R E S

ING. ERIC GÓMEZ GÓMEZ

ING. PATRICIA LORENA RAMÍREZ RANGEL
JEFE DEL DEPARTAMENTO DE
INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

DEDICATORIA

A mis Padres, hermanos, a mis hijos Juan Luis y Ana Luisa. A mi esposa Martha por estar siempre conmigo y otorgarme incondicionalmente su apoyo y compañía.

A Ciasa Comercial por ser la empresa que me dio la oportunidad y también grandes enseñanzas, especialmente al Ing. Alfredo Mendoza Ares a quién considero mi mentor en el ámbito laboral.

A Bestbuy Enterprises por haberme permitido pertenecer a ella y hacerme sentir orgulloso de formar parte de su equipo de trabajo durante la mayor parte de mi estancia, pero especialmente a Mónica Silva quien creyó en mi y me dio la oportunidad de mostrar mi aprendizaje permitiéndome desarrollar el proyecto para su área y que se describe en este texto.

Al Instituto Politécnico Nacional que me dio grandes enseñanzas y me recibió de nueva cuenta de manera generosa para permitirme cerrar el ciclo que tenía pendiente y mi más inmensa gratitud a mis profesores y asesores por su tiempo y dedicación.

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE TEMAS

OBJETIVO

Describir el desarrollo e implementación de un circuito electrónico con el propósito de optimizar el registro de actividades de un Call Center.

PREFACIO

SEMBLANZA LABORAL

A partir del egreso de la Escuela Superior de Ingeniería Mecánica y Eléctrica, en la semblanza solo menciona a dos empresas, entre ambas suman cerca de 17 años de vida laboral: Ciasa comercial S.A. de C.V. (de 1996 a 2008) y Bestbuy Enterprises S de RL de CV (de 2008 a 2014).

Ciasa Comercial S.A. de C.V. es una compañía mexicana fundada en 1968 y cuya oficina principal se ubica en la ciudad de Guadalajara Jalisco, se dedica principalmente a la venta y servicio de productos electromecánicos especializados en el proceso de efectivo tales como contadoras de billete y moneda, cajeros automáticos conocidos como "ATM´s" y productos de dispensación de efectivo entre otros productos. Entre sus clientes se encuentran bancos, empresas de traslado de valores y del sector privado. Su eslogan: "La mejor calidad y el mejor servicio".

Bestbuy Enterprises S de RL de CV es una compañía de origen estadounidense cuyas oficinas principales se ubican en Richfield, Minnesota, Estados Unidos. Con más de 1000 tiendas en dicho país, inició operaciones en México con la apertura de la primera tienda en 2008. Es una cadena de tiendas de venta al por menor de productos electrónicos y tecnología así como de un gama de servicios incluidos con los mismos productos. Empresa reconocida por su cultura y valores.

Al igual que la mayoría de los estudiantes recién egresados, el comienzo fue intentando conseguir empleo en empresas de tecnología ya sea para diseño, laboratorio o servicio. Siempre relacionados con la carrera, con la finalidad de que se me permitiera desarrollar, aprender y crecer. La realidad laboral de ese entonces y aún más hoy en día es que la mayoría de las empresas solicitan Ingenieros titulados, principalmente con alguna experiencia o conocimientos comprobables en alguna rama comercial, tecnología o marca de productos muy especializados, o por lo menos con certificación en alguna de esas tecnologías.

En un principio, en una de esos intentos de conseguir empleo, después de una entrevista donde además me practicaron una prueba sobre MSDOS, fui llamado de nuevo junto con otros candidatos seleccionados y entonces, la directora de dicha empresa mencionó que no nos llamaban para trabajar en soporte técnico, sino para ventas de sus productos aprovechando nuestros conocimientos tecnológicos. Ofrecían impartirnos cursos de mercadotecnia para tal fin. En ese entonces, decidí no asistir porque no concebía la idea de dedicarme a vender productos. Era parte de mis aspiraciones el trabajar con tecnología pero no para vender, no era lo que tenía en mente realmente al iniciar mis estudios de Ingeniería.

Reanudé mis intentos de conseguir empleo y después de una relativa larga búsqueda, por fin logré colocarme en una empresa donde una de sus ramas era el servicio técnico. Esta empresa que me otorgó la oportunidad fue Ciasa Comercial. Ingresé en principio en el puesto de Ingeniero de servicio y la empresa me capacitó para la reparación de sus equipos. En principio me pareció el trabajo un tanto diferente, ya que se trataba de reparar equipos que en realidad eran máquinas, con bandas, poleas, engranes y que se ensuciaban mucho, pero que también estaban dotadas con

sensores y tarjetas electrónicas, aunque estas eran las que menos se dañaban. No tenía nada que ver con comunicaciones, al menos en ese entonces. Me costó un poco de trabajo aceptar aquellos cambios pero decidí perseverar y sostenerme en esta empresa ya que la búsqueda había sido larga. Aprendí a reparar partes mecánicas y había momentos donde podía aplicar mis conocimientos de electrónica para reparar circuitos electrónicos incluso a nivel componente. Con el tiempo, estas actividades llegaron a agradarme, pues tenía ya nuevas habilidades. A los dos años de mi ingreso a esta empresa, me ofrecieron el puesto de supervisor de servicio y transcurridos otros dos años después, me ofrecieron ocupar la posición de Jefe de Servicio la cuál aceptaría como una gran oportunidad de crecimiento. Me encargaría en mi nueva posición de la administración y de la gestión del personal técnico del centro de servicio correspondiente al Distrito Federal y área metropolitana. Tenía nuevas tareas, administrar la logística de servicio, gestionar al personal técnico y administrativo, retener los contratos de mantenimiento existentes y negociar nuevos, asistir y apoyar a ventas en la demostración de productos, administrar la facturación y cobranza de los servicios, administrar el inventario de partes. Estando ya en esta posición, recordé aquél trabajo de ventas que rechacé por pretender no dedicarme a labores administrativas y comerciales, dándome cuenta que como parte del crecimiento profesional en una empresa, tarde o temprano nos vemos involucrados con dichas actividades y que en la vida laboral no siempre iba a realizar únicamente actividades tecnológicas pues había que abarcar otras áreas si quería continuar con mi propio crecimiento. Había que abrir la mente y aceptar el hecho de que era necesario aprender muchas cosas más para aprovechar las oportunidades que se podrían presentar.

Durante mi labor como jefe de servicio las tareas administrativas en principio no fueron fáciles. El aprendizaje fue un tanto duro y difícil pues no tenía mucha capacitación especializada al respecto. Había que superarlo y mejorar. Había que tomar el control y verdaderamente administrar. Se aprendió a gestionar sobre la marcha y a lo largo de los años, pero también hubo mucho apoyo y consejos del Gerente Nacional de Servicio que me ayudaron a sacar adelante mi trabajo.

Para mi beneplácito, también la empresa con el tiempo introdujo nuevas tecnologías y había que capacitarse sobre ellas, ya que los clientes exigían mejoras para sus procesos. Se introdujeron equipos con puertos de comunicaciones RS232 en ese entonces, equipos con más y mejores sensores y con sistemas que permitían leer la denominación de cada billete para mostrarla o para hacer operaciones de conteo. Se requería ya conectar las máquinas a equipos de cómputo y crear algunas aplicaciones para la interacción de los mismos mostrando su capacidad de conexión. Llegaron productos todavía más complejos ya con puertos USB donde había que actualizar constantemente el programa de sus microcontroladores debido a los frecuentes cambios de la moneda y de los billetes que hacía Banco de México. Se introdujeron los cajeros automáticos para su venta y con ello la posibilidad de configurarlos dentro de su red de datos utilizando instrucciones de protocolos de comunicación como el TCP/IP. Se hablaba ya en la empresa de redes de datos y sistemas de comunicación tanto a nivel de ventas como en servicio. Finalmente, al final de mi gestión, se introdujeron cajas blindadas para comercios que incluían tecnología de telefonía celular y que enviaban mensajes SMS (Short Message Service) a un servidor de datos

para indicar su estado operativo y funcional haciendo más eficientes los procesos de traslado de valores reduciendo costos para los clientes.

En cuanto a la administración y operación del centro de servicio la metodología era un tanto empírica y laboriosa en principio. Cabe mencionar que hoy todavía muchos centros de servicio trabajan así incluso de marcas de productos reconocidas con las que he tenido oportunidad de tener contacto. En aquél entonces el personal del área obtenía la lista de servicios a realizar de una hoja de cálculo conocida como “Lotus”. Elaboraba cada reporte de servicio de mantenimiento preventivo o correctivo en una máquina de escribir, uno por uno. Las partes de repuesto de los equipos que se proporcionaban a los técnicos iban acompañados de un vale que se llenaba a mano y que tenían que firmar. Se controlaba el inventario llenando formatos y haciendo conteos también a mano. Los contratos de servicio se revisaban a través de archivos impresos y para consultarlos había que sacarlos del mueble archivero ya que no había todavía una base de datos. Diariamente, al final de la jornada, cada ingeniero de servicio entregaba sus reportes de trabajo y se acumulaban para que a final de mes se contabilizara su productividad para el pago de sus comisiones. Mi primera audacia fue introducir antes que nadie en esa oficina el naciente Microsoft Excel a mi área como una alternativa para mejorar la gestión como una promesa interesante de interfaz de Windows más eficiente para hacer el trabajo. Trabajar solo con Excel no era suficiente, ya que era una inquietud personal mejorar mi gestión, anhelaba y sentía que debía tener un control mucho más eficiente de los inventarios de partes para evitar faltantes, situación que solía ocurrir debido al proceso de control existente. También era necesario poseer un mejor control de los contratos y a través de estos, de los inventarios de equipos que poseían los clientes. Era también un reto conseguir elaborar e imprimir de forma automática los reportes de servicio para ser entregados al personal técnico con la finalidad de reducir tiempos muertos. En general, tenía como objetivo lograr una administración más efectiva concentrando y registrando la operación en una sola aplicación.

No siempre es fácil vender una idea nueva sin realizar nunca antes en un sitio si no se logra mostrar y convencer a plenitud de los alcances y el impacto que estas tienen en las áreas de operación. Comprensiblemente puede ser tomada como riesgosa. También esta el asunto del presupuesto, pues no es fácil obtenerlo sin lograr previamente el convencimiento de la idea y de su beneficio o simplemente porque no se cuenta con él en el momento. El implantar un sistema que controle todo pudiera representar grandes costos considerando también que el desarrollo de programas dedicados y hechos por empresas externas resultan ser un gasto considerable comparado con la productividad de una área de soporte técnico. No era sencillo, pero no por eso desistiría de la idea, es a causa de eso que nació la iniciativa de solucionarlo personalmente, pues era mi propia operación la que necesitaba fuera más eficiente. Cabe aclarar que en mis tiempos de escuela aprendí a programar en Fortran, pero por las cercanías al año 2000 ya no se utilizaba, por lo que comencé explorando y aprendiendo a programar entornos de programación orientado a objetos, empezando por Visual Basic y ya muy avanzado su aprendizaje me pareció que algo le faltaba, pues no encontraba la forma de elaborar registros o que me permitiera generar una base de datos. Aborté el uso de Visual Basic y comencé a explorar Visual C++, un lenguaje más completo

pero muy complejo. Pasaba el tiempo y no encontraba lo que requería. Supe entonces que existía ya en el mercado una nueva aplicación de Microsoft llamada Access y comencé a estudiarla. Aprendí que en ella se podían armar bases de datos relacionales (diversas tablas de datos ligadas entre sí) de forma sencilla y que me permitía llevar el registro de algunas de las tareas que necesitaba como el control de contratos y de servicios, pero algo más faltaba, ya que necesitaba también controlar y gestionar de manera automática el costoso inventario de partes que diariamente se suministraba al personal técnico. Entonces dentro de mi exploración de Access, descubrí que a parte de diseñar formularios con algunos eventos ya incorporados por Microsoft para la visualización y control de registros, también se podía programar eventos a través de una interfaz de Visual Basic incorporada conocida como Visual Basic para Access (VBA).

Me pareció encontrar la solución, pues ya sabia programar en Visual Basic por lo que me tomé la tarea de crear mis primeras aplicaciones en Access y VBA para gestionar las actividades operativas de mi departamento, siendo la más compleja en Ciasa la del control de inventarios, que me permitía de forma controlada y precisa administrar y dar seguimiento al citado inventario de partes. También podía entregar estas mismas partes al personal técnico registrando de forma automática las bajas y las altas de los mismos permitiendo siempre llevar un registro exacto del stock existente y logrando también monitorear el uso de las mismas después de cada reparación de equipos para asegurar su buen uso y generar reportes estadísticos. Este preciso programa y su metodología de control sería el que años más tarde permitiría la creación y nacimiento al proceso de recuperación de activos en Bestbuy.

En Ciasa se operaron estas bases por algunos años. Se me permitió en algún momento presentarlas y demostrarlas por primera vez en una junta anual nacional donde el director de la empresa se mostró muy interesado, entonces me solicitó entregara una copia de ellas al director de administración. No sucedió nada más que yo supiera, pero el tiempo transcurrió y llegó SAP (del alemán: *Systemanalyse und Programmentwicklung* "Sistemas de Análisis y Desarrollo de Programas") a la empresa, una aplicación de negocios mucho más poderosa pero también muy costosa que permitía la gestión completa de las actividades de la empresa. Un sistema armado en bloques que incorporaba actividades de inventarios, traspasos, facturación, cobranza y servicios. No sé bien todavía si mis bases influyeron en algún momento para que la empresa mirara y se decidiera a mejorar su operación adquiriendo SAP, pero si fueron las primeras aplicaciones prácticas dedicadas al área de servicio técnico y demostraron hacer eficiente su operación y control.

Continué usando las bases de forma paralela ya que SAP No tenía todo lo que yo necesitaba. También durante esta gestión continuaba dando apoyo técnico y fue cuando fueron desarrolladas algunas aplicaciones en Access para preparar demostraciones de interconexión a PC´s con algunos equipos que la empresa vendía a través del puerto RS232 y utilizando el protocolo TX-RX para recibir información de las contadoras de moneda y registrando los conteos en una base de datos con la finalidad de convencer a prospectos de clientes de los alcances de los equipos en sus procesos. Para entonces, el cliente debía desarrollar sus propias aplicaciones.

Transcurrido el 2008, fui invitado a participar en el desarrollo y puesta en marcha de algunos procesos para la apertura que se preparaba para ese mismo año de la primera tienda de electrónicos de Bestbuy Enterprises en México. Fui contratado por mi experiencia en administración de servicio técnico y tomé la decisión de dejar el puesto de Jefe de Servicio en Ciasa para incorporarme a Bestbuy como Supervisor de Servicio Postventa. Mi lugar de trabajo inicial fue dentro del corporativo y mis funciones en principio fueron las de apoyar en la selección y desarrollo del proveedor que se encargaría de administrar y realizar los servicios de garantía extendida (en términos de Bestbuy se llaman Planes de servicio de producto y planes de reemplazo de producto, PSP y PRP respectivamente), seleccionar un centro de servicio y desarrollar con ellos el proceso de servicio a productos de marca propia y finalmente desarrollar el proceso de recuperación de activos del cual me haría cargo una vez abriera al público la primera tienda.

Durante mi reciente incorporación en esta compañía, una de las cosas que me mencionaron fue que era una empresa de procesos ya establecidos y estándares ya diseñados. Efectivamente era así, los procesos operativos ya estaban hechos y documentados pues provenían de los mismos procesos que se ejecutaban en las tiendas Bestbuy de Estados Unidos. Además, se estaba incorporando el sistema SAP (nuevamente me encontré con este sistema) para la operación integral de la compañía. Este era un SAP mucho más robusto y mucho más poderoso que el que yo había conocido. Con toda la información que se me había mencionado creí que ya no habría necesidad de volver a trabajar con Access, pues se me había dicho que *"todo"* estaba ya hecho, que solo había que aplicar y dar seguimiento a los procesos utilizando herramientas proporcionadas como SAP.

Durante esos meses iniciales en la compañía fui enviado algunos días a Richfield, Minnesota, Estados Unidos, para entrevistarme con algunas personas del corporativo y ahondar un poco más en los procesos de servicio de marca propia y recuperación de activos. También estuve en alguna tienda de dicho lugar para observar su proceso de devoluciones y finalmente visité un centro de distribución de la compañía para también observar las labores que ejecutaban tratando de encontrar algún proceso de servicio que me pudiera servir.

Noté que en Estados Unidos utilizaban sistemas diferentes, con aplicaciones dedicadas y maduras en sus procesos. Aplicaciones tales como generar una etiqueta con un código de barras y texto de disposición del producto cada vez que un cliente hacia una devolución en tienda, lo que realmente facilitaba las cosas. Me parecía muy efectivo dicho proceso. Pero cuando regresé a México, me di cuenta que con la migración a SAP como implementación en el país, se estaban sustituyendo los sistemas y aplicaciones que se estaban usando en Estados Unidos y que se omitieron algunos procesos desde un principio que estaban relacionados específicamente con mis áreas de trabajo.

SAP tenía muchas herramientas y realmente recaía toda la operación de la empresa en este sistema. Todos los miembros de la compañía debíamos utilizarlo al igual que sucedió en Ciasa, pero no había procesos en este sistema diseñados para la recuperación de activos, ni desde tiendas ni de ninguna manera. No existía. Había procesos dentro de SAP que permitían manejar y

transferir virtualmente inventarios, visualizar los mismos inventarios de las tiendas, mecanismos de cobro de ciertos inventarios a algún proveedor, darlos de baja pero no había mecanismos de cómo recuperar, visualizar y dar seguimiento a las devoluciones de clientes en tiendas. No bastaba ver los inventarios, no había ninguna señalización de ello en SAP. Nadie lo implementó.

Debido mi experiencia en Ciasa y sobre todo con el manejo de inventarios aprendí que el control debe ser muy cerrado y preferentemente centralizado, manejado por personal dedicado explícitamente a ello para evitar pérdidas y rezago de servicios, en el caso de las empresas de venta al por menor, tampoco ningún producto debía quedarse sin un seguimiento. Por el volumen y la complejidad de su logística de movimiento de mercancías había que tener un muy buen control. Se sugirió centralizar las devoluciones en un solo lugar y desde allí desarrollar un proceso que permitiera controlar el flujo de recuperación de una manera más eficiente. Esto fue apoyado por los directores y se centralizó el proceso en el Centro de Distribución. Pero SAP continuaba sin un proceso dedicado y útil para ello y sería muy costoso crear uno. Lo que más se hizo en SAP fue crear un almacén virtual dedicado para ver el volumen de inventario que se sabría correspondería a las devoluciones de la compañía. Nació el Centro de Recuperación de Activos, pero faltaba darle un flujo y crear un proceso con herramientas que permitiera monitorear y dar seguimiento al estado que guardaba la mercancía. Había que darle necesariamente un enfoque de servicio pues era precisamente con centros de servicio de diversas marcas con quienes se iba a trabajar. Resurgió entonces la idea de volver a utilizar Access como una solución para los procesos que era necesario crear.

Se desarrollo inmediata y paralelamente a la apertura de la primera tienda una base de datos con aplicaciones de VBA en la que recaía la operación del centro de recuperación de activos. Esta base a lo largo de 5 años evolucionó. Comenzó montándose una base matriz en una ubicación lógica de uno de los servidores de la empresa y a ella se conectaban otras bases de datos denominadas terminales para visualizar la información y operar en línea, pero debido a algunas deficiencias de Access para trabajo en red, se modificó la base de datos creando una base maestra dentro del servidor y a partir de ella crear réplicas que permitían a los administradores del centro de recuperación operar sin ningún problema incluso sin red, el único requisito era que tenían que sincronizar sus bases antes del fin de cada jornada para que al principio de la siguiente los miembros del equipo volvieran a sincronizar para que contaran todos con la misma información.

También, la recepción de productos provenientes de las tiendas había que registrarlas adecuadamente y darle el flujo correcto desde el punto de vista servicio. SAP permitía generar reportes de servicio con folios, herramienta diseñada para el seguimiento de las garantías extendidas, pero que nos permitió utilizar también para las devoluciones. Desde tienda se generaban folios de servicio para cada producto devuelto por los clientes, que servía como una especie de acta de nacimiento para cada producto con su correspondiente información como tienda de origen, falla, etc. A la base de datos se le desarrollo un procedimiento en una consulta de Access que permitía generar un código de barras del tipo *"*EAN13*"* para después imprimirlo en una impresora de uso doméstico. También se diseño una aplicación en VBA para leer los mismos códigos a través de un lector de código de barras comercial conectado a la base. Cuando se recibía

la mercancía, se registraba con un lector de código de barras el "UPC" del producto y se escribía su folio de servicio para que al final del ingreso del total de productos recibidos, se imprimieran los códigos de barras y se colocaran en el producto correspondiente. Esto resultó potencialmente útil, ya que se desarrollaron procesos, aplicaciones y candados que requerían el código de barras. Según lo que el personal requiriera hacer o el flujo que le quisieran dar al producto, debían tener que registrar el código de barras del folio de servicio, esto nos permitía conocer el flujo real que tenía la mercancía en la base de datos logrando también asegurar la integridad del inventario ya sea que un proveedor X retirara los productos o los regresasen reparados, siempre se sabría dónde estaba la mercancía. Se colocaron varios estados del flujo de mercancía según el tratamiento que recibieran, por ejemplo; reportado con proveedor, recogido por proveedor, devuelto reparado, enviar a destrucción, devuelto como cambio físico, aplicar cargo a proveedor, concluido, etc. Paralelo al proceso con lo productos se generó una aplicación en la misma base que permitiese registrar cada evento o cada cambio de estado de la mercancía, registrando incluso la fecha y hora del evento así como el nombre del operador registrando la información en registros denominados "log". Esta serie de registros, más el del inventario con sus estados permitieron elaborar todo tipo de informes que se requirieran, del volumen y avance de la recuperación, la rotación del producto, la productividad de los operadores del centro de recuperación, del desempeño del servicio de los proveedores, etc.

Por otro lado, durante la apertura de las primeras tiendas, me di cuenta que para el proceso de gestión de garantías y devoluciones los proveedores de productos estaban muy acostumbrados a imponer sus reglas, tiempos y procesos. Dichos procesos estaban diseñados para tratar directamente con tiendas y es así es como trabajaban habitualmente con las empresas de venta al por menor que existían desde antes de Bestbuy. Debido a las necesidades del proceso de servicio de la empresa y los tiempos de respuesta requeridos, así como de las métricas que la base de datos arrojaba, hubo que trabajar mucho con los proveedores, en principio renuentes a cambiar sus procesos de garantía orientados a trabajar directamente con las tiendas pero finalmente ajustándose a las necesidades de la empresa. También en cuanto a sus tiempos de respuesta y calidad de sus servicios se lograron grandes avances, cambiando los mismos proveedores a sus centros de servicio y optimizando los mismos. Incluso en marcas importantes y muy reconocidas ocurrió esto.

Se puede decir que el proceso establecido para la recuperación de activos pese a sus limitaciones resultó exitoso. Realmente el aporte de este tipo de operación influye en la reducción de pérdidas para la empresa y por consecuencia eleva su utilidad derivada de la venta de sus productos.

El área para la que trabajaba, se llama "servicio postventa", se componía de otras sub áreas: las asignadas a mi cargo, una llamada "recuperación de activos" y otra llamada "planes de servicio", pero había otra sub área con otro supervisor que se encargaba de los servicios de instalación profesional de línea blanca, audio y video y otra también denominada call center que se encargaba de gestionar, programar los servicios de entrega e instalación de productos marcando telefónicamente a los clientes y elaborar informes de programación para que el equipo de instalaciones realizara los servicios. Cada sub área comenzó gestionando y desarrollando sus

procesos operativos de la mejor manera posible encontrándose en la misma situación que las mías. No había SAP con un proceso perfecto o dedicado. De igual manera, no había muchas herramientas eficaces que permitieran registrar y ayudar a controlar sus procesos, así como compartir la información según se requiriera para bien de la propia operación y de los clientes.

El equipo de call center obtenía la información de venta de servicios de instalación de SAP y los vaciaba en una hoja de Excel que ellos llamaban base de datos. El archivo de Excel lo subían a una carpeta compartida en algún servidor y uno por uno vaciaba la información de su jornada de trabajo, uno a la vez, ya que Excel no permite el trabajo en red. Cada que terminaba un operador de registrar su información, guardaban el archivo antes de salirse y permitir que otro pudiera utilizarlo. Además, el archivo después de cierto tiempo era tan grande en tamaño, de varios Megabytes que hacía muy lenta la operación de apertura y cierre del archivo.

Ya habiéndose probado para entonces la efectividad de la base de datos del centro de recuperación de activos, se me invitó a desarrollar otras bases de datos en Access para optimizar las operaciones de los equipos del call center y del equipo de instalaciones.

La del equipo de instalaciones profesionales fue muy sencilla, Access básico, pero la del call center, además de integrar y compartir en red para el mismo equipo la información de ventas de servicios y programación de la entrega e instalación de productos, se desarrollaron a lo largo de dos años y trabajando sobre la marcha de la operación, mejoras y herramientas que permitieran hacer este proceso más eficiente como la del centro de recuperación de activos, es más, fue más compleja y completa. A la base de datos del call center durante sus fases de diseño y actualización se incorporaron herramientas tales como:

- A través del uso de API´s (Application Programming Interface) en conjunto con VBA, rutinas para el monitoreo y notificación de la conectividad de la red. Se utilizaron comandos de protocolo TCP/IP dentro de rutinas de VBA para monitorear constantemente la conectividad con el servidor, lo que permitía a los usuarios saber si estaban en línea o no.
- Se incorporó a través del uso de API´s código que permitiera recolectar información del sistema, refiriéndose específicamente al usuario y serie del equipo de cómputo para después generar un registro de actividades y eventos relacionados con las actividades en la base de datos llamado “log” y permitiría generar reportes para representar el desempeño del proceso y del personal que lo operaba.
- Se generaron rutinas en VBA que permitiera de manera controlada y mucho más rápida enviar reportes de servicio en formato “PDF” al equipo de instalaciones generadas por la misma base de datos y utilizaba también código para enviar dichos reportes al equipo correspondiente.

Finalmente, el call center tenía también la necesidad de contar con métricas tanto individual como colectivas de las cargas de trabajo, del avance y seguimiento de las programaciones comparado con las ventas de los servicios, pero también requería contar con algo práctico que pudiera tomar

registro de las llamadas telefónicas tanto realizadas como recibidas de manera precisa. Todo esto en conjunto permitiría tener una idea clara de las necesidades del área así como de las mejoras y ajustes que tuviera que hacer o por el simple hecho de conocer su propia rentabilidad. No había presupuesto autorizado para adecuar SAP o adquirir tecnología para ello. Debido a las necesidades de esta área, estudiaba yo una solución, pero veía ciertas desventajas técnicas: La red telefónica estaba compuesta por una red "CISCO" de voz y datos administrados por una empresa telefónica particular. No se trataba del teléfono común y corriente que encontramos en nuestros domicilios. La terminal telefónica recibía la información de voz proveniente de un servidor de datos e IPvoice y todo en conjunto por la misma red de tipo Ethernet. Por una terminal adicional tipo Ethernet del mismo aparato telefónico fluía la información solo de datos hacia el ordenador de cada usuario. Aún más, Bestbuy como muchas otras compañías, tiene una política de no intervención de los sistemas y equipos de cómputo ni la introducción de software externos no autorizados. Tampoco se tenía acceso al software del teléfono que nos permitiera extraer directamente la información de las llamadas hechas y recibidas. Entonces decidí realizar algunas pruebas en mi domicilio con mi propio aparato telefónico desconectando el extremo del cable terminado con un RJ9 que une el aparato telefónico con el auricular, pues en esto son similares los teléfonos convencionales con los teléfonos CISCO. Comencé a realizar mediciones de voltajes a través de las 4 los pines del conector RJ9 en varias combinaciones buscando algún comportamiento tanto al colgar como al descolgar el auricular. Se encontró en una de esas combinaciones una pequeña variación de voltaje cada que se colgaba o descolgaba el auricular. También de un par de esos 4 pines se supuso debería haber audio analógico, poder escuchar los tonos cada vez que se hacia una marcación lo cual resultó cierto. Intenté después obtener las mismas lecturas en un auricular de un aparato CISCO y aunque diferían tanto en configuración como en niveles de voltaje, el comportamiento era el mismo al de un aparato convencional. Nació entonces la idea de construir un dispositivo electrónico que se conectara como puente entre el aparato telefónico y el auricular, que detectara esas variaciones de voltaje al colgar y descolgar y enviara algún comando específico a la computadora que le indicara inmediatamente que evento ocurría en el aparato telefónico para ser registrado en la base de datos Access ya existente. La idea iría mas allá. Se podía aprovechar las señal de audio interceptada para detectar los tonos de marcación y también enviar esa información a la base de datos para registro de los números telefónicos de las llamadas realizadas obteniendo más precisión en el proceso. De nueva cuenta se encontraron dificultades para seleccionar la forma en que se enviaría la información del dispositivo a la base de datos. Ya no era muy comercial el puerto RS232 ni el paralelo. Algunos equipos ya solo contaban con puertos USB (Universal serial Bus). Había que por lo tanto implementar comunicación por USB, pero el conectar un dispositivo vía USB implicaba irremediablemente tener que utilizar o instalar un software controlador de dispositivo (driver) lo que podría ser interpretado como software externo por la empresa. Por fortuna existe una serie de controladores de dispositivo por defecto instalados en los equipos de cómputo que proporciona Windows desde la versión 98 y particularmente uno muy frecuentemente utilizado por los teclados alfanuméricos, ratones (mouses), etc. conocida como "Dispositivo de interfaz Humana" o por sus iniciales en inglés HID cuya invocación y comandos servirían para comunicar al dispositivo con la computadora a través del puerto USB utilizando como aplicación Access sin la necesidad de crear un controlador externo que violara la política de

la empresa. En cuanto a la aplicación, el hecho de que Access es un programa que viene incluido en la versión profesional de Office de Microsoft y este se encuentra instalado por defecto en los equipos que la empresa proporciona, tampoco se incurría en una violación a la política. Las dificultades conceptuales del proyecto estaban resueltos, solo había que seleccionar algún chip o diseñar un circuito que interpretara los tonos de audio provenientes del aparato telefónico y convertirlos en datos utilizables, había también que seleccionar un microcontrolador que manejara las comunicaciones USB y programarlo. Resuelto todo esto, se diseñó y construyó lo que se nombró "dispositivo *Nosy".* El nombre en inglés de *"Nosy"* fue dado al dispositivo por el supervisor de call center derivado de su significado en castellano que es "entrometido", ya que se conecta entre los cables de audio del aparato telefónico y del auricular y esta a la espera de algún evento realizado en el teléfono para inmediatamente notificarlo, en este caso a la base de datos para el registro del evento.

Esta base de datos diseñada para el call center que en conjunto con el dispositivo *"Nosy"* fue reconocida por Bestbuy con el *"Brad Anderson Legacy Stock Award"* en el año de 2012 (anexo 4). Debido a su funcionalidad y a que este proyectó incluye algunos aspectos de comunicaciones, programación y diseño electrónico es la que se describe como ejemplo de aplicación de conocimientos tecnológicos de la carrera de Ingeniería en Comunicaciones y Electrónica a lo largo de esta memoria.

Nota: Los capítulos siguientes muestran a detalle y en secuencia el desarrollo del proyecto tal como se realizó y se implementó. Algunos temas del proyecto necesariamente fueron tratados dentro de su mismo marco teórico con la intención de dar claridad a los aspectos relevantes que se debieron considerar en el desarrollo, pues con ello se pretende que la memoria sirva también de guía para quien desee iniciar un proyecto aprovechando la conectividad USB.

Algunos temas irremediablemente hacen uso de terminología, nombres y descripciones en el idioma *"inglés"* indicados entre comillas y cursiva debido a que de la misma forma se encontrará en la consulta por ejemplo de las hojas de especificaciones de datos del microcontrolador, de las definiciones y documentación proporcionada por el USB Comite*,* de la documentación y códigos proporcionados por el fabricante Microchip*,* de los códigos fuente y sintaxis para los lenguajes de programación en C y VBA. Al no existir literatura oficial en el idioma español se respetaron algunos de los tecnicismos, nombres originales y códigos de programación tal como se implementaron en el proyecto original.

CAPITULO 1

DESCRIPCIÓN DEL PROYECTO

1.1 El origen, las señales base del proceso.

Al desconectar el conector macho RJ9 del auricular del aparato telefónico CISCO y realizando mediciones de voltaje en distintas combinaciones se encontró que entre las terminales 1 y 3 se tenía una variación de voltaje de entre 0 volts cuando el auricular se encuentra colgado y 0.75 volts aproximadamente cuando el auricular se encuentra descolgado. La terminal 1 es de polaridad positiva respecto a la terminal 3. Véase la figura 1.1a. Estas mediciones fueron tomadas con carga, es decir; se realizó un puente para permitir la conexión del auricular y poder tomar las lecturas de voltaje con el auricular conectado. También se hace notar que el audio se obtuvo entre las terminales 2 y 3. La conexión del micrófono se obtuvo entre las terminales 1 y 4.

De lo anterior se dedujo que era posible tomar esta variación de voltaje entre las terminales 1 y 3 para detectar el evento de colgado y descolgado , tomando como negativa o tierra la terminal 3.

De la misma manera, se podía obtener el audio de entre las terminales 2 y 3 tomando como común o tierra la terminal 3 y entrada de audio al dispositivo la terminal 2.

Aunque en gran cantidad de aparatos telefónicos comerciales es muy similar la configuración descrita anteriormente, existe la posibilidad de cambiar entre un aparato y otro, así como también los voltajes y polaridades descritos. Para otros diseños es necesario tomar lecturas propias del aparato telefónico con el que se va a trabajar. Este diseño fue hecho para un modelo particular de aparato telefónico CISCO 7942 (Figura 1.1b).

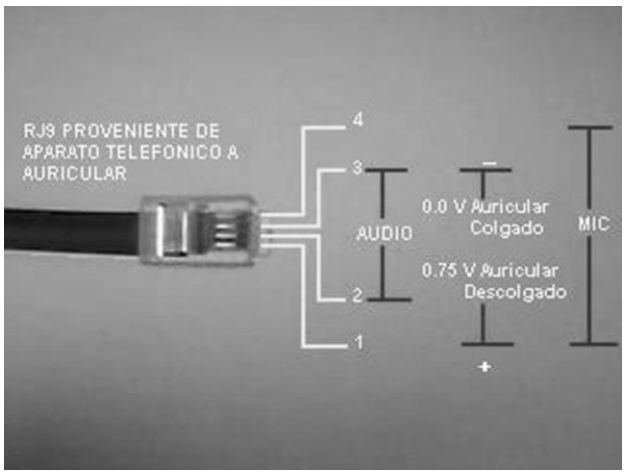


Figura 1.1a Voltajes obtenidos en el conector RJ9

Figura 1.1b Vista teléfono CISCO 7942

1.2 Esquema básico de conexión.

De acuerdo a lo explicado anteriormente, se concibió la idea de conectar un dispositivo electrónico (Nosy) que sirviera de puente entre el cordón o cable que conecta al aparato telefónico y el auricular sin interferir el servicio de voz y datos ya existente. Véase figura 1.2.

Al detectar los eventos de colgado, descolgado por medio de variaciones de voltaje y la marcación de números por medio de la decodificación de tonos de audio, se podría reinterpretar estas señales por el *"Nosy"* para posteriormente notificar estos eventos vía USB a una aplicación diseñada para el registro y detección de dichos eventos.

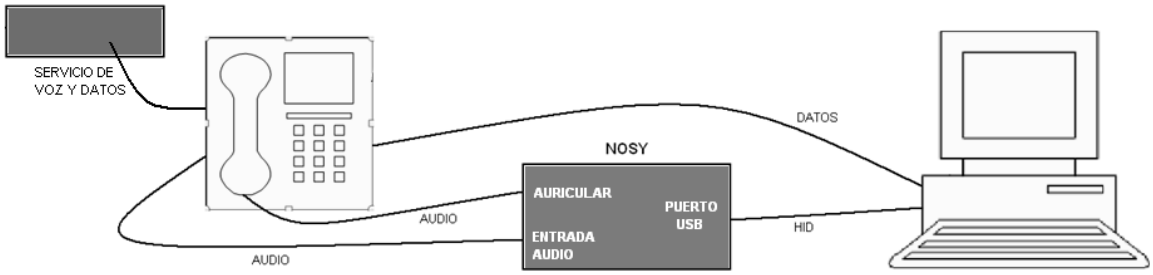


Figura 1.2 Esquema básico de conexión.

1.3 El proyecto completo en bloques.

En la figura 1.3 se muestra el proyecto completo en bloques.

De acuerdo a lo mostrado en este diagrama a bloques, se describe de manera resumida la función de cada uno de ellos. Más adelante se explica con más detalle algunos de los que cumplen una función más relevante.

El encargado de recibir los voltajes provenientes del conector RJ9 en sus terminales 1 y 3 según el estado de colgado del auricular (0 V Colgado y aproximadamente 0.75 V para Descolgado) es un circuito disparador Schmitt con histéresis a base de transistores. Este circuito cumple la función de convertir los voltajes de entrada a niveles de voltaje adecuados para el manejo de la lógica alto-bajo en el resto del circuito; 1 V para un nivel lógico bajo (Colgado) y 4V para un nivel lógico alto (Descolgado). Este circuito es necesario debido a que sin éste, cuando se colgaba o descolgaba el auricular, se llegaban a presentar varios rebotes mecánicos en el interruptor de colgado o descolgado, generando variaciones de voltaje transitorios ocasionando a su vez que se repitiera varias veces el evento de descolgado y colgado en vez de uno de descolgado, lo cual hacia que se registraran indeseadas repeticiones de la operación en la base de datos. Con la histéresis, se reducía al mínimo la posibilidad de que aparecieran varios eventos de descolgado cuando se trataba de solo uno. La señal de salida del disparador es entregada a un circuito comparador configurado como tal dentro del microcontrolador que compara dicha señal con un voltaje de

referencia y el voltaje o nivel lógico (TTL) resultante en la salida del comparador es utilizado internamente por el *"firmware"* o programa del microcontrolador para generar un cambio de flujo dentro del programa según sea su nivel de salida, es decir; corriendo rutinas dedicadas al evento de descolgado del auricular (1 lógico en el comparador) y mientras permanezca así, sea capaz de registrar también las teclas de numeración pulsadas y detectadas por el decodificador de tonos, o una vez que se cuelga el teléfono, el comparador genera un 0 lógico que nuevamente cambia el flujo de ejecución del *"firmware",* todo esto siendo notificado a la aplicación a través de la unidad controladora de USB para ser detectado por la misma y registrado en la base de datos.

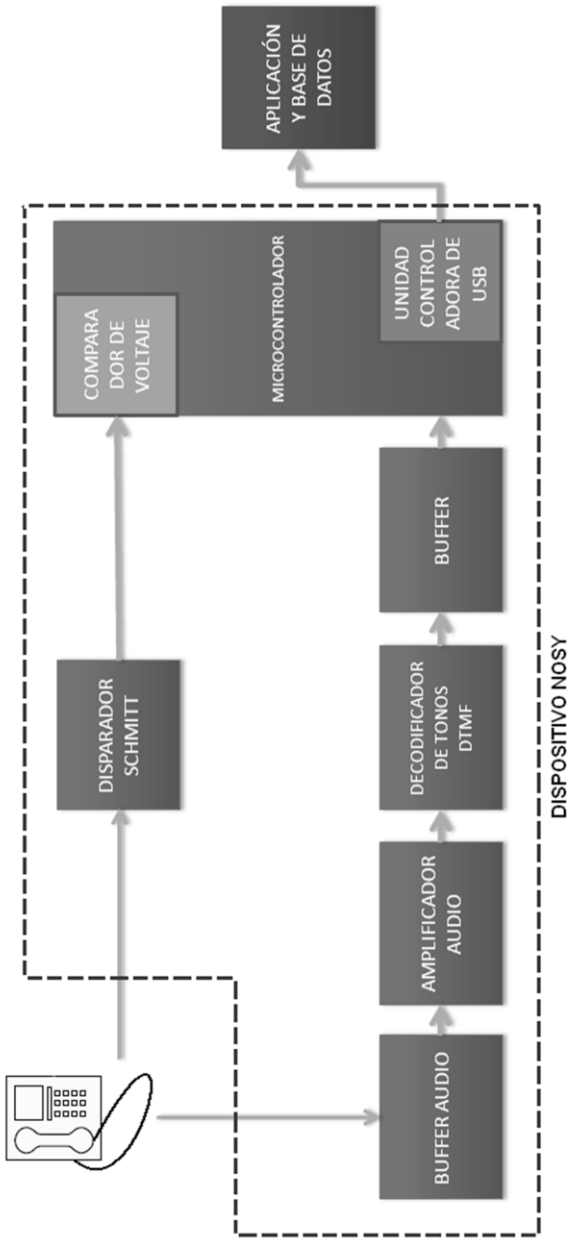


Figura 1.3 Proyecto a bloques del dispositivo

La señal de audio proveniente de las terminales 2 y 3 del RJ9, pasa primero a un amplificador operacional (buffer de audio) con la finalidad de generar la menor perturbación posible al audio del teléfono debido a la presencia del *"Nosy"*, esta es amplificada y entregada a un circuito integrado CMOS decodificador de tonos DTMF (Dual Tone Multi Frecuency) que es el encargado de interpretar los tonos que se escuchan en el audio del auricular al realizar una marcación. Cuando este decodificador logra interpretar los tonos de marcación, entrega en su salida un código binario BCD equivalente al tono o número marcado en el teléfono. Esta combinación de códigos binarios es entregada por medio de circuitos seguidores unitarios a base de amplificadores operaciones al microcontrolador para su procesamiento en conjunto con la señal proveniente del disparador Schmitt.

Los seguidores unitarios (buffer) son necesarios para garantizar el suministro necesario de corriente y niveles de voltaje adecuados en las terminales del microcontrolador debido a que la salida del decodificador es CMOS y las entradas al microcontrolador son TTL. Estos seguidores además tiene una configuración en su salida *"pull down"* que garantiza la correcta interpretación de los niveles lógicos en la entrada del microcontrolador provenientes del decodificador de tonos. Por otro lado, los seguidores aíslan al mismo del decodificador del microcontrolador impidiendo que la operación de ambos se vea afectada.

El microcontrolador es quien realiza la tarea de reinterpretar de manera combinada los datos provenientes del comparador de voltaje (Colgado, Descolgado) y del decodificador DTMF (marcación de tonos) para generar por medio de su *"firmware"* (programa interno) una serie de instrucciones identificables que puedan ser transmitidas a través de su controlador de USB incorporado y que puedan ser interpretadas de manera correcta por la aplicación final. Por su importancia, se describe el microcontrolador y la unidad controladora de USB en el capítulo 2.

Como parte final del proceso, se encuentra la aplicación que mediante código VBA y macros, es capaz de interceptar e interpretar la información que le proporciona el *"Nosy"*, para inmediatamente generar rutinas que controlen una interfaz y señalice los eventos detectados por el mismo *"Nosy"*. Además de esto, registra en la base de datos dichos eventos de manera estructurada para el uso administrativo que mejor convenga al área involucrada. Esta aplicación es descrita en el capítulo 3.

Cabe mencionar que el dispositivo *"Nosy"* simula un teclado externo convencional que se conecta vía USB a una computadora y que entrega la información igual que lo hace un teclado, es decir; pulsaciones de teclas, pero en vez de tener todo un conjunto de pulsaciones de teclas numéricas y alfanuméricas (101 teclas), envía de manera controlada solo un determinado número de pulsaciones o combinación de pulsaciones tanto de control como numéricas según el evento registrado por el *"Nosy"* y que deberán ser interceptadas y procesadas por la aplicación para su propio uso.

Esta combinación de teclas que envía el *"Nosy"* a la computadora se resumen en la Tabla 1a. mostrando los tipos de eventos que registra y los comandos que envía el dispositivo como consecuencia del evento.

Evento Telefónico	**Combinación teclas enviado por Nosy**	**Descripción**
Descolgado	CTRL + ALT + F	1. *"Nosy"* envía esta combinación de teclas para que Windows abra la aplicación si se encuentra cerrada o minimizada. Si ya está abierta, pero existen otras aplicaciones abiertas, la de interés la pasa a primer plano. Para que esto funcione hay que indicarle a Windows como debe comportarse con este comando.
	CTRL + E	2. Seguido de "CTRL + ALT+ F", *"Nosy"* envía un "CTR + E" que llama una macro de Access (aquí se supone que la aplicación ya está abierta) que abre un formulario con la interfaz de usuario para uso y registro de la llamada telefónica.
	CTRL + J	3. Después de enviar un "CTRL+E", *"Nosy"* envía un "CTRL+J" que activa una macro de Access y señaliza la interfaz de la aplicación en un estatus de descolgado.
Colgado	CTRL + ALT + F	1. Después de finalizada la conversación telefónica y simplemente se cuelga el auricular, *"Nosy"* envía nuevamente "CTRL+ALT+F" para llevar nuevamente a primer plano la aplicación, esto debido a que durante la conversación el usuario estuviera en alguna otra aplicación durante la llamada.
	CTR + E	2. Nuevamente, seguido de "CTR+ALT+F", *"Nosy"* envía "CTRL+E" para poner en primer plano el formulario o interfaz del registro telefónico.
	CTRL + D	3. Después de enviar "CTRL+E", *"Nosy"* envía "CTRL+D" Para correr una macro que pone la interfaz en estatus Colgado y lo registra en la base de datos.
Marcación numérica	0 – 9 en la misma forma que el teclado envía dichas pulsaciones	

Tabla 1a. Eventos registrados y comandos enviados por el dispositivo electrónico

1.4 El diagrama electrónico del dispositivo *"Nosy"*.

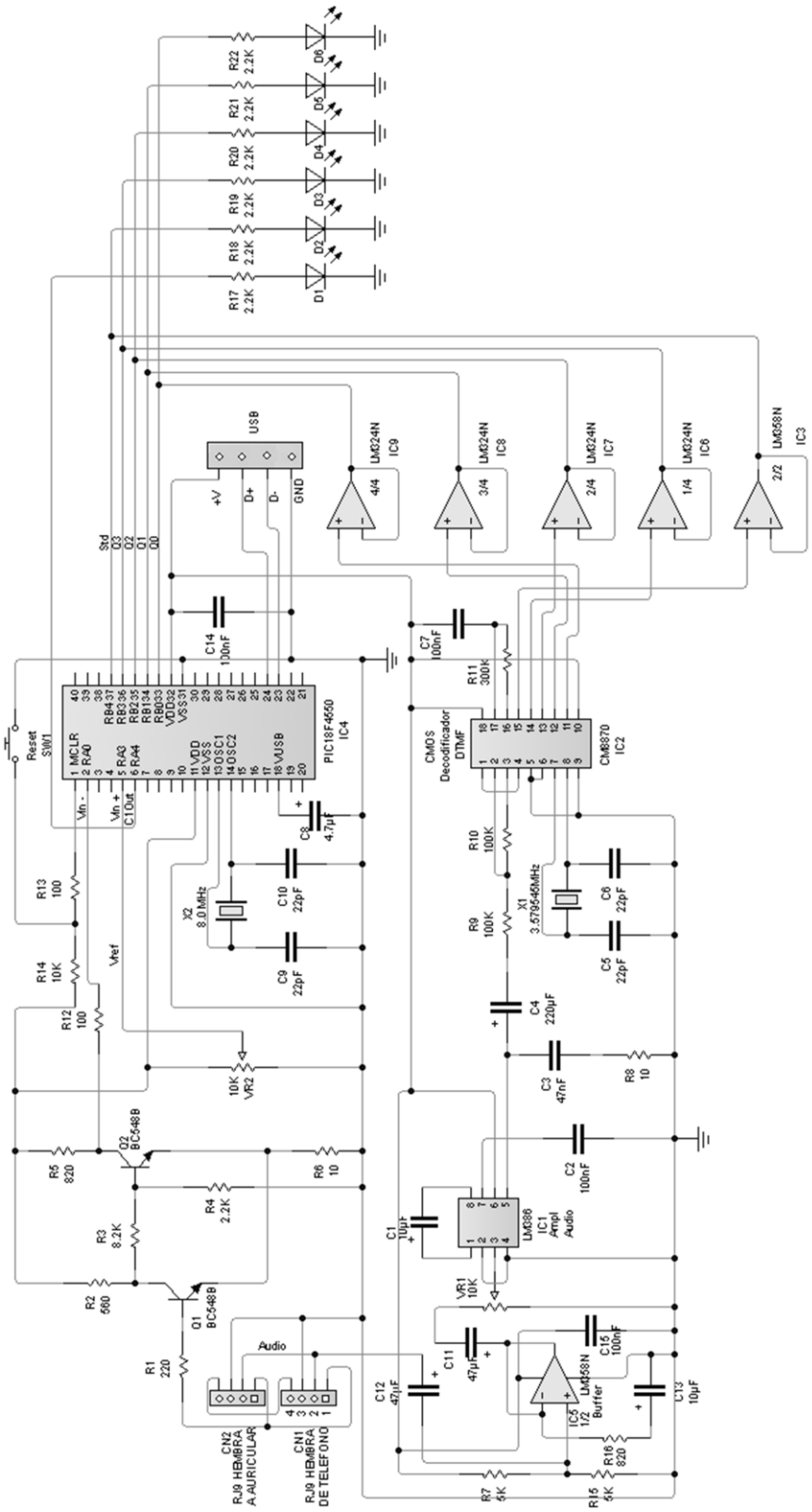


Figura 1.4 Diagrama electrónico dispositivo *"Nosy"*

1.5 El disparador Schmitt.

El disparador Schmitt es el encargado de convertir la señal de 0 volts (colgado) a un nivel lógico bajo y el de voltajes superiores a 0.75 volts (descolgado) a un nivel lógico alto (5 volts). Su configuración se muestra en la figura 1.5.

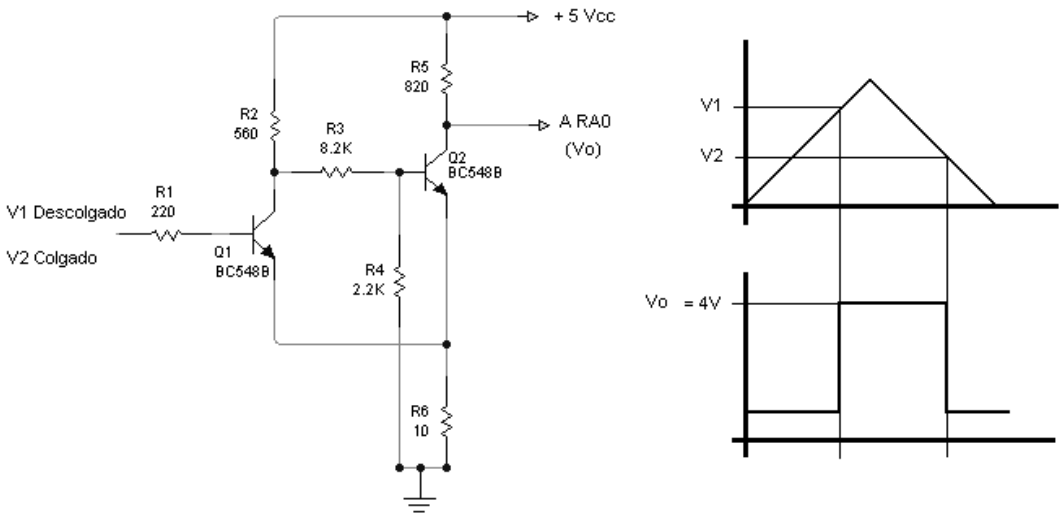


Figura 1.5 Disparador Schmitt y formas de onda a considerar

Para calcularlo, se procedió de la siguiente manera:

-Parámetros iniciales:

β Transistor BC548B= 200
Icarga = 10mA
V1 = 0.7 V (Disparo del circuito)
V2 = 0.6 V (Corte del circuito)
Vcc = 5 V
Vsal = Vo(0) = 4V
Vsal = Vo(1) = 1V
IR5 = 5 mA

Del voltaje necesario para cambiar de estado a Q1 cuando el mismo está en corte: V1 = VE2 + 0.5
Donde 0.5 es el voltaje necesario para hacer conducir el transistor.
VE2 es el voltaje de referencia entre emisor y tierra cuando Q2 está conduciendo.

VE2 = V1 – 0.5 = 0.7 – 0.5 = 0.2V ; voltaje de referencia debido a Q2 conduciendo cuando Q1 está en corte.
De donde, IC2 = IR5 + Icarga = 5mA + 10mA = 15mA

IE2 ≈ IC2

$$R6 = \frac{VE2}{IE2} = \frac{0.2V}{15mA} = 13.33\Omega$$

Como Q1 está en corte y Q2 está conduciendo, está presente Vo(1) = 1 V.

$$R5 = \frac{Vcc - Vo\ 1}{IR5} = \frac{5V - 1V}{5mA} = \frac{4V}{5mA} = 800\Omega$$

$$IB2 = \frac{IC2}{\beta} \approx \frac{15mA}{200} = 75\mu A$$

$IB2 = I1 - I2$, donde

I1 es la corriente que circula de sobre R2 y R3 a la base de Q2.
I2 es la corriente que circula sobre R4 cuando Q1 está en corte.

$$IB2 = \frac{Vcc - VB2}{R2 + R3} - VB2/R4$$

$$VB2 = VE2 + 0.7 = 0.2V + 0.7 = 0.9V$$

Cuando Q1 está saturado y Q2 está en corte:

$$IR2 \approx IR6 = IE1$$

Del voltaje necesario para poner en corte Q1: $V2 = VE1 + 0.5V$
Donde VE1 es el voltaje de referencia entre emisor y tierra cuando Q1 está conduciendo.

$$VE1 = V2 - 0.5 = 0.6 - 0.5 = 0.1V$$

$$IE1 = \frac{VE1}{R6} = \frac{0.1V}{13.33\Omega} = 7.5mA$$

$$R2 = \frac{Vcc - VE1 + 0.3}{IE1} = \frac{5 - 0.1 + 0.3}{7.5mA} = \frac{5 - 0.4}{7.5mA} = \frac{4.6}{7.5mA} = 613.33\Omega$$

Aplicando el criterio de ganancia:

$$Av = \frac{v_y}{v_x} \approx \frac{-\beta R2}{R2 + R3} \gg 1$$

$$R3 = \frac{\beta - 1\ R2}{15} = \frac{199\ \ 613.33}{15} = 8136.84\Omega$$

$$R4 = \frac{VB2}{\frac{Vcc - VB2}{R2 + R3} - IB2} = \frac{0.9}{\frac{5 - 0.9}{613.33 + 8136.84} - 75\mu A} = 2287.16\Omega$$

$$R1 = \frac{\beta + 1\ R6}{10} = \frac{201\ 13.33}{10} = 267.93\Omega$$

Los valores reales de los resistores que se ocuparon en el disparador se resumen en la tabla 1.b.

Resistor	Valor calculado (Ω)	Valor comercial (Ω)
R1	267.93	220K
R2	613.33	560
R3	8.13K	8.2K
R4	2287	2.2K
R5	800	820
R6	13.33	10

Tabla 1.b Valores finales de resistores para el disparador Schmitt

1.6 El decodificador DTMF.

Para fines del proyecto, se requería un circuito que detectara y codificara las frecuencias de los tonos de marcación que se escuchaban en la señal de audio provenientes del conector del auricular y utilizadas para uso en el *"Nosy"*.

El sistema de marcación por tonos o DTMF (Dual-Tone Multi-Frequency) es en realidad un protocolo que regular las frecuencias y forma de marcación de los tonos que se transmiten a través de las redes de telefonía comerciales.

El sistema de marcación consiste en el envío de dos tonos de diferente frecuencia que identifican inequívocamente cada tecla pulsada en el aparato telefónico. El par de frecuencias que corresponde a cada tecla se muestran en la tabla 1.c.

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

Tabla 1.c Correspondencia Par de frecuencias protocolo DTMF

Para llevar a cabo la tarea del *"Nosy"*, se tendría que haber construido filtros pasa banda que pudieran detectar no solo una, sino dos frecuencias diferentes para poder identificar la tecla marcada lo que incrementaría el tiempo y complejidad del diseño. Afortunadamente, existen circuitos integrados comerciales ya diseñados para cumplir esta tarea. Uno de los más populares es el circuito integrado CMOS CM8870PI que además de ser relativamente económico, reconoce los tonos presentes en su entrada y los codifica entregando una salida binaria según corresponda a la tecla pulsada. Su salida es mostrada en la tabla 1.d.

Tecla	Std	Q3	Q2	Q1	Q0
No cambia	0	X	X	X	X
1	1	0	0	0	1
2	1	0	0	1	0
3	1	0	0	1	1
4	1	0	1	0	0
5	1	0	1	0	1
6	1	0	1	1	0
7	1	0	1	1	1
8	1	1	0	0	0
9	1	1	0	0	1
0	1	1	0	1	0
.	1	1	0	1	1
#	1	1	1	0	0
A	1	1	1	0	1
B	1	1	1	1	0
C	1	1	1	1	1
D	1	0	0	0	0

Tabla 1.d Valores lógicos de salida del CI CM8870PI

La salida consta de 4 bits Q0 – Q3 del tipo enganchado (latch). La salida adicional *"Std"* en realidad es un bit que cambia de 1 a 0 e inmediatamente a 1 (parpadeo) cada vez que se teclea o pulsa un número. Como la salida BCD está enganchada, el Std toma un papel muy importante en la operación del *"Nosy"*, ya que es el que le indica al microcontrolador el instante en que se está pulsando una tecla en el teléfono. El microcontrolador lo detecta y registra el valor de la tecla pulsada tomando los valores BCD Q0-Q3.

El circuito electrónico del CM8870 es el que proporciona el fabricante con muy ligeras variantes el cual se muestra en la figura 1.6.

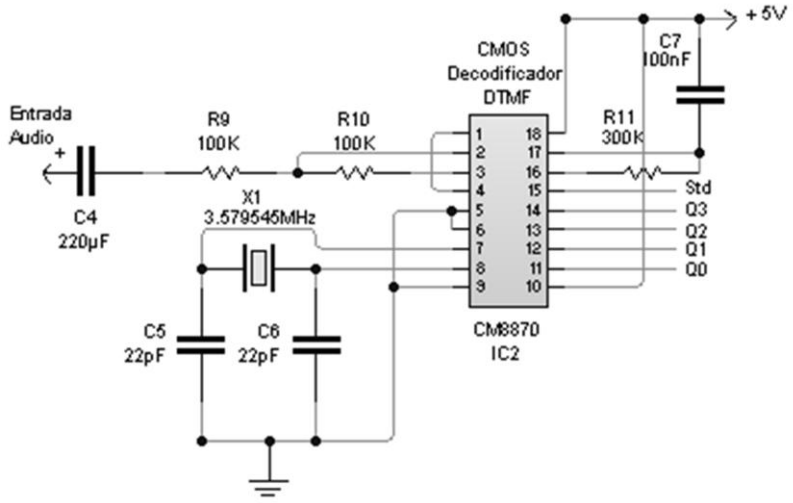


Figura 1.6 Circuito Decodificador DTMF

CAPITULO 2

GENERALIDADES DEL MICROCONTROLADOR PIC18F4550

2.1 Descripción.

El microcontrolador representa una de las piezas fundamentales del proyecto *"Nosy"*. Es el encargado de interpretar la combinación de datos provenientes del disparador Schmitt y del decodificador DTMF como consecuencia de los eventos registrados en el aparato telefónico, para posteriormente convertirlas en un nuevo código que será transmitido a la computadora a través del controlador USB mediante una trama de datos reconocida por Windows, simulando un teclado que solo envía cierta combinación de teclas a las que deberá responder la aplicación o base de datos. Para lograr esto, fue necesario configurar y programar el microcontrolador (firmware).

Para este proyecto y su posterior implementación, fue utilizado el microcontrolador PIC18F4550 (PIC) del fabricante Microchip Technology Inc. La razón de la elección de este microcontrolador se debe principalmente a lo siguiente:

- Previo conocimiento sobre la tecnología y programación de microcontroladores PIC, particularmente de las familias 16F.
- La familia 18F en algunos modelos (PIC18F2550, PIC18F4550) tienen un controlador USB integrado lo que facilitaría la implementación en el programa del dispositivo y la comunicación con la computadora y la aplicación.
- Relativo bajo costo (actualmente, alrededor de unos 100 a 120 pesos en el mercado nacional). En realidad el costo total del *"Nosy"* oscila alrededor de unos $500.00 MX, lo que si comparamos el costo contra el beneficio de su uso y aclarando que no tiene fines comerciales resulta excelente su construcción, aunque el diseño e implementación resultó ser mucho más compleja de lo esperado.
- Facilidades, literatura basta y utilerías del fabricante Microchip proporcionadas de forma gratuita en su página web, lo que permitió su desarrollo e implementación.
- Un compilador C gratuito proporcionado por Microchip e incluido dentro del entorno de desarrollo MPLAB que permitiría programar el microcontrolador en este lenguaje en vez de utilizar los nemónicos del lenguaje ensamblador del propio microcontrolador.

El PIC18F4550 es un microcontrolador que pertenece a la familia de los PIC18CXXX ubicados dentro de la gama alta con un bus de datos de 8 bits y un bus de instrucciones de 16 bits. El microcontrolador está basado en la arquitectura Harvard que mantiene la memoria de datos separada de la memoria de instrucciones (programa).

Entre las características principales del PIC18F4550 proporcionadas por el fabricante se destacan:

- Módulo de comunicaciones Universal serial Bus (USB), que cumple con las especificaciones USB 2.0 y soporta velocidades del tipo "Baja" y "Completa". También incorpora su propio

"transceptor" y regulador de voltaje interno de 3.3V. Es capaz de soportar "transceptores" y reguladores de voltajes externos.

- Varias formas de configurar el oscilador y varios modos de conectar un reloj externo.
- Memoria Flash de 32Kb para la memoria del programa y EEPROM para datos.
- Set de 75 instrucciones de programa en ensamblador y la posibilidad de programarlo en lenguaje C.
- Módulo CCP (Capture/Compare/PWM).
- Unidad USART (Universal Synchronous Asynchronous Receiver/Transmitter) accesible.
- 13 Convertidores A/D de 10 bits.
- Puerto ICD/ICSP (In-circuit debugging / In circuit serial programming) dedicado.
- 5 puertos bidireccionales de I/O (In / Out) con entradas TTL y salidas CMOS.
- Puerto paralelo para Streaming (lectura en continuo utilizado para aplicaciones USB).
- 2 Comparadores de voltaje con entradas análogas y salidas digitales con posibilidad de multiplexar las entradas.
- Corriente de bajada/suministro en sus puertos: 25mA/25mA
- 3 interrupciones externas.

La conexión e implementación de todas sus funciones se muestra en la figura 2.1.

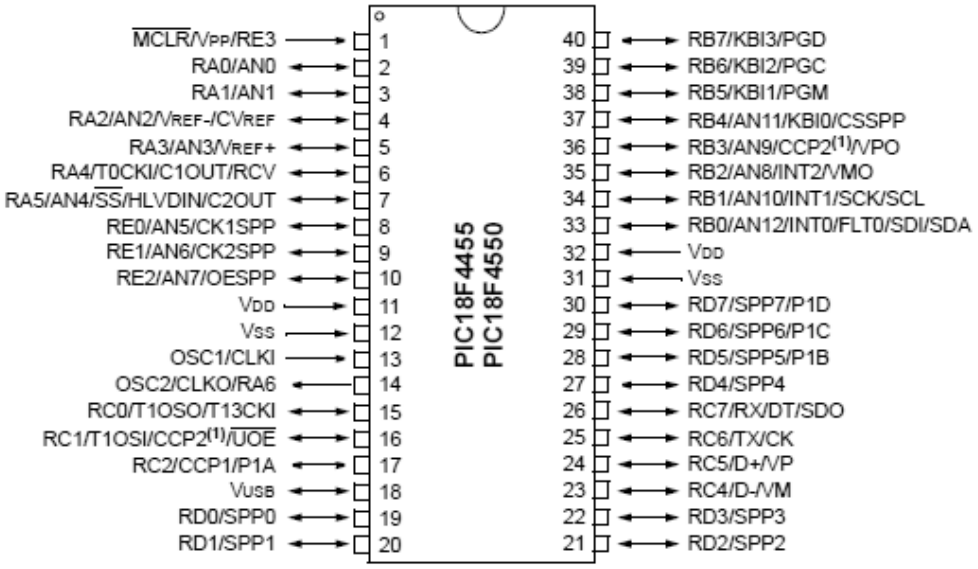


Figura 2.1 Configuración PIC18F4550 (3)

Como se observa en la figura 2.1, algunas de las características mencionadas se encuentran multiplexadas en las terminales con otras características. Para usar algunas de estas características, es necesario establecer la configuración e iniciarlas en el *"firmware"* del microcontrolador.

Para fines del diseño, se ocuparon únicamente las siguientes características o funciones del microcontrolador:

- Comparador de voltaje con salida digital configurando RA0 y RA3 del Puerto A (puesto I/O) como entradas (terminales 2 y 5) y RA4 como salida, además de establecer en los registros específicos que RA0 actúe como V_{IN-} y RA3 como $V_{IN+} = V_{REF}$, además de que RA4 adquiriera la funcionalidad de C1out (Salida de comparador 1).
- Configuración del Puerto B para recibir las señales provenientes del decodificador DTMF.
- Unidad controladora de USB, a través de los pines 23 y 24 para las señales de salida +D y –D y la terminal 18 para el V_{USB}.
- Selección y configuración del tipo de oscilador y frecuencia del mismo para uso tanto por la unidad central de proceso como por la unidad controladora de USB.

Para la implementación de estas funciones, la programación y obtención del *"firmware"* correspondiente es necesario conocer como está estructurada la memoria del microcontrolador.

Como se mencionó anteriormente, el microcontrolador dispone de una memoria Flash de 32Kb para almacenar las instrucciones de programa. Cada instrucción se compone de 16 bits.

La memoria de programa está mapeada tal como se muestra en la figura 2.2.

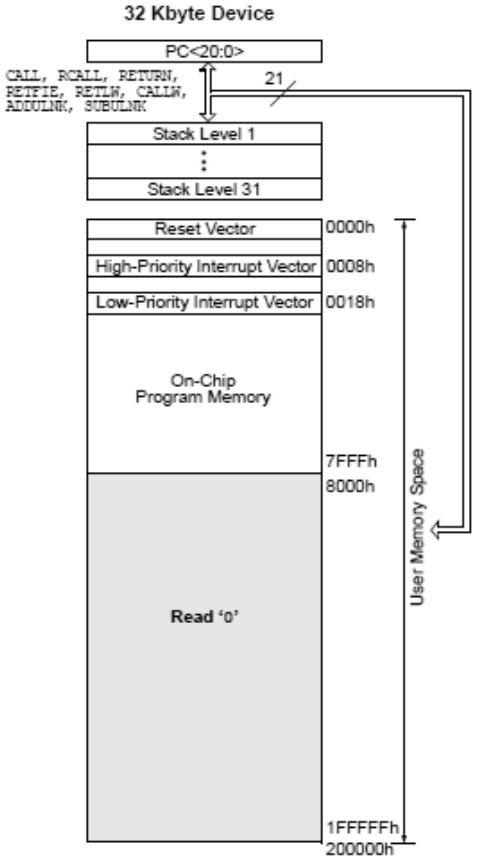


Figura 2.2 Mapa de memoria de programa del PIC18F4550 (3)

La memoria del programa está compuesta por instrucciones que ocupan 2 bytes de memoria cada una de ellas.

Las primeras direcciones de la memoria están reservadas para los vectores de interrupción.

La dirección 0000h es reservada para el vector de reinicio de programa.

La dirección 0008h es reservada para el vector de Interrupción de alta prioridad (utilizado en nuestra aplicación para las interrupciones por USB).

La dirección 0018h es reservada para interrupciones de baja prioridad. El resto del espacio de memoria es ocupado por el programa.

El mapa cubre hasta 2 Mega bytes posiciones, aunque solo están implementadas 32 Kilo bytes en el PIC18F4550.

El PIC maneja 3 vectores de interrupción debido a:

- Vector de reinicio de programa. Cuando se produce un reinicio ya sea por la introducción de un pulso bajo (0) en la terminal 1 (MCLR) del PIC o por software, el contador de programa del microcontrolador apunta a esta dirección de memoria reinicializando todo el sistema.
- Vectores de interrupción de alta y baja prioridad. El microcontrolador se puede programar y configurar en ciertos registros específicos (SFR) correspondientes para aceptar interrupciones externas o internas, es decir; el microcontrolador puede abandonar el flujo normal de su programa principal para ejecutar otras subrutinas de programación derivados de algún cambio en algunas de las terminales específicas de sus puertos I/O, del desbordamiento de un contador de pulsos, temporizador o de algún requerimiento de atención solicitado por la unidad de USB. El tipo de interrupción que nos interesa es el relativo a los requerimientos del mismo hechos por la unidad de USB y el microcontrolador se programa para ello. La asignación de alta prioridad se debe a que tiene mayor peso este vector de interrupción al de baja prioridad. Una interrupción de alta prioridad interrumpe una subrutina de baja prioridad en cualquier momento. Cuando una interrupción ocurre, el contador de programa apunta a la dirección que contiene el vector de baja o alta prioridad apilando la ultima instrucción ejecutada en la pila (de 31 niveles). Generalmente estos vectores contienen la dirección donde se encuentran las rutinas que han ejecutar. Cuando estas terminan de ejecutarse, el contador de programa retoma el ultimo registro en la pila para retornar a la siguiente instrucción que debía ejecutar antes de la interrupción. Las banderas y registros relacionados de control relacionados con la interrupción, también se restablecen para estar a la espera cuando vuelva a ocurrir alguna interrupción nuevamente.

Por otro lado, el microcontrolador dispone de una memoria RAM de 2Kbytes donde cada campo es de 8 bits. La memoria RAM del PIC18F4550 está mapeada según se muestra en la figura 2.3.

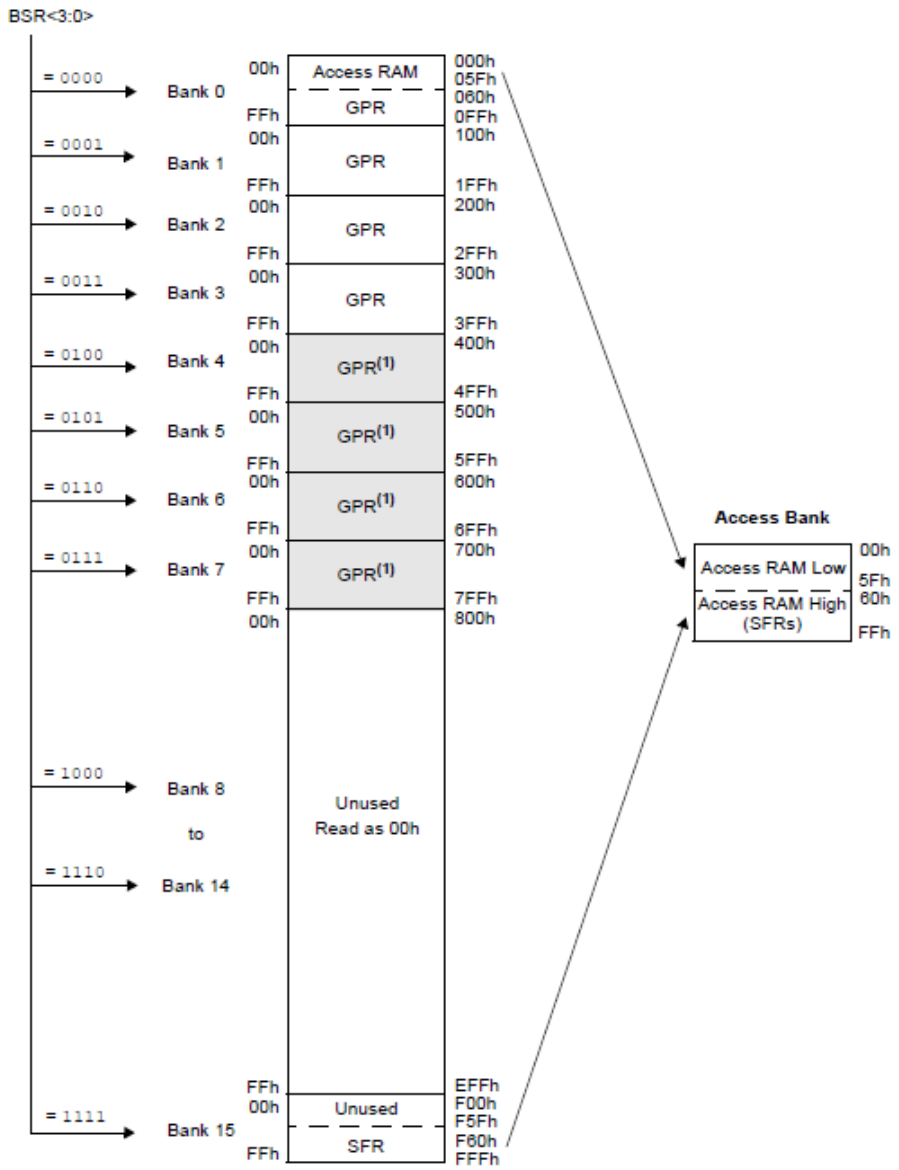


Figura 2.3 Mapa de memoria de datos del PIC18F4550 (3)

El mapa para la memoria RAM de los PIC18F esta dividida en 16 bancos, cada uno de 256 bytes que pueden ser accedidos por direcciones de 12 bits pudiendo este campo acceder hasta 4096 bytes de memoria, sin embargo solo 2048 están implementadas físicamente.

De esos 16 bancos, 4 de ellos (del 4 al 7) están reservados para su uso como *"buffer"* de las operaciones del controlador USB cuando esta configurado para ser utilizado. Este espacio de memoria también es compartida con el microcontrolador para transferir datos directamente entre éste y la unidad de USB. Cuando el USB no es habilitado, estos bancos de memoria pueden ser utilizados como registros de propósito general (GPR), es decir, almacenar datos.

Para direccionar cada banco, es necesario el uso del registro selector de banco (BSR). Si se deseara programar el microcontrolador en lenguaje ensamblador, el uso de este registro tomaría alta prioridad. Para fines del proyecto *"Nosy"*, se utilizó el compilador C que interpreta las funciones requeridas y minimiza el tener que usar a detalle este tipo de registros y el ensamblador nativo.

El banco 15 del mapa de la memoria RAM es asignada para almacenar los registros de propósito específico (SFR) y son utilizados por la unidad central de procesamiento en combinación con las instrucciones del programa, algunos para controlar la operación y funcionalidad del microcontrolador y otros para controlar la funcionalidad de los periféricos que contiene.

Los SFR del PIC18F4550 completos proporcionados por el fabricante son descritos en la hoja de datos del microcontrolador. Para fines del proyecto solo se describirán los requeridos en los apartados siguientes.

2.2 Puerto I/O A.

El Puerto A es utilizado para habilitar un comparador de voltaje donde RA0 se convertirá en V_{IN-} y RA3 se convertirá en $V_{IN+} = V_{REF}$. RA4 se convertirá en C1out del comparador del voltaje. Para llevarlo a cabo, es necesario primero, configurar las terminales RA0 y RA3 como entradas y RA4 como salida. Esto se lleva a cabo modificando la configuración del SFR TRISA.

Dentro del archivo *"Keyboard.c"* que contiene la aplicación principal (más adelante se explicara sobre el proyecto en el MPLAB y el contenido de los archivos del proyecto) y dentro del *"main(void)"* (función principal en lenguaje C) en la fase de inicialización de variables, se establecen los valores iniciales para TRISA de la siguiente manera:

TRISA=0X0F; //en binario: 00001111 donde bit=1 significa "entrada" y bit=0 significa "salida".

De acuerdo a la configuración en el SFR para TRISA, se esta colocando RA0 a RA3 como entradas y RA4 a RA6 como salidas de acuerdo a la estructura mostrada en la tabla 2.a.

Nombre	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
TRISA	-	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

Tabla 2.a Registro SFR para configurar Puerto A

2.3 Comparador de voltaje.

Una vez que se han configurado las terminales del Puerto A RA0, RA3 como entradas y RA4 como salida es necesario habilitar y configurar el comparador de voltaje. Para ello es necesario modificar el SFR CMCON que tiene la configuración mostrada en la tabla 2.b.

Nombre	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
R/W	R-O	R-O	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1

Tabla 2.b Registro SFR CMCON

Donde:

C2OUT: Salida de comparador 2.

C1OUT: Salida de comparador 1. Accediendo este registro y al bit 6, podemos tomar lectura del valor de la salida del comparador para ser utilizado internamente por el *"firmware"*.

C2INV: En este bit se establece el nivel de salida que C2 debe cumplir de acuerdo a las combinaciones mostradas en la tabla 2.c.

C2OUT	C2INV=0	C2INV=1
1	$V_{IN+} > V_{IN-}$	$V_{IN+} < V_{IN-}$
0	$V_{IN+} < V_{IN-}$	$V_{IN+} > V_{IN-}$

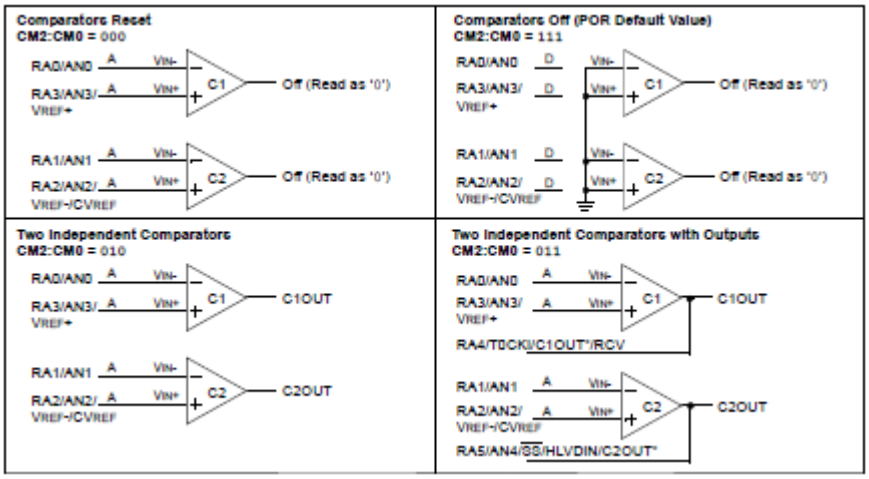
Tabla 2.c Configuración modo salida Comparador 2

C1INV: En este bit se establece el nivel de salida que C1 debe cumplir de acuerdo a las combinaciones mostradas en la tabla 2.d.

C1OUT	C1INV=0	C1INV=1
1	$V_{IN+} > V_{IN-}$	$V_{IN+} < V_{IN-}$
0	$V_{IN+} < V_{IN-}$	$V_{IN+} > V_{IN-}$

Tabla 2.d Configuración modo salida Comparador 1

CM2:CM0: Modo del comparador. De acuerdo a las hojas de datos del fabricante, tenemos las combinaciones posibles mostradas en la figura 2.4:



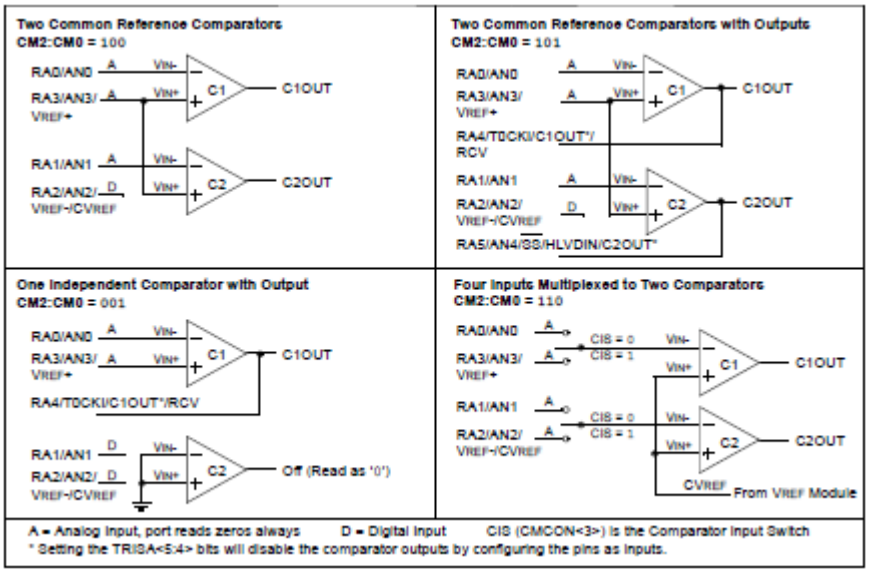


Figura 2.4 Modo de Comparador (3)

CIS: Cuando CM2:CM0= 110,

1 = C1 V_{IN}_ conecta a RA3
C2 V_{IN}_ conecta a RA2

0= C1 V_{IN}_ conecta a RA0
C2 V_{IN}_ conecta a RA1

Para fines del diseño del *"Nosy"*, se estableció la configuración para el comparador de acuerdo a lo que se describe en los párrafos siguientes.

En el archivo de cabecera "HardwareProfile.h" se definieron nuevas variables de acuerdo a las definiciones de las estructuras proporcionadas para el PIC18F4550 en el archivo "p18f4550.h":

```
#define cmConf0  CMCONbits.CM0     //bit 0 CMCON
#define cmConf1  CMCONbits.CM1     //bit 1 CMCON
#define cmConf2  CMCONbits.CM2     //bit 2 CMCON
#define cmInvC1  CMCONbits.C1INV   //bit 4 CMCON
#define COMPARADOR  PORTAbits.RA4  //Variable para ser utilizada en main()
```

Luego, dentro del archivo principal "Keyboard.c" dentro de la función *"main()"* se inician las variables de la forma siguiente:

```
cmConf0=1;        //bit 0 CMCON
cmConf1=0;        //bit 1    CMCON
cmConf2=0;        //bit 2 CMCON
cmInvC1=1;        //bit 4 CMCON
```

De esta forma, durante la etapa de inicialización del microcontrolador, se configura el comparador como Comparador independiente con salida a RA4 y salida C1 tal que:
C1out = 1 cuando $V_{IN-} > V_{IN+}$ y C1out = 0 cuando $V_{IN-} < V_{IN+}$

De tal manera que cuando el voltaje proveniente del disparador Schmitt aplicado en la terminal 2 sea mayor que el voltaje de referencia prefijado y suministrado a la terminal 5, se tendrá una salida en la terminal 4 de 1 lógico cuyo valor puede ser leído en el programa del *"firmware"* accediendo al SRF CMCON bit 6 o tomado físicamente de la terminal 4. En el circuito, a esta terminal se conecta un *"led"* para indicar el estado de colgado y descolgado y es una manera visible de monitorear la salida del comparador.

2.4 Puerto I/O B.

Este puerto es utilizado para introducir al microcontrolador la información proveniente del decodificador DMTF pasando antes por amplificadores operacionales seguidor unitario. Para ello solo basta configurar en el registro SFR TRISB los bits necesarios como entradas de datos (1 para entradas y 0 para salidas) y puedan después ser leídos por el microcontrolador a través del registro PORTB. La configuración de ambos registros es mostrada en la tabla 2.e.

Nombre	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0

Tabla 2.e Registro SFR para configurar Puerto B

En el archivo "HarwareProfile.h" se define la variable "SWstd" para ser utilizada dentro del *"main()"* del programa principal y es utilizada para identificar el bit STD proveniente del decodificador DTMF:

```
#define SWstd                PORTBbits.RB4
```

Dentro del main() del archivo "Keyboard.c" en la fase de inicialización de variables se configura el registro SFR TRISB de la siguiente manera:

```
TRISB=0X1F;        //Port B as Input.
```

En binario corresponde al valor 00011111 que establece las terminales RB0 a RB4 como entradas y las terminales RB5 a RB7 como salidas. Esto resulta conveniente debido a que las terminales RB5 a RB7 no tendrán conexión con ningún otro dispositivo en el circuito y solo son de interés las lecturas obtenidas en RB0:RB4. Estas terminales además de estar conectadas físicamente a la salida de amplificadores operacionales que lo aísla del decodificador DTMF, también tienen conectados resistores en serie con *"leds"* en una configuración *"pull-down"* que nos permite darle más estabilidad a los niveles lógicos obtenidos de la salida de cada amplificador operacional.

La lectura de los valores de entrada en las terminales RB0:RB4 se obtiene leyendo los datos en el registro SFR PORTB. Dentro del *"main()"* del programa del proyecto para el *"firmware"* la forma en que se leen los valores del PORTB se muestra en el siguiente fragmento de código (anexo 1):

```
switch (PORTB)
{
case 0x11:        //si PORTB= 10001(STD=1 + numero 1 marcado)
KEY2=0x1E;        //asigna variable KEY=1E (usage ID para keyboard 1)
break;
case 0x12:        //si PORTB= 10010(STD=1 + numero 2 marcado)
KEY2=0x1F;        //asigna variable KEY=1F (usage ID para keyboard 2)
break;
case 0x13:        //si PORTB= 10011(STD=1 + numero 3 marcado)
KEY2=0x20;        //asigna variable KEY=20h (usage ID para keyboard 3)
break;
case 0x14:        //si PORTB= 10100(STD=1 + numero 4 marcado)
KEY2=0x21;        //asigna variable KEY=21h (usage ID para keyboard 4)
break;
case 0x15:        //si PORTB= 10101(STD=1 + numero 5 marcado)
KEY2=0x22;        //asigna variable KEY=22h (usage ID para keyboard 5)
break;
case 0x16:        //si PORTB= 10110(STD=1 + numero 6 marcado)
KEY2=0x23;        //asigna variable KEY=23h (usage ID para keyboard 6)
break;
case 0x17:        //si PORTB= 10111(STD=1 + numero 7 marcado)
KEY2=0x24;        //asigna variable KEY=1E (usage ID para keyboard 1)
break;
case 0x18:        //si PORTB= 11000(STD=1 + numero 8 marcado)
KEY2=0x25;        //asigna variable KEY=25h (usage ID para keyboard 8)
break;
case 0x19:        //si PORTB= 101001(STD=1 + numero 9 marcado)
KEY2=0x26;        //asigna variable KEY=26h (usage ID para keyboard 9)
break;
case 0x1A:        //si PORTB= 11010(STD=1 + numero 0 marcado)
KEY2=0x27;        //asigna variable KEY=27h (usage ID para keyboard 0)
break;
case 0x1B:        //si PORTB= 11011(STD=1 + numero * marcado)
KEY2= 0x04;       //asigna variable KEY=04h (usage ID para keyboard a)
break;
case 0x1C:        //si PORTB= 11100(STD=1 + numero # marcado)
KEY2= 0x05;       //asigna variable KEY=05h (usage ID para keyboard b)
break;
default:
break;
}
```

En la sección 2.7.10 correspondiente al HID de la comunicación USB se describirá a detalle el término *"Usage ID"* (ID Uso) para teclados.

2.5 El oscilador de trabajo y configuración inicial.

Los microcontroladores de la serie 18F manejan una variada gama de configuraciones de reloj para su funcionamiento. De manera general utilizan 4 modos o tipos de osciladores y se muestran en la tabla 2.f.

Modo	Rango de frecuencias	Descripción
LP	32KHz – 200 KHz	Cristal o resonador cerámico de baja frecuencia
XT	200KHz – 4 MHz	Cristal o resonador cerámico de frecuencia media
HS	4 MHz – 25 MHz	Cristal o resonador cerámico de alta velocidad
Externo		Pulsos de reloj de fuente externa

Tabla 2.f Modos genéricos de osciladores en PIC

Específicamente, el PIC18F4550 puede operar en 12 distintos modos de configurar el oscilador, 4 de estos involucran el uso de dos osciladores a la vez cuyos modos (conectados a OSC1 y OSC2) básicos son los que se muestran en la tabla 2.g.

Modo	Descripción
XT	Cristal / Resonador (solo 4 Mhz para PICF18F4550)
XTPLL	Cristal / Resonador (solo 4 Mhz para PICF18F4550) con PLL habilitado.
HS	Cristal / Resonador de alta velocidad (8MHz, 16MHz o 20MHz para PIC18F4550)
HSPLL	Cristal / Resonador de alta velocidad con PLL habilitado
EC	Reloj externo con Fosc/ Output
ECIO	Reloj externo con I/O en RA6
ECPLL	Reloj externo con PLL habilitado y FOSC/4 Salida en RA6
ECPIO	Reloj externo con PLL habilitado, I/O en RA6
INTHS	Oscilador Interno usado como fuente de reloj del microcontrolador, oscilador HS usado como fuente de reloj USB.
INTIO	Oscilador interno usado como fuente de reloj del microcontrolador, Oscilador EC usado como fuente de reloj USB, I/O digital en RA6.
INTCKO	Oscilador interno usado como fuente de oscilador del microcontrolador, Oscilador EC usado como fuente de reloj para USB, salda FOSC/4 en RA6.

Tabla 2.g Modos de oscilador para el PIC18F4550

El modo del oscilador y las opciones de PLL (Phase Locked Loop) se establece en los registros de configuración CONFIG1L y CONFIG1H. Los bits FOSC3:FOSC0 son usados para seleccionar los modos descritos anteriormente.

Por otro lado, la unidad de USB para su operación, requiere trabajar con un reloj de 48MHz independiente de la velocidad de reloj a la que trabaje el microcontrolador.

Si se conecta un cristal en OSC1:OSC2 para operación del microcontrolador, es posible utilizar esta frecuencia de operación configurando apropiadamente el microcontrolador para obtener de este mismo los 48MHz requeridos por la unidad de USB.

El PLL es un divisor de frecuencia programable y existes varias unidades contenidas en el microcontrolador para soportar varias configuraciones de reloj.

Si se tiene un cristal de 8MHz conectado a OSC1:OSC2 para operación del microcontrolador, para obtener los 48MHz, se debe de configurar de tal manera el PIC, que se active un PLL para dividir esa frecuencia entre dos (el PLL puede ser configurado para dividir la frecuencia ÷1, ÷2, ÷3, ÷4, ÷5, ÷6, ÷10 hasta ÷12) para obtener 4MHz. Es necesaria esta frecuencia, ya que el microcontrolador tiene un circuito multiplicador a 96MHz que requiere 4MHz forzosamente en su entrada. Después de haber obtenido estos 96MHz, son divididos nuevamente entre 2 para obtener finalmente los 48MHz requeridos por la unidad de USB.

Para fines del diseño, es necesario primero configurar el modo del oscilador y la frecuencia a utilizar. Si la unidad de USB es habilitada, es necesario también habilitar el PLL y seleccionar la división que realizará éste.

2.6 Los bits de configuración del microcontrolador.

En gamas más bajas de los microcontroladores PIC, por ejemplo los 16F, este conjunto de bits especiales para configuración es conocido como "palabra de configuración". En la gama de los 18F, los bits de configuración se ubican en la localidad de memoria de programa a partir de la dirección 300000h a la 3FFFFFh que no es visible el mapa de memoria descrito con anterioridad, por lo que no es posible acceder a este rango de memoria directamente por el usuario. Es un espacio de memoria que solo puede ser accedido en el momento que se programa el microcontrolador y contiene los registros de configuración (no son registros de propósito específico ubicados en la RAM) que permiten establecer algunas configuraciones básicas del funcionamiento del microcontrolador como:

- Selección del oscilador.

- Fuentes de reinicialización:
  . *POR (Power on Reset)*.- Reinicio al encender el dispositivo.
  . *PWRT(Power-up Timer).*- Temporizador de encendido.
  . *OST(Oscilador Start-up Timer).*- Temporizador Arranque Oscilador.
  . *BOR (Brown-out Reset).*- Reinicio por fallo en alimentación.

- Interrupciones:
  . *WDT (Watchdog Timer).*- Perro Guardián.
  . Monitor de fallo de la fuente de reloj externa.
  . *Protección de código.*
  . *Localizaciones ID*

Las características anteriores pueden ser configurados en los registros mostrados en la tabla 2.h.

File Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default/ Unprogrammed Value
300000h	CONFIG1L	—	—	USBDIV	CPUDIV1	CPUDIV0	PLLDIV2	PLLDIV1	PLLDIV0	--00 0000
300001h	CONFIG1H	IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0	00-- 0101
300002h	CONFIG2L	—	—	VREGEN	BORV1	BORV0	BOREN1	BOREN0	$\overline{\text{PWRTEN}}$	--01 1111
300003h	CONFIG2H	—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN	---1 1111
300005h	CONFIG3H	MCLRE	—	—	—	—	LPT1OSC	PBADEN	CCP2MX	1--- -011
300006h	CONFIG4L	$\overline{\text{DEBUG}}$	XINST	ICPRT(3)	—	—	LVP	—	STVREN	100- -1-1
300008h	CONFIG5L	—	—	—	—	CP3(1)	CP2	CP1	CP0	---- 1111
300009h	CONFIG5H	CPD	CPB	—	—	—	—	—	—	11-- ----
30000Ah	CONFIG6L	—	—	—	—	WRT3(1)	WRT2	WRT1	WRT0	---- 1111
30000Bh	CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	—	111- ----
30000Ch	CONFIG7L	—	—	—	—	EBTR3(1)	EBTR2	EBTR1	EBTR0	---- 1111
30000Dh	CONFIG7H	—	EBTRB	—	—	—	—	—	—	-1-- ----
3FFFFEh	DEVID1	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0	xxxx xxxx(2)
3FFFFFh	DEVID2	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3	0001 0010(2)

Legend: x = unknown, u = unchanged, - = unimplemented. Shaded cells are unimplemented, read as '0'.
Note 1: Unimplemented in PIC18FX455 devices; maintain this bit set.
2: See Register 25-13 and Register 25-14 for DEVID values. DEVID registers are read-only and cannot be programmed by the user.
3: Available only on PIC18F4455/4550 devices in 44-pin TQFP packages. Always leave this bit clear in all other devices.

Tabla 2.h Registros para la palabra de configuración

De las características que más interesa en el diseño es la configuración del oscilador. Este se configura colocando los bits FOSC3:FOSC0 del registro CONFIG1H como 111x (HSPLL_HS). Si el USB es habilitado, tanto el microcontrolador como el módulo de USB utilizan el mismo modo. Este modo indica que además se utilizará el divisor de frecuencia PLL.

Del mismo registro, se debe colocar el bit IESO=0 . IESO deshabilita el uso del oscilador interno del microcontrolador.

Según muestra la figura 2.5 de la hoja de datos del PIC, del registro CONFIG1L, los bits PLLDIV2:PLLDIV0=001 (2 en PLLDIV de keyboard.c) indican que se va a utilizar el PLL dividiendo entre 2 la frecuencia del oscilador primario debido a que el cristal es de 8MHz para obtener 4MHz que rigurosamente debe ser alimentados al circuito multiplicador a 96MHz.

El bit USBDIV=1 lógico (2 en keyboard.c) para indicar que la fuente de reloj para USB proviene del PLL de 96MHz dividido por 2 y no directamente del oscilador primario.

Los bits CPUDIV1:CPUDIV0=00 (CPUDIV) indican que si se selecciono el reloj primario HSPLL, la salida del 96MHz PLL es dividido entre 2 para derivarlo al reloj del sistema (Reloj del CPU).

Del registro CONFIG2L, el bit VREGEN=1 que habilita el regulador de voltaje de 3.3V interno de la unidad de USB.

Como el *"firmware"* del microcontrolador fue creado desde un compilador C para evitar tener que programarlo en ensamblador, la programación de los bits de configuración se hizo modificando

algunos parámetros del fragmento siguiente del archivo de la aplicación “Keyboard.c” (el código completo del archivo se muestra en el anexo1):

```
/** CONFIGURATION *************************************************/
 // Configuración de bits basado en PIC18F4550
      #pragma config PLLDIV   = 2                 // (8 MHz crystal para 4 Mhz entrada a PLL)
      #pragma config CPUDIV   = OSC1_PLL2         // Reloj CPU = 96MHz / 2 = 48 MHz (96MHz del PLL)
      #pragma config USBDIV   = 2             // Fuente de reloj desde el 96MHz PLL/2 para obtener 48MHz
      #pragma config FOSC     = HSPLL_HS      // Oscilador HS,PLL habilitado, HS usado por USB
      #pragma config FCMEN    = OFF
      #pragma config IESO     = OFF
      #pragma config PWRT     = OFF
      #pragma config BOR      = ON
      #pragma config BORV     = 3
      #pragma config VREGEN   = ON            //Regulador de voltaje de USB activado
      #pragma config WDT      = OFF
      #pragma config WDTPS    = 32768
      #pragma config MCLRE    = ON
      #pragma config LPT1OSC  = OFF
      #pragma config PBADEN   = OFF                 //Entradas RB0:RB4 digitales
//    #pragma config CCP2MX   = OFF
      #pragma config STVREN   = ON                  //Reinicio por sobreflujo
      #pragma config LVP      = OFF
```

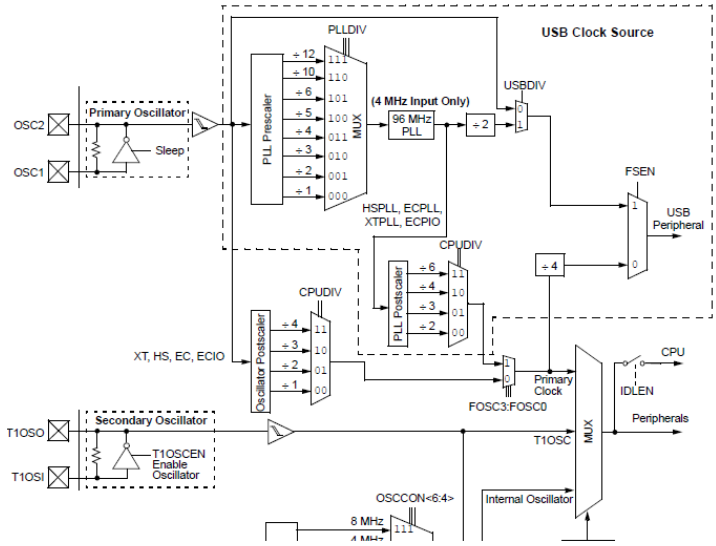


Figura 2.5 Diagrama de reloj PIC18F4550 (3)

2.7 Implementación de la comunicación USB.

La implementación USB (Universal Serial Bus) fue necesaria debido a que hoy en día, los equipos de cómputo disponen en su totalidad de comunicación USB y raramente disponen de un puerto de comunicación RS232 o paralelo. El objetivo principal del proyecto sería contar con un dispositivo que los usuarios fácilmente pudiera conectar a las estaciones de trabajo (PC) sin tener que exigir algún tipo de implementación o interface adicional con configuración especial.

Como se ha mencionado anteriormente, El PIC18F4550 cuenta con una unidad controladora USB cuya activación y configuración se puede lograr a través de la manipulación de varios registros de control adicionales a otros 22 registros que se usan para las transacciones, entre los que se mencionan:

- *USB Control register (UCON)*
- *USB Configuration register (UCFG)*
- *USB Transfer Status register (USTAT)*
- *USB Device Address register (UADDR)*
- *Frame Number register (UFRMH:UFRML)*
- *Endpoint Enable register 0 to 15 (UEPn)*

El diseño de circuitos con USB, su implementación y uso resulta más compleja a diferencia del empleo de otros sistemas de comunicación. Para lograr configurar la unidad, activarlo y responder a todo el conjunto de instrucciones de control que le envía el sistema al que se conecta el USB denominado "*host",* se tendría que escribir una gran cantidad de instrucciones y nemónicos para ello, lo que resultaba ser bastante complejo y añadiría mucho más tiempo al desarrollo. Para ahorrar esto, Microchip tiene ya desarrolladas funciones y rutinas necesarias en C y están disponibles en sus librerías para el desarrollo de proyectos que incluyan la unidad controladora de USB. En el proyecto están ubicadas en la carpeta denominada *"USB Stack"* y los archivos *"usb_config"*.

Sin embargo, a pesar de que se cuenta con dichas utilerías, es necesario entender el funcionamiento y requerimientos básicos necesarios de la comunicación USB para configurarlo y lograr que adquiera la funcionalidad que se desea, en este caso, del proyecto *"Nosy"*.

En lo que resta de este capítulo, se dará una breve descripción del funcionamiento del USB detallando paralelamente los puntos importantes y configuraciones que se debieron considerar para lograr la implementación del USB en el proyecto.

USB es un protocolo de comunicación definido por el USB Implementers Forum, Inc. (USB-IF) y su contenido e información puede ser consultado en www.usb.org.

Al igual que con el protocolo RS232, la comunicación USB es también asíncrona y utiliza básicamente 3 velocidades de bus: Alta en 480MBits/seg, Completa en 12 Mbits/seg y Baja en 1.5Mbits/seg. El USB 1.0 solo trabaja con velocidades Baja y Completa. El USB 2.0 trabaja con las 3.

Los dispositivos USB suelen ser de bajo consumo de energía y pudieran no requerir alimentación externa, ya que la interface de USB suministra voltajes nominales de +5 Volts y hasta 500 mA de corriente. Si el periférico requiere un consumo de corriente arriba de estas especificaciones el diseñador debe proporcionársela de manera externa. El tipo de alimentación ya sea proporcionada por el bus o de manera externa debe quedar definido en el *"firmware"* del periférico, específicamente dentro de los descriptores del dispositivo que se explicarán más adelante.

Para el usuario final de dispositivos USB (memorias, discos duros, etc.) la instalación resulta relativamente fácil, ya que este tipo de dispositivos no necesitan configuración especial, tal como ocurre con la conexión de dispositivos que trabajan con RS232 donde es necesario conocer los parámetros de comunicación y establecerlos por ejemplo en la configuración de dispositivos dentro del panel de control de Windows.

El USB es universal y es empleado para una gran cantidad de dispositivos. La velocidad de transmisión de datos es inclusive superior a la de una comunicación RS232 y llega alcanzar velocidades de hasta 480Mbits/seg comparado con los hasta 20Kbits/seg del puerto RS232. Cuando se conecta el dispositivo USB por primera vez, Windows detecta el periférico y mediante un proceso denominado enumeración, aprende de las capacidades y uso del dispositivo para después, ya sea que solicite el controlador de dispositivo o asigne él mismo uno procedente de su librería de controladores de dispositivo almacenando sus datos en la carpeta INF (*INF files: Device Setup information file*) y en el registro de datos de Windows. Una vez terminada la tarea de enumeración de manera satisfactoria, el dispositivo está listo para usarse. Para subsecuentes conexiones del dispositivo a Windows, este lo reconocerá al obtener la información almacenada en el registro de Windows desde la primera vez lo que le permitirá brincarse ciertas etapas de configuración para su uso inmediato.

El USB también cuenta con ciertas desventajas: Las comunicaciones son de punto a punto, es decir; *"host"* a dispositivo o viceversa. No es posible establecer comunicaciones directamente entre dispositivos. Como ya se ha mencionado, desde el punto de vista desarrolladores, también la complejidad de la programación y la necesidad de contar con un ID de Proveedor *(*Vendor ID*)* o identificador de proveedor para la comercialización de los productos. El dispositivo debe contar con un ID de Proveedor y un ID de Producto (Product ID) o identificador de producto integrado en cada dispositivo para que el mismo pueda ser reconocido por el sistema operativo durante la etapa de enumeración. EL ID de Proveedor debe ser asignado y proporcionado por el USB-IF mediante el pago de derechos para la comercialización de dispositivos USB. Cabe aclarar, que el dispositivo *"Nosy"* no se realizó con la finalidad de comercializarlo, sino dar una solución a un proceso interno. Para poder implementarlo, se utilizó el ID de Proveedor del fabricante del Microcontrolador (Microchip) proporcionado por ellos mismos en sus literaturas de desarrollo el cual es x4D8. El ID de Producto es asignado por un servidor.

El *"host"* es el encargado del bus y este debe conocer que dispositivos y con que capacidades se conectan a él. En el caso de una computadora personal las tareas del *"host"* son asimiladas por Microsoft Windows y las ejecuta con funciones del sistema operativo. Las aplicaciones en realidad no necesitan conocer los detalles de cómo se comunican los dispositivos con el *"host"*. Lo único que tienen que hacer es enviar y recibir datos usando funciones estándar del sistema operativo.

Para establecer la comunicación, el *"host"* debe ejecutar las siguientes tareas:

- Detectar los dispositivos, a través del proceso denominado enumeración. El *"host"* asigna una dirección y solicita información adicional una vez que el dispositivo es conectado al

puerto. Si un dispositivo es removido o nuevamente conectado, el *"host"* enumera el nuevo dispositivo conectado y remueve el dispositivo desconectado de su lista de dispositivos disponibles.
- Controla el flujo de datos.
- Maneja y checa los errores de transmisión de datos.
- Proporciona energía al dispositivo (hasta 500ma).
- Intercambia datos con el periférico.

De forma recíproca, el dispositivo tiene las siguientes tareas:

- Cuando el *"host"* inicia comunicaciones, el dispositivo debe responder de manera recíproca a los requerimientos de este a través del controlador USB. El dispositivo no puede iniciar las comunicaciones por sí mismo.
- Cada dispositivo monitorea y detecta la dirección contenida en las comunicaciones del bus y no toma ninguna acción hasta que el chip que corresponde a dicha dirección ha detectado la comunicación.
- Como ya se ha mencionado, el dispositivo debe responder a todos los requerimientos del *"host"* durante la enumeración, pero aún después de terminado este proceso también debe ser capaz de responder, ya sea para consultas sobre sus capacidades, estado o para tomar alguna otra acción.
- Maneja también el chequeo de errores durante la comunicación.
- Cuando cesan las comunicaciones entre el bus, el *"host"* entra en un estado de bajo consumo de energía. El dispositivo debe detectarlo y entrar en un modo que lo limite a un bajo consumo de corriente y pueda salir del mismo cuando el bus sea activado.
- Después de haber sido configurado el dispositivo mediante la enumeración, comienza el intercambio de datos entre el *"host"* y el dispositivo.

El protocolo USB permite manejar 4 tipos básicos de transferencias de datos cuyo uso, depende de la función del dispositivo y sus necesidades, de la velocidad de transferencias requerida, de los tiempos de respuesta y del manejo y corrección de errores durante las transferencias. Estos tipos son:

- Control. Este tipo de transferencia tiene dos usos, el primero es que todo dispositivo USB debe tener implementado y soportar trasferencias de control ya que se utiliza para la configuración del dispositivo durante el proceso de enumeración y para los requerimientos que hace el host al dispositivo aún después de finalizado el proceso. Para ello, el protocolo establece una serie de comandos de control al que el dispositivo debe ser capaz de responder.
- Por volumen. Las transferencias por volumen (Bulk) están diseñadas para situaciones donde la velocidad de transferencia es importante pero no crítica. La transferencia de datos puede esperar si es necesario y se puede utilizar en tareas tales como el envío de un archivo a una impresora, envío de datos de un escáner a una computadora, acceso de archivos a un disco duro, etc.

- Asíncrono. Para este tipo de transferencias, el envío de datos se realiza en tiempo real (Streaming) y sin detenerse. La velocidad de transmisión es importante pero no existe el manejo de corrección de errores, es decir; no existe retransmisión de datos si estos se reciben con errores, por lo que los errores ocasionales son aceptables. Este tipo solo acepta velocidades Completa y Alta. Se utiliza principalmente para transferencias de audio, video, etc.
- Interrupción. Es un tipo de transferencia donde se requiere atención o envío de datos de manera espontánea. Para el manejo de velocidades Baja es la única forma de transferir datos. Ejemplos de dispositivos que usan trasferencias de interrupción son los teclados, ratones, etc. Para poder realizar estas transferencias, el protocolo de USB maneja una serie de instrucciones de interrupción que los dispositivos deben soportar.

Para fines de esta aplicación y diseño, se utilizó la transferencia de datos por interrupción ya que se está simulando un teclado numérico. Este tipo de transferencia queda configurado en el descriptor de Endpoint.

Nota: Un punto muy importante a mencionar es que en este tipo de transferencias, una transferencia de interrupción termina de una de las dos siguientes formas: cuando la cantidad de datos esperada ha sido transferida o cuando una transacción contiene **0 bytes de datos** u otro número de bytes que es menor al tamaño máximo de paquete en el Endpoint. Esta situación se ve reflejada en la aplicación principal del *"firmware"*, ya que fue necesario implementar el envío de transferencias con 0 bytes de datos para terminar la transferencia. Una vez que se enviaba algún dato, por ejemplo un carácter numérico, la transmisión se envolvía en un ciclo infinito transmitiendo el mismo carácter sin detenerse. El envío de una transferencia de 0 bytes que seguía al carácter transmitido, detenía la transmisión.

2.7.1 El Endpoint.

La transferencia de datos del dispositivo al *"host"* proviene de un Endpoint. Este es un *"buffer"* que almacena múltiples bytes. Físicamente es un bloque de memoria o registros dentro del chip controlador. Los datos almacenados en el Endpoint pueden ser datos recibidos o datos transmitidos aunque la comunicación se realiza en una sola dirección a la vez. El *"host"* no tiene Endpoints aunque también tiene *"buffer"* para los datos recibidos o transmitidos.

Se puede tener varios Endpoints los cuales deben tener un número y dirección asignada. La dirección es asignada desde el punto de vista del *"host"*, así, tenemos "Endpoint IN" refiriéndose a la transferencia de datos del dispositivo al *"host"* y "Endpoint OUT" refiriéndose a la transferencia de datos del *"host"* al dispositivo.

Todo dispositivo debe tener al menos un Endpoint 0 configurado como Endpoint de control utilizado durante la enumeración y una vez concluida esta, todas las transacciones en el bus inician con un paquete que contiene el número de Endpoint y un código que indica el flujo de datos.

2.7.2 La Enumeración.

Enumeración, figura retórica que consiste en la acumulación de palabras o lista de elementos lingüísticos de un conjunto y adoptada por USB-IF para describir un proceso o lista de tareas que se inician al conectar un dispositivo USB a un *"host"* y antes de que las aplicaciones puedan comunicarse con los dispositivos, el *"host"* lo ejecuta con la finalidad de aprender de las funciones, características y habilidades del dispositivo para configurarlo y asignar un controlador de dispositivo para que éste pueda ser utilizado, entre algunas otras cosas más. Hasta que la enumeración ha sido ejecutada de manera satisfactoria, las aplicaciones pueden utilizar el dispositivo.

Aunque el proceso de enumeración es aun más complejo y el orden en que se ejecuta no necesariamente es siempre el mismo, básicamente se describe la secuencia típica que el sistema operativo ejecuta al conectar un dispositivo USB a la interface enviando una serie de transferencias de control al Endpoint 0 del dispositivo:

- El usuario conecta un dispositivo al puerto.
- El *"host"* se da cuenta de la presencia del dispositivo al detectar variaciones de voltaje a través de sus resistores *"pull-down"* en las líneas D+ D- del puerto.
- El *"host"* detecta la velocidad del dispositivo, si se trata de baja, completa o alta mediante el envío de señales especiales y de reinicio.
- El *"host"* establece comunicación con el Endpoint 0 dirigiéndose a la dirección por defecto 00h.
- El *"host"* comienza a enviar transferencias de control al Endpoint 0 para obtener el máximo tamaño del paquete de datos en la nueva asociación (pipa) que se da entre el Endpoint 0 y el *"host"*.
- El *"host"* asigna una dirección lógica al dispositivo mediante la instrucción "Set_Address". Esta dirección será valido hasta que el dispositivo sea desconectado o reinicializado. Cada vez que se conecta de nuevo el dispositivo o se reinicializa el sistema, el proceso de enumeración comienza de nuevo, asignando una nueva dirección para el dispositivo.
- El *"host"* aprende de las habilidades del dispositivo enviando "Get_Descritor". Este es el punto donde el *"host"* lee el contenido de los descriptores que contienen información básica del dispositivo. Los descriptores serán tratados más a detalle en el apartado siguiente.
- El *"host"* asigna y carga el controlador de dispositivo (driver). Una vez que el *"host"* ha aprendido acerca del dispositivo a través de los descriptores, busca el mejor controlador de dispositivo que maneje las comunicaciones entre el *"host"* y el dispositivo. El *"host"* intenta conciliar la información contenida en la carpeta INF de Windows con los datos obtenidos de los descriptores como el ID de Proveedor y el ID de Producto, si esto no se logra, trata de conciliar los datos de clase, subclase y protocolo contenidos en los descriptores. En el caso de nuestro diseño, se estableció en el descriptor de interface la clase HID para facilitarle a Windows la búsqueda del controlador de dispositivo (genérico incluido en Windows). Si el sistema no encuentra un controlador para el dispositivo, suele

solicitar la asistencia del usuario para seleccionar el controlador y especificar donde debería encontrarlo. Después de asignar el controlador, este queda almacenado en el registro del sistema (registros de Windows) para que cuando el dispositivo vuelva a ser numerado nuevamente, Windows no tenga que buscarlo de nuevo en la carpeta INF.

- El controlador de dispositivo del host pudiera requerir alguna configuración adicional al dispositivo enviando "Set_Configuration".
- El dispositivo está listo para usarse.

Después de que la enumeración ha sido completada, Windows agrega el nuevo dispositivo al Administrador de dispositivos del panel de control donde es posible visualizarlo abriendo dicha aplicación. Cuando el dispositivo es retirado del *"host"*, el dispositivo desaparece del administrador de dispositivos aunque, su configuración queda guardada en el registro de Windows.

2.7.3 Los descriptores.

Los descriptores son estructuras de datos que le permiten al *"host"* conocer acerca del dispositivo. Durante el proceso de enumeración, el *"host"* (Windows) utiliza transferencias de control para obtener los descriptores del dispositivo respondiendo este último a los requerimientos del *"host"*.

Conforme la enumeración progresa, uno a uno va obteniendo los datos de los descriptores en orden de importancia. Se puede tener algunos descriptores que son subordinados de otros y cuya existencia dependen de la configuración que presenten los anteriores.

En el diseño del proyecto, los descriptores se encuentran en el archivo fuente denominado "usb_descriptors.c".

2.7.4 Descriptor de dispositivo.

El primer descriptor solicitado por el *"host"* mediante la instrucción "Get_Descriptor request" es el denominado descriptor de dispositivo, el cual contiene información básica acerca del mismo.

Este descriptor contiene 14 campos y dentro del archivo "usb_descriptors.c" se muestra de la siguiente manera:

```
/** INCLUDES *******************************************************/

#include "./USB/usb.h"
#include "./USB/usb_function_hid.h"
/** CONSTANTS ******************************************************/
#if defined(__18CXX)
#pragma romdata
#endif
```

```
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
```

	Explicación:
0x12,	Indica la longitud en bytes (Hex) del descriptor.
USB_DESCRIPTOR_DEVICE,	Es una constante que indica el tipo de descriptor. El valor numérico de la constante es 01h.
0x0200,	Indica la versión del dispositivo USB. En este caso corresponde a USB 2.0
0x00,	Indica valor del código de clase del dispositivo. Es cero debido a que la clase es definida en el descriptor de interface.
0x00,	Indica el código de subclase.
0x00,	Indica el código de protocolo.
USB_EP0_BUFF_SIZE,	Es una constante que indica el máximo tamaño del paquete en el Endpoint 0. Este es de 8 bytes indicado en el "usb_config.h".
MY_VID,	Constante que indica el ID de Proveedor. Se le dio como valor el 0x4D8 en el archivo "usb_config.h" que pertenece a Microchip.
MY_PID,	Constante que indica el ID de Producto. En el archivo "usb_config.h" se le asignó el valor 0x0001 por un servidor.
0x0001,	Número de realización del dispositivo en formato BCD. El fabricante asigna este valor.
0x01,	Índice que apunta a una cadena que describe al fabricante. Debe ser cero si no hay descriptor de fabricante. Ver 2.7.11.
0x02,	Índice que apunta a una cadena de texto que describe el producto. Este valor debe ser cero si no hay descriptor de cadena. Ver 2.7.11.
0x00,	Índice que apunta a una cadena que contiene el número de serie del producto. Es cero ya que no existe.
0x01	Número de posibles configuraciones.

```
};
```

2.7.5 Descriptor de configuración.

Después de que el *"host"* ha obtenido el descriptor de dispositivo, obtiene a continuación la configuración del dispositivo, la interface y el descriptor de Endpoint.

El descriptor de configuración contiene una descripción de las características y habilidades del dispositivo. Los dispositivos por lo menos tienen una configuración y de tener varias, debe también tener tantos descriptores de configuración como sean necesarios, pero solo una puede estar activa a la vez.

Cada descriptor de configuración contiene subordinada a ellos uno o más descriptores de interface y descriptores de Endpoint.

El *"host"* obtiene su información contenida a través del envío de un "Get_Descriptor request".

Consta de 8 campos y en el diseño del proyecto el código correspondiente se encuentra dentro del mismo archivo "usb_descriptors.c". Dentro de este archivo, tanto el descriptor de interface, el

descriptor de clase HID y el descriptor de Endpoint, se encuentran subordinados al descriptor de configuración y deben incluirse dentro de los mismos corchetes del descriptor de configuración.

/* Configuration 1 Descriptor */ ROM BYTE configDescriptor1[]={	
/* Configuration Descriptor */	**Explicación:**
0x09,//sizeof	Longitud en bytes del descriptor de configuración (hex).
USB_DESCRIPTOR_CONFIGURATION,	Constante qué indica el tipo de descriptor y cuyo valor corresponde a 02h.
DESC_CONFIG_WORD(0x0029),	Número de bytes en el descriptor de configuración y todos sus descriptores subordinados.
1,	Número de interfaces en la configuración.
1,	Identificador para los requerimientos de control correspondientes a “Set_Configuration” y “Get_Configuration”.
0,	Índice a una cadena del descriptor de configuración (no hay en este caso).
0xA0,	Atributo para indicar si el suministro de poder es proporcionado por el bus y las configuraciones necesarias para activar la reactivación remota. Si el bit 6=1, el dispositivo es alimentado por si mismo (fuente de poder para el dispositivo), si es 0, es alimentado por el bus. Si el bit 5=1, el dispositivo soporta reactivación remota. El valor 0xA0 indica un 10100000 binario. Si consideramos una numeración de 0 a 7 de derecha a izquierda, vemos que el bit 6 es 0 por lo que el dispositivo es alimentado por el bus. El bit 5 es igual 1 por lo que soporta la reactivación remota y el bit 7 es utilizado por USB 1.0 que indica alimentado por el bus.
50,	Máximo consumo de energía que el dispositivo requerirá expresado en (2X mA). Se observó que el dispositivo no consume más de 100 mA por lo que el valor en el descriptor corresponde a 50. El bus USB solo puede suministrar hasta 500mA, arriba de esto se requeriría alimentación externa.
Continúa descriptor de interface ↓	

2.7.6 Descriptor de Interface.

Este descriptor de interface es subordinado al descriptor de configuración.

Un dispositivo puede tener y realizar múltiples funciones como por ejemplo, funciones de Video cámara, teclado, etc. y cada función puede a su vez tener varias tareas como controlar la video cámara, transportar la señal de video, etc. El término interface, dentro del contexto de los descriptores del dispositivo, se refiere a las características o funciones que el dispositivo implementa o lo que hace. Cuando se implementan varias interfaces o varias funciones, se debe también crear tantos descriptores e interface se requieran, cada uno con su correspondiente descriptor de Endpoint, ya que estos están subordinados al descriptor de interface.

Para implementar el uso del descriptor de asociación de interfaces, es necesario indicarle al *"host"* que use dicho descriptor. Para ello, es necesario modificar algunos parámetros de los campos del descriptor del dispositivo como el bdeviceClass=EFh, el bDeviceSubClass=02h y el bDeviceProtocol=01h. Entonces, el *"host"* obtiene un descriptor de asociación de interface como consecuencia de los requerimientos en el descriptor de configuración para la configuración a la que la asociación de interface pertenece.

Como este proyecto solo fue diseñado para una sola interface, el procedimiento anterior no fue necesario y se prescindió del descriptor de asociación de interface. En vez de eso, solo se utilizó un descriptor de interface dentro del descriptor de configuración, aunque una configuración podría tener múltiples interfaces asociadas con una sola función o varios aunque no estuvieran relacionadas.

El descriptor de interface proporciona información acerca de las características o funciones que el dispositivo implementa y del número de Endpoints que utiliza, por lo que debe contener por lo menos un descriptor de Endpoint subordinado a este descriptor.

El descriptor de interface es requerido por el *"host"* como consecuencia de los requerimientos en el descriptor de configuración, según la configuración a la cual pertenezca.

El único descriptor de interface contenido en el proyecto es el que se muestra a continuación y se escribe a partir de la siguiente línea del descriptor de configuración descrito anteriormente:

/* Interface Descriptor */	**Explicación:**
0x09,	Número de bytes en el descriptor.
USB_DESCRIPTOR_INTERFACE,	Constante que indica el tipo de descriptor. Su valor es 04h.
0,	Número que identifica esta interface. Por defecto es cero.
0,	Para usos en múltiples y exclusivas interfaces. Por defecto es cero.
2,	Número de Endpoints que la interface soporta adicionales al Endpoint 0.
HID_INTF,	Constante que indica la clase especificada para la interface. Su valor es 03h que lo describe como "Dispositivo de Interface Humana" y es definido en el archivo "usb_function_hid.h".
0,	Cuando se indica en el campo anterior que se trata de un HID, este campo denominado "subclase" determina si el dispositivo va a soportar una interface de arranque (*booteable)*. Es decir, que se pueda usar un protocolo que permita utilizar el dispositivo antes de que sea cargado su correspondiente controlador de dispositivo. Ejemplo, cuando se ingrese al DOS o al BIOS. 0 indica que no acepta el uso de sistemas de arranque. 1 Lo contrario.
0,	Si el campo denominado "subclase" es 0, entonces este campo llamado bInterfaceProtocol debe ser cero. Si la subclase fuera 1, los valores que puede tomar son 1 Teclado y 2 para el ratón.
0,	Índice que apunta a una cadena de texto de interface.

Continúa: /* HID Class-Specific Descriptor */ ↓

2.7.7 Dispositivos de Interfaz Humana (HID).

La clase HID o dispositivos de interfaz humana fue una de las clases de dispositivo que se integraron a Windows a partir de la versión 98 y contiene una serie de controladores (driver) que permiten las comunicaciones con dispositivos tales como teclados, ratones, palancas de mando, lectores de código de barras, etc. Su nombre tiene sentido ya que fue diseñado para dispositivos que tuvieran interacción directa con el humano como la simple pulsación de un botón, etc. Sin embargo, debido a que el HID define una serie de reglas y comandos, es posible aprovechar y utilizar los mismos para desarrollar algún dispositivo que permita comunicarlo con el *"host"*, pues no existe la necesidad de desarrollar un controlador especial para comunicar el dispositivo ya que Windows lo tiene incorporado, pero es necesario cumplir con los párrafos que se describen a continuación:

Si se define una clase HID desde el descriptor de interface, es necesario agregar también su descriptor de HID.

Todo el intercambio de información se efectúa a través de estructuras de datos conocidas como "reportes". Un HID puede soportar uno o más reportes.

Si se define una interface HID debe de tener al menos un Endpoint IN de interrupción definido en su correspondiente descriptor para el envío de reportes desde el dispositivo al *"host"*.

Se puede definir de manera opcional un Endpoint OUT de interrupción definido en su correspondiente descriptor para el envío de reportes del *"host"* al dispositivo.

La utilización de un Endpoint IN de interrupción sugiere que todo HID debe tener al menos un *"INPUT Report"* definido en su correspondiente descriptor de reporte HID.

El uso de la clase HID solo permite el uso de transferencias de datos mediante los tipos "Control" o "Interrupción". El de interrupción es el que se utiliza en esta aplicación.

2.7.8 Descriptor HID.

/* HID Class-Specific Descriptor */	**Explicación:**
0x09,	Tamaño del descriptor en bytes
DSC_HID,	Constante que indica el tipo de descriptor. Su valor es 21h.
DESC_CONFIG_WORD(0x0111),	Constante que indica el número de realización de las especificaciones HID. Su valor es 0x0111 que corresponde a la versión 1.11
0x00,	Código de país para el dispositivo. Como no aplica su valor es cero.
HID_NUM_OF_DSC,	Constante definida en el archivo "usbcfg.h" que indica el numero de descriptores de reporte o físicos subordinados. Su valor es 1.
DSC_RPT,	Constante que indica el tipo de descriptor subordinado. Su valor es 22h que indica que es un descriptor de reporte.
DESC_CONFIG_WORD(63),	Función que establece el tamaño del descriptor de reporte en bytes. Su valor es de 63 bytes.

2.7.9 Descriptor de Endpoint.

Debido a que desde el descriptor de interface se establecen dos Endpoints adicionales al Endpoint 0. También debido a que se definió la clase HID para el dispositivo, se agregan dos descriptores de Endpoint, uno para el correspondiente *"Endpoint IN"* y otro para el *"Endpoint OUT"*.

/* Endpoint Descriptor */	*"Endpoint IN"*
0x07,	Tamaño del descriptor en bytes
USB_DESCRIPTOR_ENDPOINT,	Constante que indica el tipo de descriptor. Su valor es 05h para el Endpoint.
HID_EP | _EP_IN,	Constante que indica el número y dirección (IN) del Endpoint. HID_EP=1, _EP_IN=0x80
_INTERRUPT,	Constante que indica el tipo de transferencia de datos utilizado. Su valor es 0x03.
DESC_CONFIG_WORD(8),	Función que establece el máximo tamaño de paquete soportado (8 bytes).
0x01,	Indica el máximo intervalo de consultas al dispositivo (polling).
/* Endpoint Descriptor */	*"Endpoint OUT"*
0x07,	Tamaño del descriptor en bytes
USB_DESCRIPTOR_ENDPOINT,	Constante que indica el tipo de descriptor. Su valor es 05h para el Endpoint.
HID_EP | _EP_OUT,	Constante que indica el número y dirección (OUT) del Endpoint. HID_EP=1, _EP_OUT=0x00
_INTERRUPT,	Constante que indica el tipo de transferencia de datos utilizado. Su valor es 0x03.
DESC_CONFIG_WORD(8),	Función que establece el máximo tamaño de paquete soportado (8 bytes).
0x01	Indica el máximo intervalo de consultas al dispositivo (polling).
};	**FIN DEL CONTENIDO DESCRIPTOR DE CONFIGURACION**

2.7.10 Descriptor de reporte HID.

El descriptor de reporte especifica el tamaño y contenido de datos en el reporte del dispositivo y también informa de cómo esos datos pueden ser utilizados por el receptor de los mismos.

El descriptor de reporte es diferente a los descritos anteriormente, ya que no consta de una serie fija de parámetros establecidos con valores que describen el comportamiento del dispositivo. Su tamaño y valores puede ser variable y es construido a través de una serie campos denominados "ítems", que según su tipo y valor van modificando los valores subsecuentes del reporte o reportes. De esta manera y para saber como el descriptor determina la forma o sintaxis del reporte se debe entender lo que se describe en los siguientes párrafos.

Cada línea del descriptor se denomina Ítem que es una simple pieza de información dentro de un descriptor de reporte. A nivel de bits, se tienen dos tipos de ítems: los ítems cortos y los ítems largos. Lo más utilizados comúnmente son los ítems cortos que pueden tener un tamaño de hasta

5 bytes. El ítem corto tiene un byte como prefijo cuyo campo de bits se compone de tres partes: bTarea, bTipo y bTamaño. El bTarea es un campo de 4 bits que indica la función o tarea del ítem. El bTipo es un campo de 2 bits que indica si se trata de un ítem Principal, Global o Local. El campo bTamaño se compone de dos bits que indican el número de bytes de datos en el ítem adicionales al prefijo. Pueden ser hasta 4 bytes.

-Ítems Principales. Estos agrupan los siguientes Ítems:

."*Input"*: Con prefijo "1000 00 nn" que se refiere a los datos que va a enviar el dispositivo al *"host"*.
."*Output"*: Con prefijo "1001 00 nn" que se refiere a los datos que va a entregar el *"host"* al dispositivo a través del reporte.
.Características (Features): Con prefijo "1011 00 nn" que describe características propias del dispositivo.
.Colección (Collection): Con prefijo "1010 00 nn". Cuando se coloca un ítem de colección, significa agrupamiento de ítems *"Input", "Output"* y de Características.
.Fin de Colección (End Collection): Con prefijo "1100 00 nn" este indica fin de la colección descrita anterior.

-Ítems Globales. Estos definen algunos parámetros y características del reporte. Cuando se escribe algún ítem perteneciente al tipo Global, los subsiguientes ítem principales asumirán las características establecidas por el ítem Global que los precede hasta que se defina o escriba un nuevo ítem Global.

Es importante definir el concepto de Uso (Usage) introducido por el HID_USB Comite. Los "Usos" son parte de un descriptor de reporte que suministran información relacionada para una función específica del HID, un control o un grupo de controles. Existen unas tablas descritas por el USB Comite y publicadas en su página de internet que describen de manera detallada todos los "usos" o función asignados para aplicaciones HID, así también contiene la lista de los Índices de Uso (Usage Id) en formato hexadecimal para los controles o sub funciones pertenecientes a cada "Uso". El "Uso" es un valor de 32 bits sin signo donde el valor alto de 16 bits corresponde a la "Página de Uso"(ítem global) y el valor bajo de 16 bits corresponde al *" Id Uso"* (Índice de *Uso* descrito en los ítems locales como "Uso o Usage").

Entre los principales ítem calificados como Globales se mencionan:

.Página de Uso (Usage Page): Con prefijo "0000 01 nn" Son utilizados en el descriptor para conservar el espacio colocando el valor en la parte alta de los 16 bits en los subsecuentes "Usos". Cualquier "Uso" (ítem local) que le siga es definido como los 16 bits de la parte baja e interpretados como *" Id Uso"* siendo concatenado con *"*Página de Uso*"* para formar los 32 bits.
.Mínimo Lógico (Logical Minimum): Con prefijo "0001 01 nn" establece el valor mínimo que puede tomar el arreglo (array) o variable (dato) en el reporte. Si tiene el calificador de variable, el valor que puede tomar depende del número de bits del campo del reporte. Si es un arreglo el valor que toma es un índice al control.
.Máximo Lógico (Logical Maximum): Con prefijo "0010 01 nn" establece el valor máximo que puede tomar el arreglo o variable (dato) en el reporte. Si tiene el calificador de variable, el valor que puede tomar depende del número de bits del campo del reporte. Si es un arreglo el valor que toma es un índice al control.

. Tamaño de Campo de Reporte (Report Size): Con prefijo “0111 01 nn” es un entero que especifica el número de bits que comprende un campo de datos dentro del reporte.
. Número de Campos de Reporte (Report Count): Con prefijo “1001 01 nn”. Entero sin signo que indica el número de campos de dato dentro del reporte del tamaño determinado por “Tamaño de Campo de Reporte”.

-Ítem Locales. Estos ítem también definen características de los controles, pero a diferencia de los ítem globales, las características solo se trasladan al siguiente ítem principal al que precede, pero no a los siguientes. Se tendría que escribir un ítem local para cada ítem principal que se quiera configurar. Cuando se tiene definido varios controles para varios “Usos”, se tendría que utilizar ítem locales para cada control. Para asignar un único ítem local para cada control en un simple ítem principal, se puede especificar cada “Uso” secuencialmente mediante el empleo de los ítems “Mínimo Uso” y “Máximo Uso”.

Los ítems locales más utilizados se describen a continuación:

.*Uso (Usage):* Con prefijo “0000 10 nn” es un índice de uso (Id Uso) especificado en las tablas publicadas por el USB-HID Comite. Representa un “Uso” sugerido para el ítem o la colección. En el caso donde el ítem represente múltiples controles, un “Uso” pudiera sugerir una funcionalidad para toda variable o elemento en un arreglo.
.Mínimo Uso (Usage Minimum): Con prefijo “0001 10 nn”. Define el inicio del “Uso” asociado con el arreglo o variable.
.Máximo Uso (Usage Maximum): Con prefijo “0010 10 nn”. Define el fin del “Uso” asociado con el arreglo o variable.

-Los validadores de datos. Dentro del conjunto de bits que componen a los ítems cortos y contiguos al primer byte (prefijo compuesto por el bTarea, bTipo y bTamaño), en los campos asignados para datos, los bits 8 al 16 del ítem son frecuentemente utilizados como bits validadores de datos. Entre los más utilizados se encuentran:

.Dato (0) | Constante (1): Ubicado en el bit 8. Este bit establece si el ítem es dato o valor constante. “Dato” indica que el ítem está definiendo campos con datos del dispositivo modificables. Constante indica que el ítem es un campo estático de solo lectura y no puede ser modificado por el *“host”*.
. Arreglo (0) | Variable (1): Ubicado en el bit 9, establece si el ítem crea campos de dato variables o estructuras de datos (array) en el reporte. Si es variable, cada campo representa datos de un control físico. Los bits reservados para cada campo queda determinado por los ítem Tamaño de campo de Reporte / Número de Campo de Reporte que preceden. Un arreglo es una forma alternativa de describir los datos retornados de un grupo de controles. En vez de retornar bits para cada botón del grupo, un arreglo retorna un índice en cada campo que corresponda con el botón presionado. Los ítems Mínimo Lógico y Máximo Lógico especifican el rango de valores índice retornados por el arreglo.
. Absoluto (0) | Relativo (1): Ubicado en el bit 10, establece si el dato es absoluto o un valor preestablecido o relativo si el dato cambia de valor desde el último reporte.

En vista de lo explicado anteriormente, se muestra a continuación el descriptor de reporte para el dispositivo *“Nosy”* contenido en el archivo “usb_descriptors.c” del proyecto.

```
//Class specific descriptor - HID Keyboard
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
{  0x05, 0x01,   // USAGE_PAGE (Generic Desktop)
   0x09, 0x06,   // USAGE (Keyboard)

   0xa1, 0x01,   // COLLECTION (Application)
   0x05, 0x07,   //   USAGE_PAGE (Keyboard)

   0x19, 0xe0,  // USAGE_MINIMUM (Keyboard LeftControl)
   0x29, 0xe7,  // USAGE_MAXIMUM (Keyboard Right GUI)
   0x15, 0x00,  //   LOGICAL_MINIMUM (0)
   0x25, 0x01,  //   LOGICAL_MAXIMUM (1)
   0x75, 0x01,  //   REPORT_SIZE (1)
   0x95, 0x08,  //   REPORT_COUNT (8)
   0x81, 0x02,  //   INPUT (Data,Var,Abs)
   0x95, 0x01,  //   REPORT_COUNT (1)
   0x75, 0x08,  //   REPORT_SIZE (8)
   0x81, 0x03,  //   INPUT (Cnst,Var,Abs)
   0x95, 0x05,  //   REPORT_COUNT (5)
   0x75, 0x01,  //   REPORT_SIZE (1)
   0x05, 0x08,  //   USAGE_PAGE (LEDs)
   0x19, 0x01,  //   USAGE_MINIMUM (Num Lock)
   0x29, 0x05,  //   USAGE_MAXIMUM (Kana)
   0x91, 0x02,  //   OUTPUT (Data,Var,Abs)
   0x95, 0x01,  //   REPORT_COUNT (1)
   0x75, 0x03,  //   REPORT_SIZE (3)
   0x91, 0x03,  //   OUTPUT (Cnst,Var,Abs)
   0x95, 0x06,  //   REPORT_COUNT (6)
   0x75, 0x08,  //   REPORT_SIZE (8)
   0x15, 0x00,  //   LOGICAL_MINIMUM (0)
   0x25, 0x65,  //   LOGICAL_MAXIMUM (101)
   0x05, 0x07,  //   USAGE_PAGE (Keyboard)
   0x19, 0x00, //USAGE_MINIMUM (Reserved)
   0x29, 0x65,  // USAGE_MAXIMUM (Keyboard Application)
   0x81, 0x00,  //   INPUT (Data,Ary,Abs)

   0xc0}          // End Collection
};
```

Explicación:

Establece la función del dispositivo HID en lo general como una Computadora genérica (0x01) y en lo particular la función de un Teclado (0x06).

Se abre una colección del tipo aplicación. Como el Teclado se compone de más elementos como teclas, *"leds"*, etc, se establece el Teclado (0x07) como control principal dentro de la colección (ítem global).

Crea 8 campos de 1 bit dentro del *"Input Report"* específicos para los controles teclado leftControl a Right GUI cuyo rango de Id Uso va desde 0xE0 a 0xE7. Sus valores solo pueden tomar el 0 o el 1 ya que son variables.

Crea 1 campo adicional para el reporte Input de 8 bits. Su valor es constante.

Crea 5 campos de tamaño de 1 bit para el "*Output Report"* cuyo control principal se refiere a la señalización de Leds. Abarca valores de Id Uso desde 0x01 (num Lock) hasta 0x05 (kana).

Crea 1 campo adicional de 3 bits cada uno para el *"Output Report"* cuyo valor es constante.

Para el *"Input Report"*, crea 6 campos adicionales de 8 bits cada uno para el control Teclado. Los valores lógicos (Id Uso) que pueden tomar sus controles son los índices que varían de 0x00 a 0x65 y que corresponden a las 101 teclas que comprende el teclado. Estos campos del *"Input Report"* son establecidos como un arreglo.

Fin de la colección.

Cabe mencionar que la sintaxis mostrada anteriormente es una estructura de datos donde se visualiza mostrando dos campos hexadecimales. El primer número hexadecimal indica el prefijo del ítem, el siguiente valor hexadecimal contiene el dato o valor que le sigue al prefijo del ítem y en el caso de los INPUT y OUTPUT ese dato se refiere a los validadores de datos. Ejemplo, la primera línea de la estructura del descriptor de reporte esta escrita como:

0x05, 0x01, //USAGE_PAGE… : 0x05 es un valor hexadecimal que indica el ítem. En binario corresponde a 0000 01 01. Este indica que se tiene un campo de datos y la función corresponde a al ítem global llamado Página de Uso.

El valor hexadecimal 0x01 corresponde al campo de dato y representa el valor "Uso" en hexadecimal que según las tablas del HID_USB Comite corresponde a la descripción de una Computadora Genérica (Generic Desktop).

La segunda línea por ejemplo, 0x09, 0x06, // USAGE…… : 0x09 es un valor hexadecimal que corresponde con la sintaxis del ítem en binario "0000 10 01" que de igual manera, indica que se trata de un ítem local denominado *"Uso"* y contiene un byte de datos.

Ese byte de datos es representado por el valor 0x06 que corresponde según las tablas publicadas por el HID_USB Comite al "ID Uso" de un Teclado.

Debido a lo anterior, durante la trasferencia de datos del dispositivo al *"host"*, se envía un *"Input Report"* cuyo tamaño es de 8 bytes descrito en el campo DESC_CONFIG_WORD(8) del descriptor de Endpoint utilizando una trama de datos similar a la figura 2.6. El *"Output Report"* no es utilizado en la aplicación final y no se muestra en este documento.

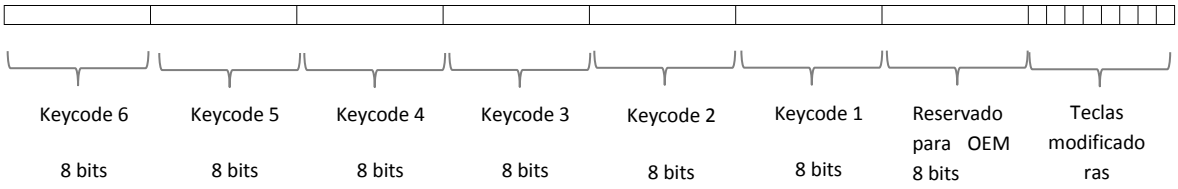


Figura 2.6 Input Report resultante del descriptor de reporte

Esta es una trama reconocida y utilizada por los teclados comerciales conectados vía USB a una computadora personal. Se cuenta con un bloque de 8 campos de 1 bit cada uno asignado para las teclas modificadoras, un campo de 8 bits reservado para OEM y 6 campos de 8 bits para los códigos del teclado (del 0 al 101 o en hexadecimal, del 0x00 al 0x65).

En virtud de la trama mostrada anteriormente, es posible enviar en un solo *"Input Report"* datos que representen la pulsación combinada de varias teclas simultáneamente.

Las teclas modificadoras son una combinación de bits dentro del campo asignado para ellos y toman su valor de acuerdo a la tabla 2.i. Para ello, basta con poner un 1 en el bit correspondiente para representar la tecla pulsada y 0 para las no pulsadas. Se puede tener varias pulsadas a la vez.

Bit	Tecla modificadora
0	Left Ctrl
1	Left Shift
2	Left Alt
3	Left GUI
4	Right Ctrl
5	Right Shift
6	Right Alt
7	Right GUI

Tabla 2.i Campo de Bits de teclas modificadoras

Como el diseño requería el envío al *"host"* de la simulación de varias teclas pulsadas de forma combinada para después, la aplicación las reconociera y las reinterpretara ejecutando código VBA

necesario para completar finalmente el objetivo de registrar los eventos telefónicos en la base de datos, con la utilización de los campos o bits asignados para las teclas modificadoras en el reporte como lo muestra la tabla 2.j, podemos contar con dos combinaciones de teclas modificadoras requeridas: “CTRL + ALT” o simplemente “CTRL”.

Combinación teclas modificadoras	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**Valor hexadecimal**
Ctrl + Alt	0	0	0	0	0	1	0	1	0x05
Ctrl	0	0	0	0	0	0	0	1	0x01

Tabla 2.j Bits necesarios para combinación de teclas modificadoras en proyecto

En la trama generada por el descriptor de reporte (figura 2.6) se hace notar que aunque se cuenta con 6 campos de 8 bits cada uno para colocar en cada campo valores relativos al valor índice de cada tecla (Id Uso), solo se requirió la utilización de uno de esos campos. Para Windows es indiferente que campo sea utilizado ya que finalmente es su tarea interpretarlo. En el diseño (véase código C del anexo 1), la variable *“hid_report_in[0]”* del “buffer” representa el campo de las teclas modificadoras y toma el valor de la variable “KEY1”. La variable *“hid_report_in[2]”* (campo keycode 1 en figura 2.6) representa uno de los 6 campos de 8 bits asignados para los índices de teclas (0 a 101) y toma el valor de las variables “KEY2” que contiene el valor índice hexadecimal de las teclas numéricas o de “KEY3” que contiene el valor índice hexadecimal de las letras “F”, “E”, “J” o “D” según fuera el evento dentro del *“Nosy”*. Tanto *“hid_report_in[0]”* como *“hid_report_in[2]”* transmitidos dentro del mismo *“Input Report”* simulan teclas oprimidas simultáneamente. Todo el conjunto de datos o comandos transmitidos por el dispositivo USB denominado *“Nosy”* visto desde el contexto *“Input Report”* es mostrado en la tabla 2.k.

Campo teclas modificadoras (hid_report_in[0])	**Campo Keycode 1 (hid_report_in[2])**	**Acción**
0x05	0x09	CTRL + ALT + F
0X01	0X08	CTRL + E
0X01	0X0D	CTRL + J
0X01	0X07	CTRL + D
0X00	0X1E	Tecla del #1
0X00	0x1F	Tecla del #2
0X00	0X20	Tecla del #3
0X00	0X21	Tecla del #4
0X00	0X22	Tecla del #5
0X00	0X23	Tecla del #6
0X00	0X24	Tecla del #7
0X00	0X25	Tecla del #8
0X00	0X26	Tecla del #9
0X00	0X27	Tecla del #0
0X00	0X04	Tecla de a
0X00	0X05	Tecla de b

Tabla 2.k Lista de códigos hexadecimales necesarios para generar los comandos emitidos por Nosy

2.7.11 Descriptores de cadena de texto.

Ubicados en el archivo "usb_descriptors.c", estos descriptores de cadena de texto son consecuencia del descriptor de dispositivo que menciona en sus campos para primero, apuntar a un índice (1) de una cadena de texto que describe al fabricante:

```
//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'M','i','c','r','o','c','h','i','p',' ',
'T','e','c','h','n','o','l','o','g','y',' ','l','n','c','.'
}};
```

Y segundo, apuntar a un índice (2) de una cadena de texto que describe al producto:

```
//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[22];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'B','e','s','t',' ','B','u','y',' ','N','o','s','y','.'
}};
```

Durante el proceso de enumeración, en la primera ocasión que se instala el dispositivo USB, esta cadena de texto es visualizada en el menú inicio tal como se muestra en la figura 2.7 siendo un indicativo de la ejecución del mismo proceso y es el nombre con el que se ubica el *"Nosy"* en el administrador de dispositivos del panel de control de Windows.

Figura 2.7 Resultado de cadenas de texto en descriptores

CAPITULO 3

DESARROLLO E IMPLEMENTACIÓN DEL PROYECTO

3.1 Descripción.

Una vez definida la configuración y características necesarias para la operación del microcontrolador incluido su controlador de USB, es necesaria la construcción de la aplicación principal (función main en C) que representa el corazón del dispositivo *"Nosy"*. Este código se encuentra dentro del archivo "Keyboard.c" y ejecuta principalmente las tareas de iniciar el microcontrolador y comenzar mediante llamadas al *"USB stack"* el proceso de enumeración entre el *"host"* y el dispositivo USB. Una vez terminada la enumeración y establecida la comunicación con el *"host",* el programa debe monitorear las entradas al microcontrolador procedentes del comparador (detector de colgado/descolgado) y del decodificador DTMF para reinterpretar la información de eventos lo que le lleva finalmente a enviar los comandos requeridos para la interacción con la base de datos Access que registrara dichos eventos.

Construida y depurada la aplicación, es necesario compilarla junto con el resto de archivos fuente y el *"USB Stack"* para convertirlo en un archivo con extensión "hex" (*.hex) utilizando la herramienta MPLAB de Microchip que se describirá más adelante. Cuando ya se cuenta con el archivo *.hex, este es grabado en el microcontrolador con dispositivos dedicados a ello. Para este proyecto se utilizó el PCKit2 de Microchip.

Armado en principio en una placa de pruebas (protoboard) y habiendo verificado la funcionalidad del dispositivo electrónico utilizando inicialmente Microsoft Word para la lectura de los comandos recibidos mediante macros, se procedería finalmente a pasar el dispositivo a placa de circuito impreso creando el prototipo número 2 el cual serviría para las pruebas finales en campo.

Se finalizó la implementación del proyecto adecuando e insertando el código VBA y creando los formularios necesarios en la base de datos Access para que fuera capaz de interactuar con el dispositivo.

El anexo 2 muestra evidencias fotográficas de las distintas fases del diseño del dispositivo *"Nosy"*.

3.2 El flujo lógico del firmware del Microcontrolador.

Como se explicó en párrafos anteriores, el archivo fuente "Keyboard.c" es el que contiene el programa principal, es decir; el *"main()"* además del conjunto de instrucciones y funciones que va a correr en primera instancia el microcontrolador una vez que se le suministre energía eléctrica, lo que sucede realmente cuando el dispositivo *"Nosy"* es conectado al puerto USB de la terminal o

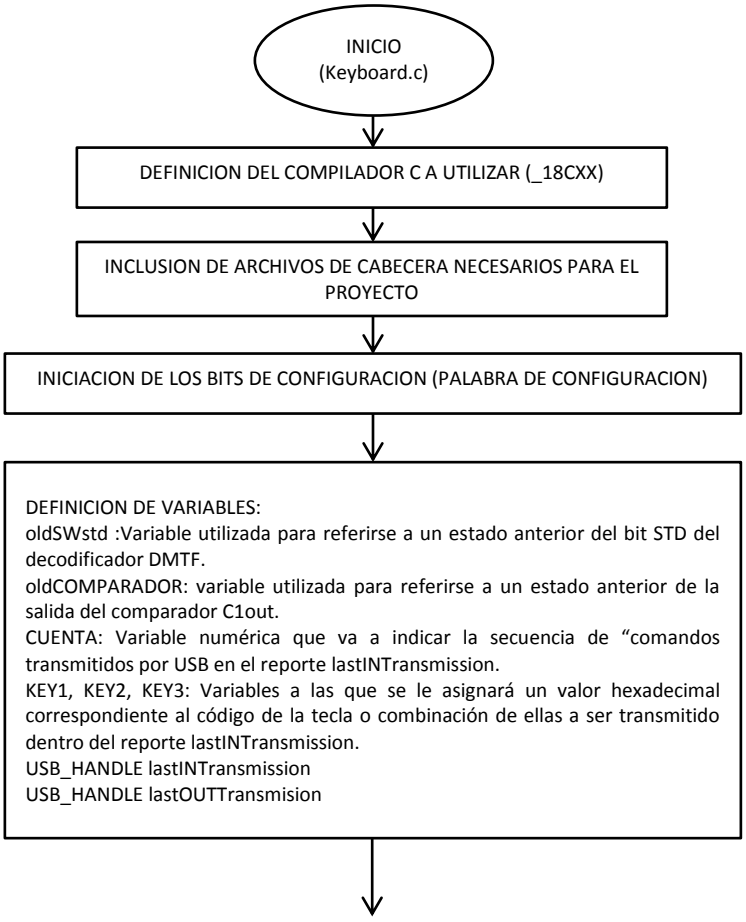
computadora. El resto de archivos contenidos en el proyecto proporcionan las herramientas, definiciones, configuraciones y subrutinas requeridas por el programa principal para su operación y funcionamiento deseado que serán llamadas durante el proceso de compilación y creación del archivo *.hex. La descripción del resto de los archivos fuente y herramientas son tratados en el apartado 3.3 de este capítulo.

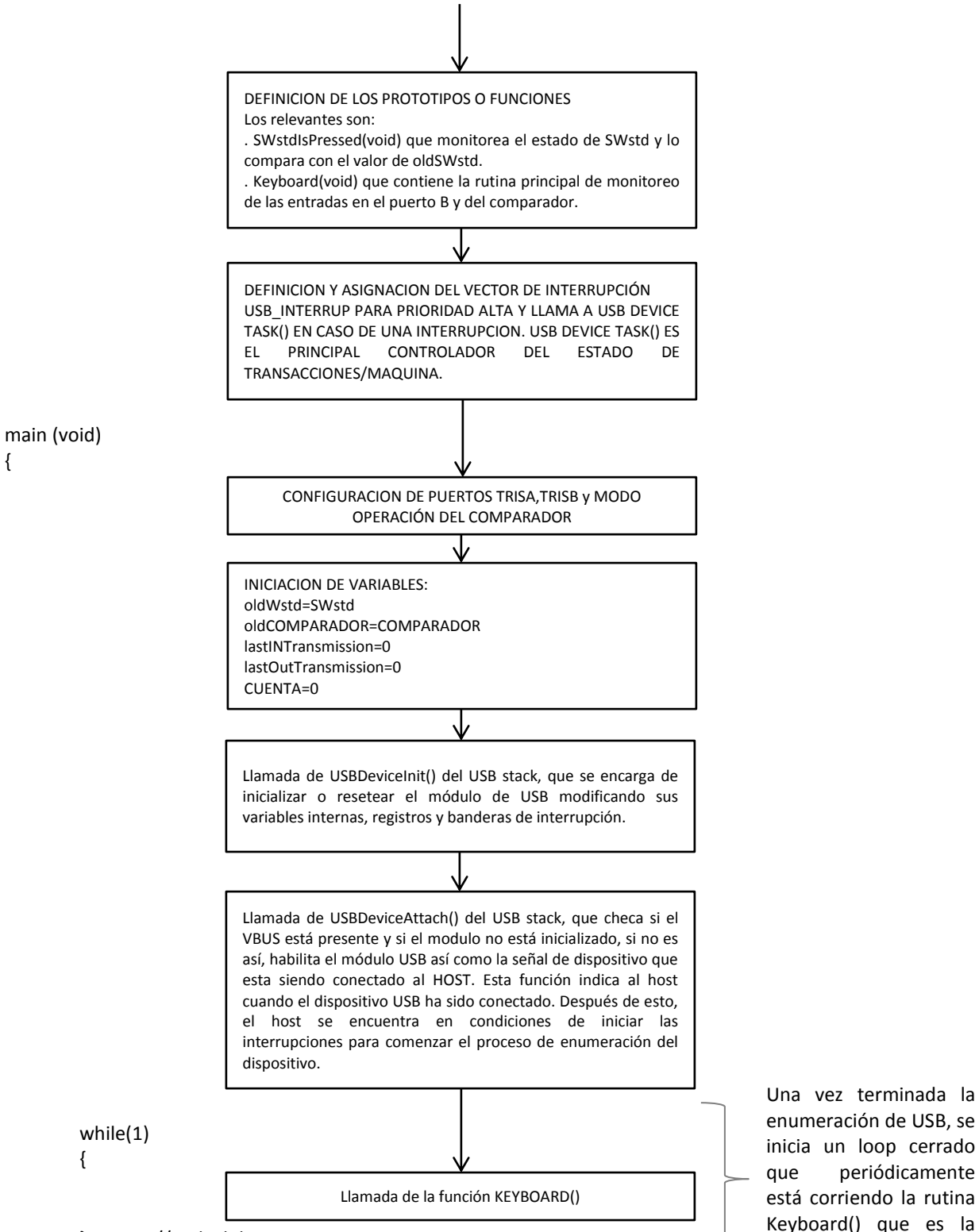
La logística del programa principal descrito en el "Keyboard.c" se representa en el flujo lógico de operación que se muestra a continuación. El código completo y escrito en lenguaje C se muestra en el Anexo 1.

Se hace notar para el buen entendimiento de este flujo, que se tienen variables ya definidas en los archivo de cabecera "hardwareProfile.c" e invocadas en el "keyboard.c":

cmConf0 //bit 0 CMCON
cmConf1 //bit 1 CMCON
cmConf2 //bit 2 CMCON
cmInvC1 //bit 4 CMCON
SWstd //valor en RB4 de Puerto B (señal proveniente de la terminal STD del decodificador).
COMPARADOR //Valor de la salida del comparador C1out

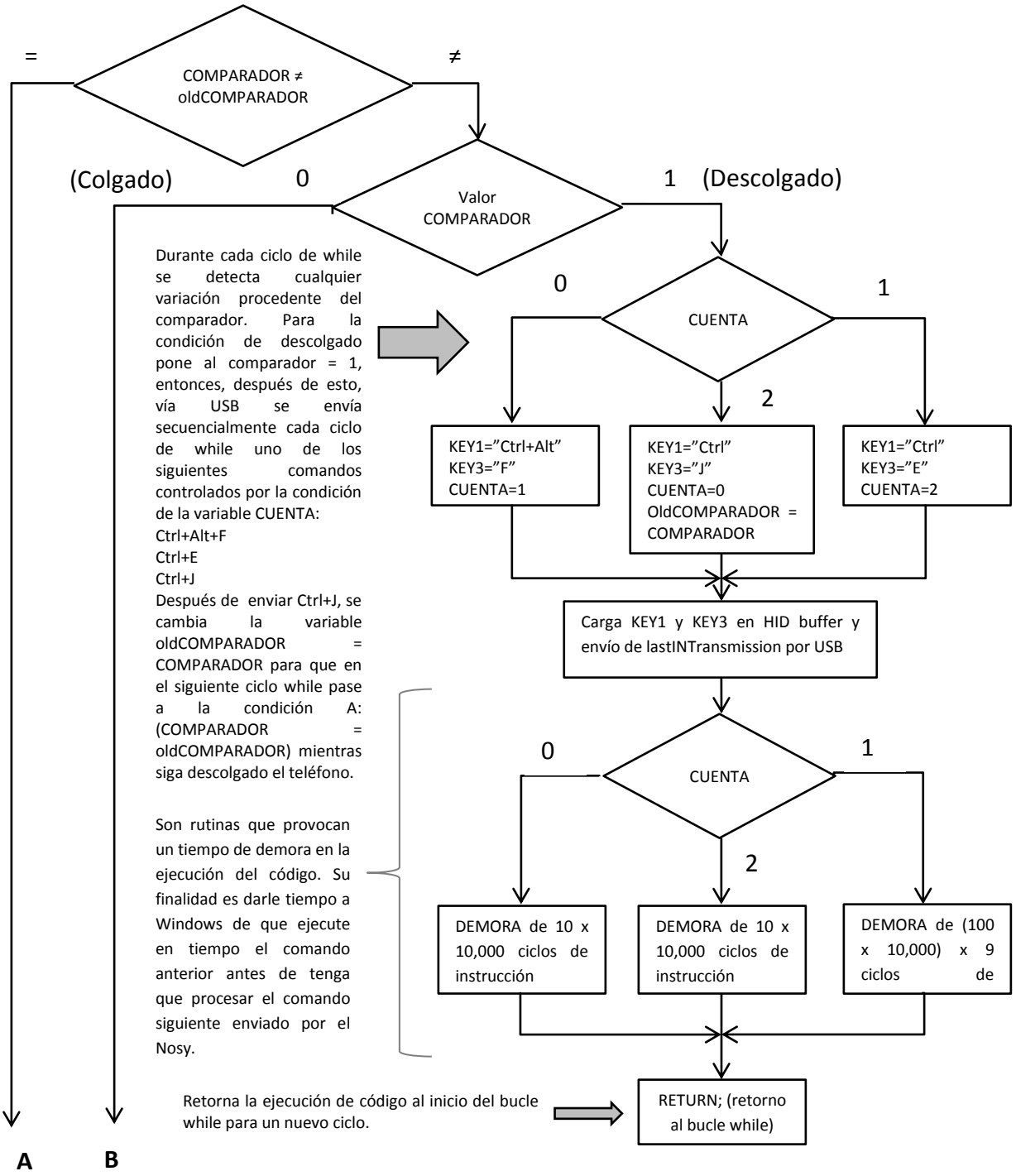
El flujo es el siguiente:

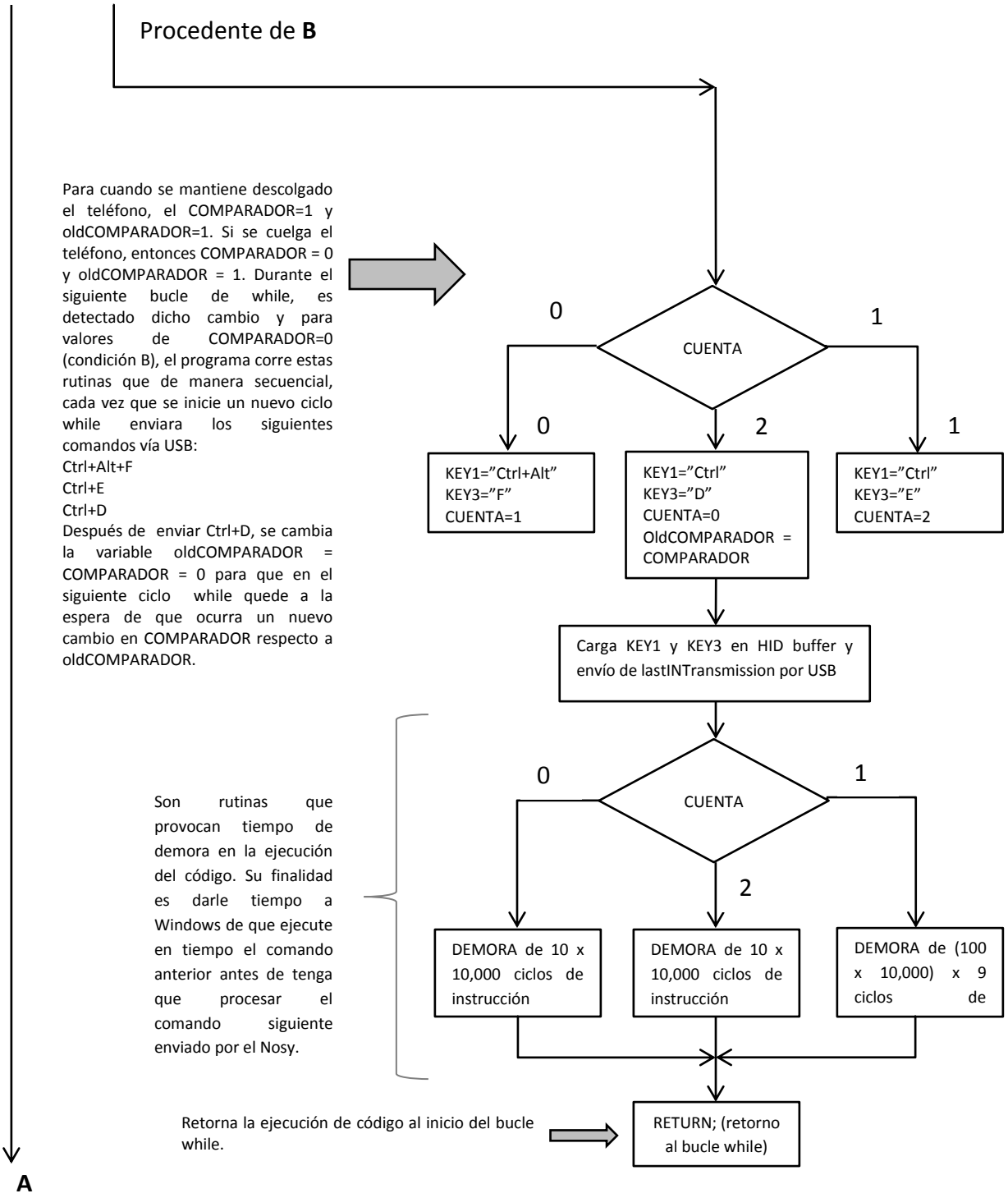


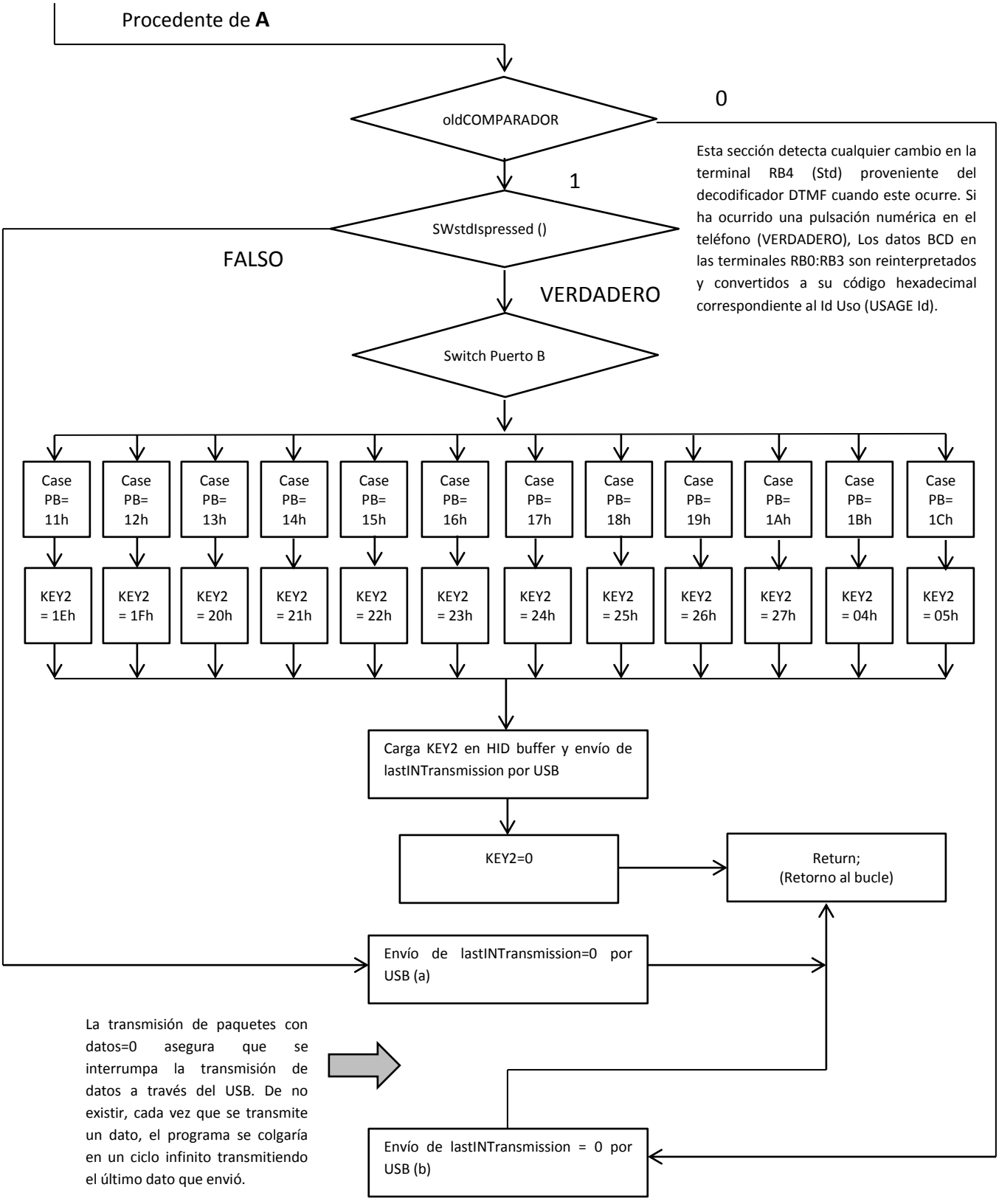

DEFINICION DE LOS PROTOTIPOS O FUNCIONES
Los relevantes son:
. SWstdIsPressed(void) que monitorea el estado de SWstd y lo compara con el valor de oldSWstd.
. Keyboard(void) que contiene la rutina principal de monitoreo de las entradas en el puerto B y del comparador.
DEFINICION Y ASIGNACION DEL VECTOR DE INTERRUPCIÓN USB_INTERRUP PARA PRIORIDAD ALTA Y LLAMA A USB DEVICE TASK() EN CASO DE UNA INTERRUPCION. USB DEVICE TASK() ES EL PRINCIPAL CONTROLADOR DEL ESTADO DE TRANSACCIONES/MAQUINA.
main (void)
{
CONFIGURACION DE PUERTOS TRISA,TRISB y MODO OPERACIÓN DEL COMPARADOR
INICIACION DE VARIABLES:
oldWstd=SWstd
oldCOMPARADOR=COMPARADOR
lastINTransmission=0
lastOutTransmission=0
CUENTA=0
Llamada de USBDeviceInit() del USB stack, que se encarga de inicializar o resetear el módulo de USB modificando sus variables internas, registros y banderas de interrupción.
Llamada de USBDeviceAttach() del USB stack, que checa si el VBUS está presente y si el modulo no está inicializado, si no es así, habilita el módulo USB así como la señal de dispositivo que esta siendo conectado al HOST. Esta función indica al host cuando el dispositivo USB ha sido conectado. Después de esto, el host se encuentra en condiciones de iniciar las interrupciones para comenzar el proceso de enumeración del dispositivo.
while(1)
{
Llamada de la función KEYBOARD()
} //end while
} //end main
Una vez terminada la enumeración de USB, se inicia un loop cerrado que periódicamente está corriendo la rutina Keyboard() que es la encargada de monitorear los eventos de colgado, descolgado y marcación numérica para transmitirlos al USB.

El flujo de la función Keyboard() se describe a continuación:

```
void Keyboard(void)
{
   if(!HIDTxHandleBusy(lastINTransmission)) //Checa si el Input endpoint no está ocupado y si no lo está,
   {                                        //checa si queremos enviar datos de eventos al host.
```




Procedente de **B**
Para cuando se mantiene descolgado el teléfono, el COMPARADOR=1 y oldCOMPARADOR=1. Si se cuelga el teléfono, entonces COMPARADOR = 0 y oldCOMPARADOR = 1. Durante el siguiente bucle de while, es detectado dicho cambio y para valores de COMPARADOR=0 (condición B), el programa corre estas rutinas que de manera secuencial, cada vez que se inicie un nuevo ciclo while enviara los siguientes comandos vía USB:
Ctrl+Alt+F
Ctrl+E
Ctrl+D
Después de enviar Ctrl+D, se cambia la variable oldCOMPARADOR = COMPARADOR = 0 para que en el siguiente ciclo while quede a la espera de que ocurra un nuevo cambio en COMPARADOR respecto a oldCOMPARADOR.
CUENTA
0
1
0
2
1
KEY1="Ctrl+Alt"
KEY3="F"
CUENTA=1
KEY1="Ctrl"
KEY3="D"
CUENTA=0
OldCOMPARADOR = COMPARADOR
KEY1="Ctrl"
KEY3="E"
CUENTA=2
Carga KEY1 y KEY3 en HID buffer y envío de lastINTransmission por USB
Son rutinas que provocan tiempo de demora en la ejecución del código. Su finalidad es darle tiempo a Windows de que ejecute en tiempo el comando anterior antes de tenga que procesar el comando siguiente enviado por el Nosy.
CUENTA
0
1
2
DEMORA de 10 x 10,000 ciclos de instrucción
DEMORA de 10 x 10,000 ciclos de instrucción
DEMORA de (100 x 10,000) x 9 ciclos de
Retorna la ejecución de código al inicio del bucle while.
RETURN; (retorno al bucle while)
A




```
   }
return;
}//end keyboard()
```

3.3 El entorno de desarrollo del firmware.

El fabricante del microcontrolador PIC18F4550 Microchip Inc., ha puesto de forma gratuita en su página de internet a disposición de los diseñadores de circuitos con microcontroladores PIC una gran cantidad de utilerías, literatura y herramientas para facilitar el desarrollo de proyectos. Entre la literatura podemos encontrar la hoja de datos de los microcontroladores que incluye también el repertorio de instrucciones, la guía de usuario del compilador C18 que incluye la sintaxis necesaria para programar el microcontrolador en lenguaje C, el manual del MPLAB, la librería para los módulos de entrenamiento, etc.

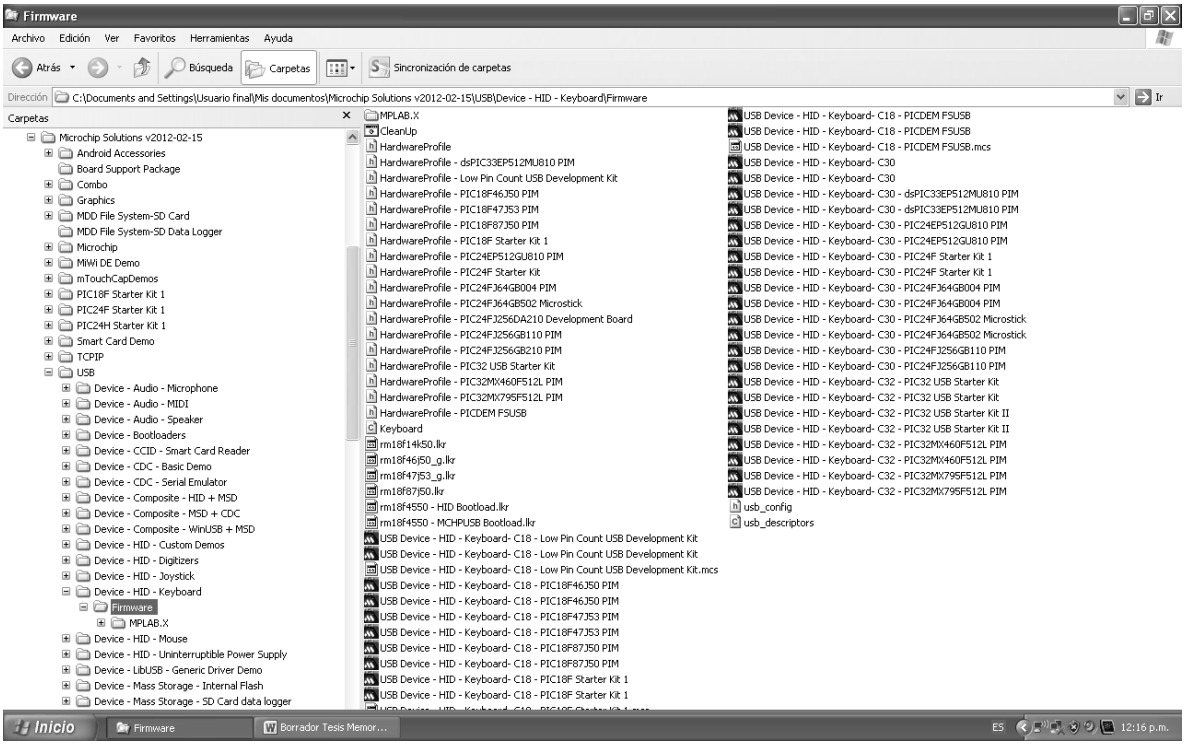


Figura 3.1 Vista de la carpeta Microchips solutions v2012-02-15

Microchip incluye también el software conocido como MPLAB, que es una aplicación cuya principal característica es proporcionar un espacio de trabajo que nos permite generar primero el código, ya sea en ensamblador nativo (nemónicos para los PIC) o en lenguaje C con la incorporación de compiladores dedicados para ello (a partir de la gama 18F en adelante).

Además de las herramientas mencionadas anteriormente, Microchip también proporciona una gran cantidad de ejemplos de proyectos y es ubicada como una carpeta llamada *"Microchip solutions v2012-02-15"* como se muestra en la figura 3.1 (la versión puede estar más actualizada, aquí se muestra la última que se descargó) que incluyen sus librerías y códigos fuente que pueden ser inspeccionados y tomados como base para el desarrollo de aplicaciones en MPLAB.

Es uno de estos ejemplos (USB Device - HID - Keyboard- C18 - Low Pin Count USB Development Kit) que se tomó como plataforma para el desarrollo del *"firmware"* del *"Nosy"* modificándolo y adaptándolo a las necesidades específicas del proyecto.

La estructura del proyecto visto en el MPLAB se muestra en la figura 3.2

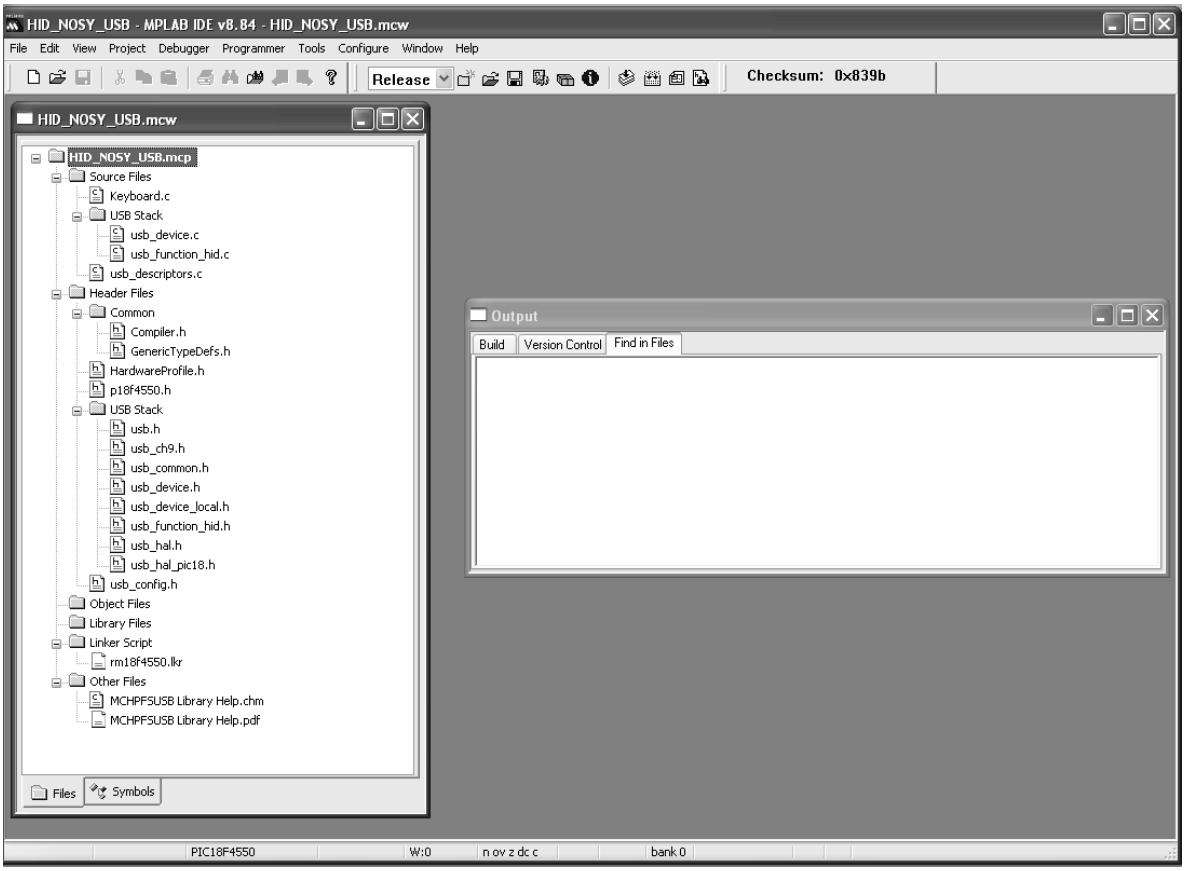


Figura 3.2 Vista de aplicación MPLAB de Microchip

En esta aplicación se muestran básicamente dos ventanas: La ventana del proyecto y la consola de salida.

En la ventana del proyecto (nombrado HID_NOSY_USB.mcw) se muestra una estructura de árbol, todas las carpetas de archivos relacionadas con el mismo proyecto, que incluye tanto los archivos de programa como los de cabecera (tal como se estructura un programa en lenguaje C). Se hace notar que esta ventana de proyecto es un tanto parecida a la de los compiladores comerciales de C tales como Visual Estudio de Microsoft por ejemplo. Al hacer doble pulsación sobre cada archivo podemos ver el código contenido en cada uno de ellos.

La siguiente ventana, que es la consola de control (output) es una ventana que indica el estado que guarda el proyecto, ya sea que se esté depurando o que se compile para finalmente intentar generar el archivo *.hex necesario para programar el microcontrolador (el firmware). Esta ventana indica el estatus de la tarea ya sea de compilación o depuración mostrando si ocurre algún error

debido a algún archivo faltante o algún código mal escrito durante el proceso de escritura del mismo. El objeto de este texto no es el explicar a detalle como funciona y como se usa el MPLAB, sino el de describir como está finalmente armado el proyecto para la creación del *"firmware"* (archivo *.hex) del microcontrolador.

Partiendo de la figura 3.2, todo el conjunto de archivos juegan un papel en el momento de la compilación y la creación del archivo *.hex. El proyecto tiene varios archivos fuente (hablando en términos del lenguaje C). El archivo denominado "Keyboard.c" ya citado en el punto 3.2 es el que contiene el *"main()"* o subrutina principal (programa principal) que dará funcionalidad al microcontrolador y es uno de los varios archivos que fue necesario modificar sustancialmente y adaptarlo a las necesidades del proyecto.

El archivo "usb_device.c" contiene iniciación de variables y subrutinas desarrolladas por Microchip que ahorran la necesidad de tener que crearlas desde cero y que controlan gran parte de la compleja operación de la unidad USB. Es necesario aclarar que para establecer la comunicación vía USB, antes de que el dispositivo comience a intercambiar datos con las aplicaciones, es necesaria la iniciación y configuración de los dispositivos mediante el intercambio de cierta cantidad e datos entre el *"host"* y el dispositivo debiendo ser capaz este ultimo de responder a una serie de comandos y requerimientos de acuerdo al protocolo de comunicación USB. Sin embargo, para que esta unidad opere tal como un diseñador lo requiere, es indispensable tener conocimiento tanto de la teoría del funcionamiento del USB, como también de las estructuras de datos y configuraciones que se requieren implementar en el *"firmware"* para la correcta comunicación con el *"host"* y las aplicaciones.

Del mismo modo, el archivo "usb_function_hid.c" contiene la iniciación de variables y subrutinas necesarias para la transmisión de reportes HID a través de la interface USB descritos en el punto 2.7.10.

El archivo "usb_descriptors.c" contiene las estructuras de datos correspondientes a los descriptores de USB requeridos por el host (en este caso Windows) requeridos durante el proceso de enumeración del dispositivo después de ser conectado al puerto de USB tratados en los puntos 2.7.3 al 2.7.10. Este archivo es otro de los que requieren ser modificados y revisados minuciosamente ya que afectan sustancialmente la operación de la unidad USB.

Generalmente los archivos de cabecera (con extensión *.h) suelen ser librerías que contienen la definición de variables y constantes utilizados por el *"main()"* durante la compilación del programa.

La carpeta *"USB Stack"* contiene entre sus archivos variables y definiciones desarrolladas por Microchip como soporte para la unidad USB.

Entre los archivos de cabecera que son de consideración o que tuvieron que ser modificados para el *"Nosy"* se encuentran:

Archivo “HardwareProfile.h”. Aunque es un archivo desarrollado originalmente por Microchip para una aplicación demo, se utilizó y fue modificado aprovechando que es invocado desde el archivo fuente principal. Este archivo contiene algunas definiciones e iniciaciones necesarias tales como el tipo de alimentación del dispositivo USB y de las variables relacionadas con el comparador de voltaje y la puerta RB4 vinculadas con el monitoreo del cambio de estatus dentro del programa principal.

Archivo “p18f4550.h”. Este archivo es muy importante, ya que contiene “todas” las definiciones de variables, estructuras y constantes relacionadas con cada registro SFR que pertenecen específicamente al PIC18F4550. La invocación de este archivo permite que cuando se escriba código en lenguaje C en cualquier parte del proyecto, nos podamos referir a cada SFR como si se tratase de una variable o estructura como cualquier otra que se usa dentro de este lenguaje de programación. A final de cuentas, durante la compilación, con la inclusión de este archivo de cabecera, es el ensamblador quién sabrá cuando se esta refiriendo al cambio de ciertos parámetros dentro del SFR y a que banco se está dirigiendo, por ejemplo dentro del *“main()”*:

```
void main(void)
{
  //******INICIALIZACION DE SISTEMA***********
TRISB=0x1F;    //Port B as Input
TRISA=0x0F;
.
```

Finalmente, dentro del archivo “usb_config.h” existe una variedad de iniciación de variables relacionadas con USB. Dentro de estas se encuentran la USB_EPO_BUFF_SIZE (valor 8), USB_MAX_NUM_INIT (1), USB_MAX_EP_NUMBER (1), el tipo de interrupción utilizado por el USB (USB_INTERRUPT), el VID y el PID, el tamaño y localización de los Endpoints, el tamaño del reporte HID. Este archivo de igual manera requirió de una inspección minuciosa y de cambios en sus parámetros de acuerdo a las necesidades de la aplicación.

3.4 Generalidades de Microsoft Access.

Access de Microsoft es una herramienta de trabajo sencilla y relativamente económica que integra bases de datos, creación de consultas (a través de su motor de SQL) y la incorporación de código de programación VBA (Visual Basic Access) que permite crear y programar una cantidad inimaginable de aplicaciones tanto para uso comercial e incluso científicas y tecnológicas. Dentro de su entorno de programación VBA es posible también incorporar funciones API (Application programming Interface) de Windows, lo que abre la posibilidad de extender la operación controlando el sistema de una computadora e incluso los puertos de comunicaciones para la operación de dispositivos externos. Con Access es posible también trabajar dentro de una red de datos con varios usuarios a la vez.

Access es muy basta y se pueden generar aplicaciones combinadas con código VBA y API´s que permiten controlar por ejemplo inventarios, incluso crear un verdadero punto de venta,

generación y registro de códigos de barras para búsqueda de materiales, facturación, cobranza, etc. En el ámbito científico, es posible por ejemplo interconectarlo con dispositivos externos para el registro de algún muestreo de datos para su almacenamiento y análisis, incluso generar gráficas de resultados.

Access también tiene sus desventajas, si bien, se puede generar todo un sistema de operación y a pesar de que permite crear en su entorno cierto nivel de seguridad, para negocios este solo representaría un programa útil y económico para salir del paso. A un nivel estrictamente superior, no compite con sistemas y bases de datos más poderosas (por ende, muchísimo más costosas) como Oracle, SAP, etc. No permite tampoco una gran cantidad de usuarios en red. A medida que se concentran más usuarios (más de 6) se vuelve más lenta y es mucho más volátil, relativamente fácil de dañar, aunque existe realmente una técnica para compensar un poco esta lentitud y vulnerabilidad que es el proceso de sincronización de réplicas.

Básicamente se puede crear una base de datos a partir de varios objetos de trabajo y de manera secuencial:

Tablas. Son el objeto principal de toda base de datos. Con una estructura similar a las hojas de cálculo, en ella se pueden almacenar los registros que contienen a su vez una serie de campos relacionados al tema, por ejemplo, en la aplicación existe una tabla llamada General que contiene una orden de venta, estatus de la orden de venta, número de transacción en punto de venta, fecha de venta, número de cliente, nombre del cliente, teléfono, etc. Access trabaja bajo un contexto llamado “tablas relacionales” en la cual, podemos crear varias tablas y relacionarlas entre sí por un campo común a ellas, por ejemplo, se relaciona la tabla general ya citada con otra que contiene una lista de artículos vendidos por la tienda. Esta tabla de artículos vendidos puede contener otros campos como el número de artículo, su descripción, cantidad vendida y el número de orden de venta. Ambas tablas se relacionan a través del campo *“Sales Order”* (orden de venta) que es común entre ellas. Las tablas relacionadas de la aplicación se muestran en la figura 3.3. La línea que une a las tablas (*“sales Order”* de TBGeneral2 a *“Sales Order”* de TbGeneral) une el campo común a ambas tablas y puede ser una relación de uno a varios generalmente.

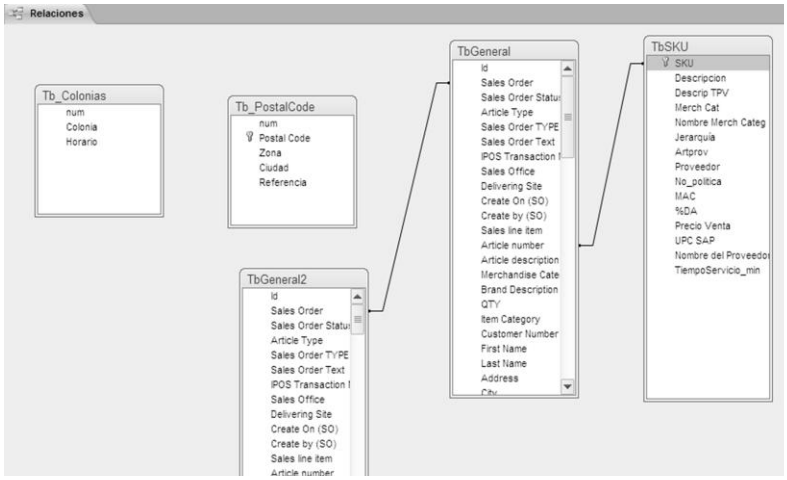


Figura 3.3 Vista de tablas relacionadas en Access

Consultas. Si trabajamos con tablas relacionadas, es posible crear vistas condensadas tomando solo los campos que nos son útiles de las tablas relacionadas entre sí. A estas vistas condensadas se les llama consultas. Además de visualizar solo los campos que nos son útiles, también es posible colocar dentro de las consultas filtros de registros, se pueden incrustar nuevos campos con funciones y cálculos matemáticos sobre los campos, actualizar grupos de registros de tablas, incluso crear nuevas tablas a partir de las consultas. A diferencia de una hoja de cálculo como Excel, una vez creadas las consultas de Access, estas realizan cálculos matemáticos o ejecutan funciones en tiempo real sobre el grupo de registros seleccionado y no es necesario copiar y pegar formulas como se realiza en Excel.

Las consultas le dan poder de selección y operación matemática a la base de datos. Las consultas son la fuente de datos para el resto de objetos básicos que contiene Access, por ejemplo los formularios o los reportes.

En la figura 3.4 se muestra una consulta del proyecto en la vista diseño. Dentro de esta se incluyen las tablas relacionadas que la componen y algunos de los campos que se desean visualizar.

Se observan un filtro de texto “como 600” debajo del campo *“Sales Order”.* Este filtro lo que hace es seleccionar solo los registros que contengan como *“sales order”* que comiencen con un texto 600XXXXX. Dentro de los *“sales order”* existen otro tipo de folios, pero para los intereses particulares de esta consulta, solo se filtran los que empiezan con los dígitos 600.

Se observa un campo *“Week”* (semana) que se creó para calcular y visualizar en un formato “ww” el número de la semana transcurrido a partir del campo *“Sales Date”* (Fecha de venta). El cálculo se realiza colocando el siguiente texto dentro del generador de expresiones para el campo *“Week”: ParcFecha("ww",[Sales Date],5)-1*

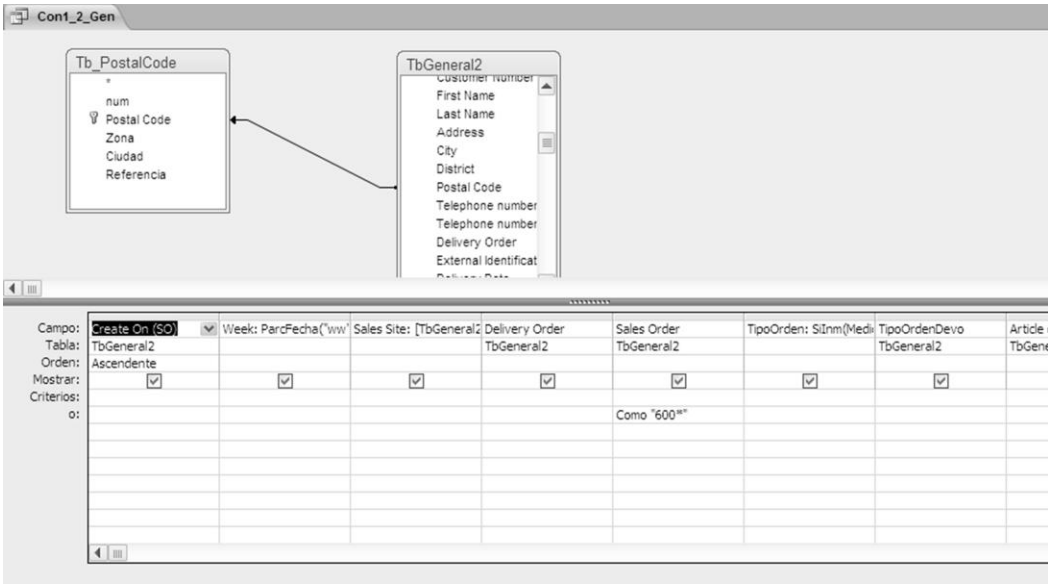


Figura 3.4 Vista de una consulta de datos en Access

Formularios. Estos son una interfaz gráfica que se utilizan para visualizar los datos y se pueden alimentar a partir de las consultas e incluso las tablas tan solo ligándolos a ellos. Los formularios son un tipo de objeto que puede tener a su vez otros objetos conocidos como controles. Entre los controles podemos encontrar botones de mando, cuadros de texto, listas desplegables, cuadros de opción y todo tipo de controles conocidos y permitidos por la interfaz de Windows. Existen varios tipos de formularios y se pueden visualizar los datos incluso en forma de hoja de datos (como Excel) con controles incrustados tales como botones de listas desplegables, etc., o podemos visualizar los datos en ventanas con tantos cuadros de texto como campos se requieran y controles que ejecutan alguna función. Estos controles pueden ejecutar alguna acción predeterminada por Access o se puede crear alguna a partir de la inserción de código VBA. Esta propiedad de programación de objetos conducida por eventos con VBA dentro de los formularios es la que permitió darle un nivel de aplicación adicional a la base de datos y ha sido aprovechada para el proyecto *"Nosy"*.

Reportes. Los reportes son otro tipo de formularios, pero su función principal es crear un formato de documento imprimible o archivo adjunto para algún mail que se genera a partir de los datos obtenidos de consultas diseñadas específicamente para ello.

La base de datos del call center tiene una cantidad considerable de tablas, consultas, formularios y reportes para su operación diaria y administración. En esta memoria solo se mostrarán los formularios y códigos relativos a la operación con el dispositivo *"Nosy"*.

3.5 Implementando la conexión Access - Nosy.

Para que la base de datos de call center funcione con el *"Nosy"*, se debe programar Windows para que responda adecuadamente al comando de teclas "CTR+ALT+F". Para esto, debe existir un icono para acceso a la base en el escritorio de la computadora. Una vez creado, apuntando sobre el icono y oprimiendo el botón derecho del ratón, aparece un menú del cual si se selecciona "propiedades", se verá una ventana como la de la figura 3.5.

Figura 3.5 Ventana propiedades de icono en escritorio de Windows

En el campo llamado “Tecla de método abreviado” se sostiene la tecla de “Ctrl” y sin soltarla se oprime la letra “F”. En el campo correspondiente aparecerá “CTRL + ALT + F”. Se oprime aceptar y queda grabado dicho comando. Cada vez que se oprima la secuencia de teclas “CTRL+ALT+F” se abre la base de datos “BD_CallC_V2.3” (figura 3.6).

Figura 3.6 Asignación de CTR+ALT+F para apertura de base de datos

En caso de que ya esté abierta la base pero se encuentre minimizada o detrás de alguna otra aplicación abierta, por ejemplo Excel, el comando “CTR+ALT+F” trae la base de dato a primer plano. Esto es necesario debido a que el *“Nosy”* transmite tres comandos en forma secuencial como se ha explicado anteriormente:

Primero, “CTRL + ALT + F” para llamar la base o ponerla en primer plano.
Segundo, “CTRL + E” para abrir el formulario de control telefónico
Tercero, “CTRL + J” Para indicar a la base el descolgado o “CTR + D” para indicar el colgado del teléfono.

“CTRL+E, CTRL+J y CTRL+D” solo son reconocidos por la base de datos, pero “CTR+ALT+F” debe ser ejecutado por Windows. Sin la programación correcta de “CTR+ALT+F” los demás comandos no tendrían sentido.

Cuando se abre la base de datos, la primera ventana que se muestra es un formulario de control llamado “FormControl” (Figura 3.7).

Este formulario es llamado en primera instancia debido a que se programa para ello desde Archivo\Opciones\Base de datos Actual\Mostrar formulario (la secuencia depende de la versión de office, para esta memoria se utilizó la versión 2010).

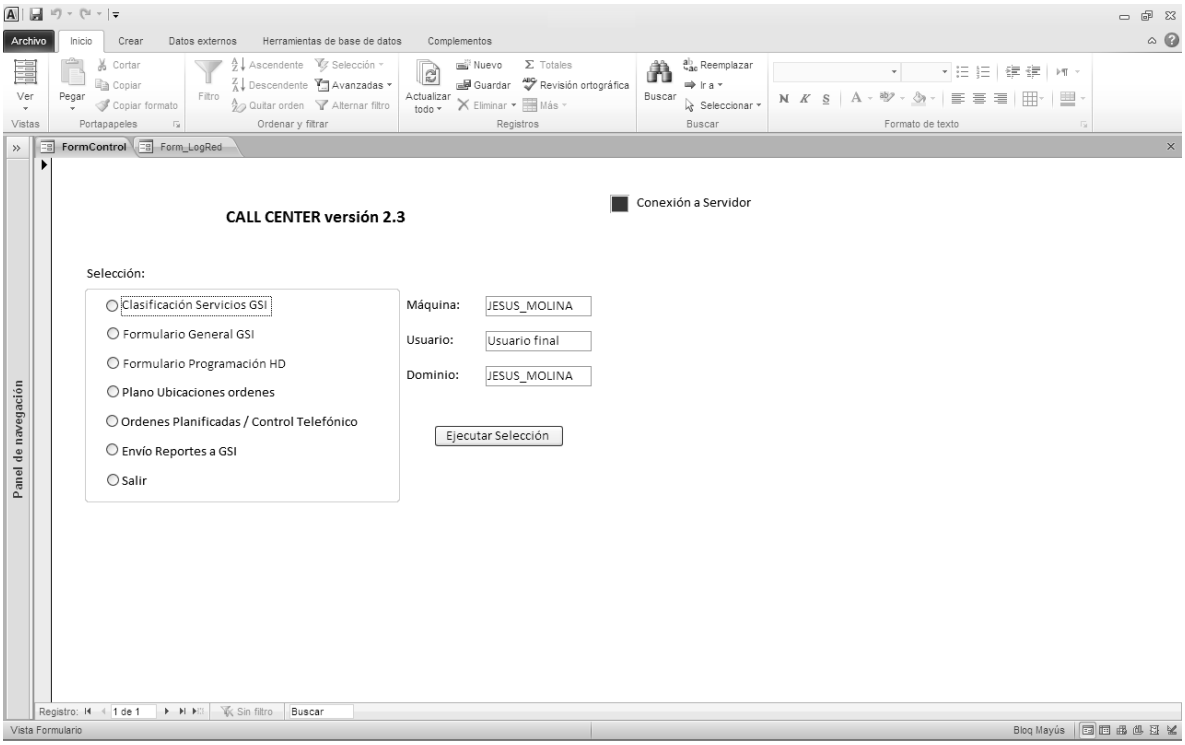


Figura 3.7 Vista del formulario principal de la aplicación en Access

Tiene un cuadro de selección con varias opciones que permiten ejecutar varias operaciones administrativas del proceso de programación de entregas del call center. Es de interés en esta memoria solo el formulario “Ordenes Planificadas / Control Telefónico” (FormItinerario).

Del lado derecho del formulario “FormControl” aparece un cuadro de control que cambia de color y funciona como un semáforo nombrado “Conexión a servidor”. Este es un indicador de la conexión al servidor que mediante el envío periódico de comandos TCP / IP, monitorea la conexión al mismo para mostrarles a los usuarios que están comunicados con el servidor donde reside la base de datos principal. Se pone en verde cuando hay comunicación con el servidor y en rojo cuando no lo está. El código para llevar esta tarea se describe al final de este capítulo.

Hay tres campos importantes: Máquina, usuario y dominio que registran la información del usuario contenida en el sistema de la computadora. Estos datos son utilizados en un formulario denominado “Form_LogRed” creado a partir de una tabla de datos para tal fin, donde se pretende que se generen registros con los datos contenidos en dichos campos además de la fecha, hora del evento y evento mismo cada vez que ocurra alguno y sea detectado por el *“Nosy”*. La finalidad de estos datos es llevar un registro no solo de los eventos detectados por el *“Nosy”*, sino también del usuario y computadora que los realiza. Los eventos de interés que registra esta tabla corresponden al colgado, descolgado, número marcado, etc.

Cada ocasión que el formulario de control se abre, dentro del evento de carga de formulario se ejecuta un código que abre el “Form_LogRed” y lo pone oculto detrás del “FormControl”.

La sintaxis en VBA para abrir el “Form_LogRed” por el evento “Form_Load()” es el siguiente:

```
Private Sub Form_Load()
    [ Código adicional utilizado por la aplicación ]

DoCmd.OpenForm "Form_LogRed", acFormDS        ‘Abre Form_LogRed en vista Hoja de datos.

    [ Código adicional utilizado por la aplicación ]
End sub
```

La vista del “Form_LogRed” cuando se selecciona pulsando con el ratón sobre él es parecida a la vista que se muestra en la figura 3.8.

La ventana de la figura 3.8 muestra la información de los eventos detectados por el *“Nosy”* más otros datos adicionales que servirían incluso para generar estadísticos y métricas de la operación del call center. El registro de estos datos son el objetivo final del proyecto *“Nosy”*.

Para obtener los datos de usuario del sistema dentro del evento “Form_load()” del formulario “FormControl” se introduce el siguiente código VBA:

```
Private Sub Form_Load()

    [ Código adicional utilizado por la aplicación ]

'OBTENER IDENTIFICADOR USUARIO
Dim wshNetwork As Object 'New wshNetwork
Set wshNetwork = CreateObject("WScript.Network")

Maquina = wshNetwork.ComputerName
Usuario = wshNetwork.UserName
Dominio = wshNetwork.UserDomain

Set wshNetwork = Nothing
    [ Código adicional utilizado por la aplicación ]

End sub
```

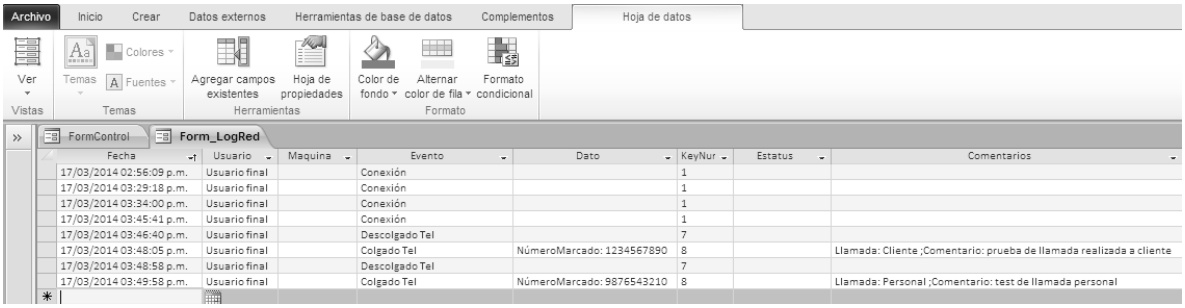


Figura 3.8 Vista Formulario "Log" de datos

3.6 Descripción del formulario para el Control Telefónico en Access.

Este formulario (FormItinerario) y su respectiva consulta de datos fue diseñada para registrar los eventos detectados por el dispositivo *"Nosy"*. Se abre seleccionando "Ordenes Planificadas / Control Telefónico" desde el Formulario de Control o tecleando "CTRL+E". También se abre inmediatamente después de que el dispositivo *"Nosy"* ha enviado la segunda secuencia de comando (CTRL+E) posterior al "CTRL+ALT+F" (apertura de base de datos).

Este formulario tiene un calendario diseñado para filtrar aquellas ordenes de compra que previamente el usuario de call center tiene previstas programar en determinada fecha, es decir; aquellas ordenes seleccionadas para que el usuario hable telefónicamente a los clientes correspondientes para establecer y confirmar una fecha de entrega e instalación de sus productos adquiridos en tiendas.

Este formulario muestra un subformulario visualizado en forma de hoja de datos que muestra los campos relevantes de la orden de compra tales como número de orden, nombre del cliente, domicilio, teléfono, etc. y permitirá al usuario utilizarlos ya sea para iniciar una conversación telefónica o consulta durante la misma.

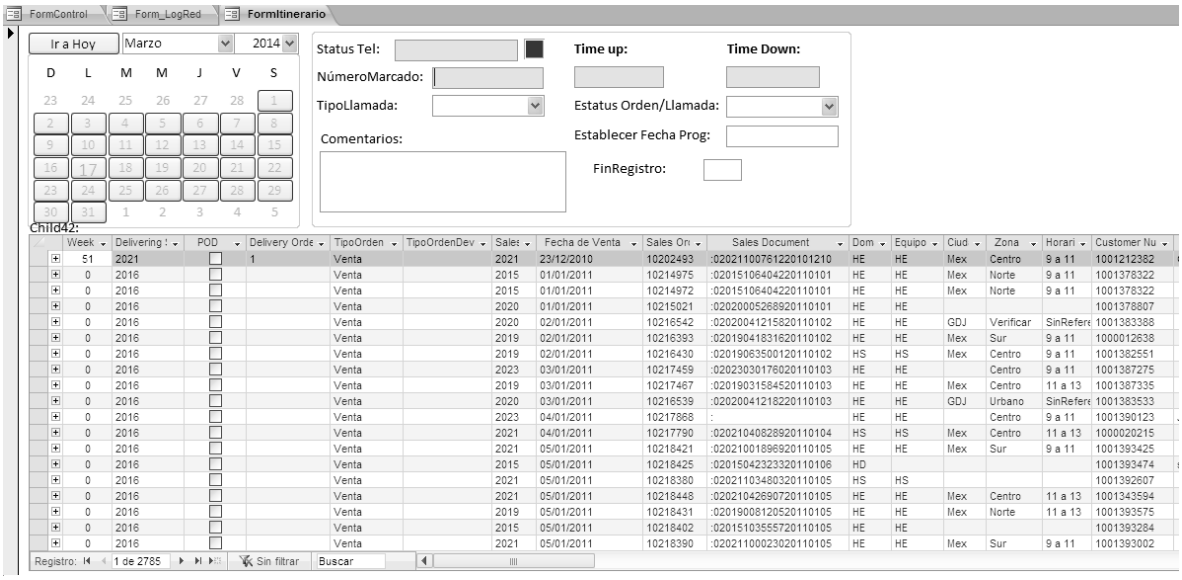


Figura 3.9 Vista formulario para control telefónico en Access

Finalmente, muestra una serie de campos que son utilizados por el *"Nosy"* y por el mismo usuario durante una llamada telefónica. Este formulario se muestra en la figura 3.9.

3.7 Las macros de la base de datos que responden a los comandos de Nosy.

Para que la base de datos responda a la secuencia de comandos enviados por el dispositivo *"Nosy"*, primero debe estar abierta y en primer plano (como resultado del CTRL+ALT+F enviado por el Nosy). Después de enviar este comando y retardado por 9 millones de ciclos de instrucción (aproximadamente 0.75 segundos), el *"Nosy"* envía de forma autónoma un segundo comando: CTRL+E, que para ese instante se supone la base de datos ya debe estar abierta y entonces abre el formulario "FormItinerario" o de control telefónico. Finalmente y de forma secuencial *"Nosy"* envía ya sea un comando "CTRL+J" (Descolgado) o un "CTRL+D" (Colgado) dependiendo de la actividad realizada en el aparato telefónico que ejecutara finalmente alguna acción dentro del "FormItinerario" relacionada con el evento correspondiente.

Para que la base de datos responda a los comandos "CTR+E, CTRL+J y CTRL+D" es necesario crear unas macros propias de la base de datos Access.

La forma en que se crearon estas macros utilizando la versión 2010 de office es la siguiente:

. Macro CTR+E.

Esta macro debe abrir el formulario "FormItinerario", para ello se selecciona "Crear" del menú principal. Después se selecciona "Macro", inmediatamente aparece la ventana de la figura 3.10.

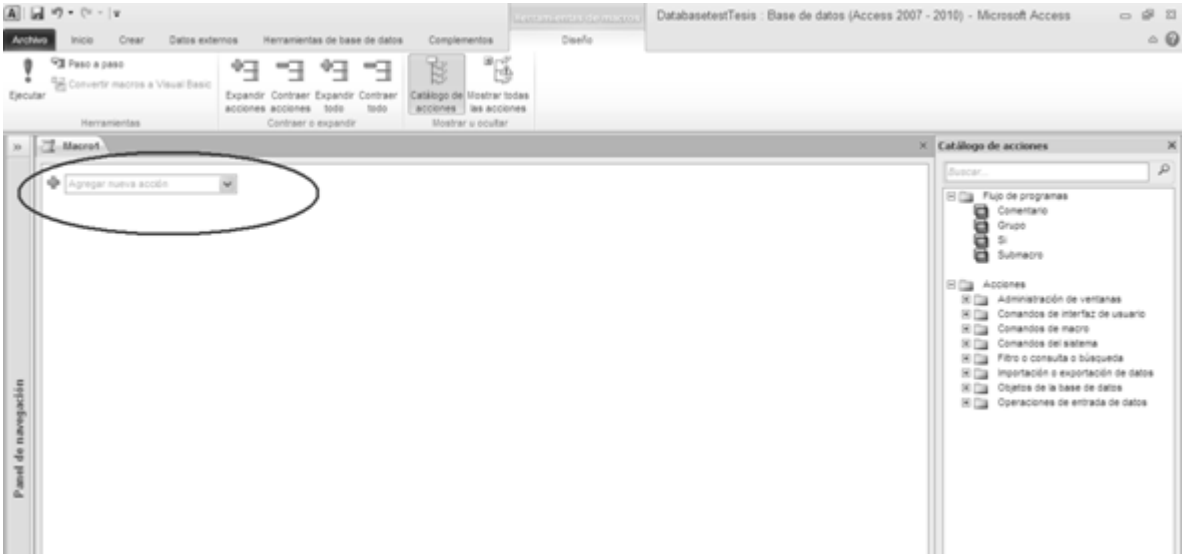


Figura 3.10 Creando macros para la aplicación. Paso 1

De la figura anterior, en el cuadro señalado por el círculo, se abre una lista, entonces se escoge "Submacro" mostrado en la figura 3.11.

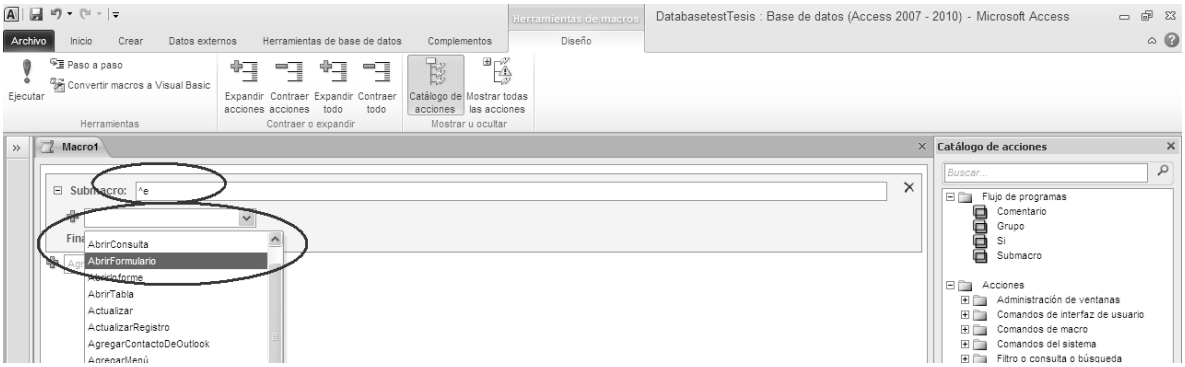


Figura 3.11 Creando macros para la aplicación. Paso 2

En el campo de texto "submacro" se da nombre a la misma que corresponde al comando con que se invoca a dicha submacro. Para este caso se escribe "^e". "^" es el indicativo para Access de "CTRL". Después se abre el cuadro desplegable y se selecciona la acción "Abrir formulario". Después de esto y como se muestra en la figura 3.12 se selecciona el nombre del formulario que se desea abrir (FormItinerario para la aplicación).

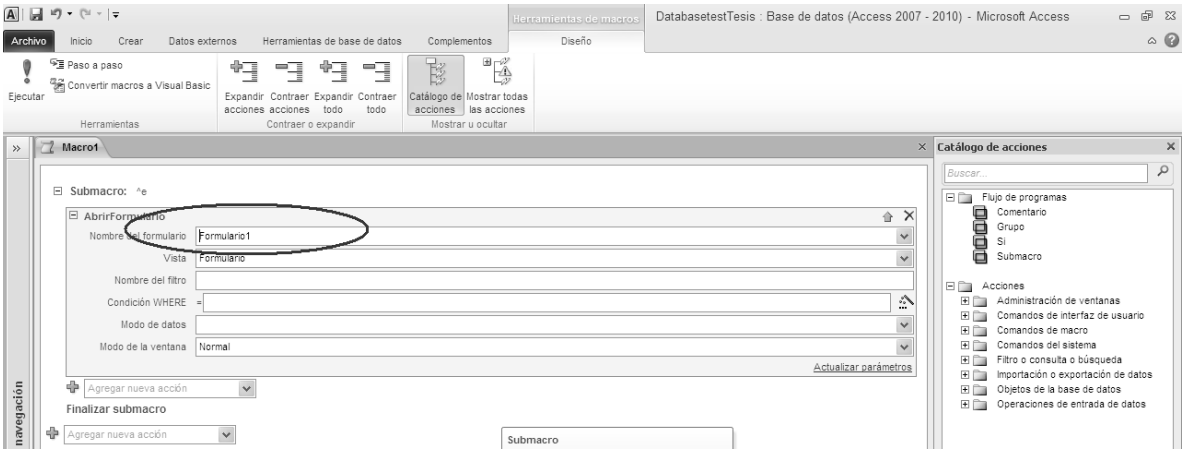


Figura 3.12 Creando macros para la aplicación. Paso 3

Finalmente, se guarda la macro nombrándola "Autokeys" (figura 3.13).

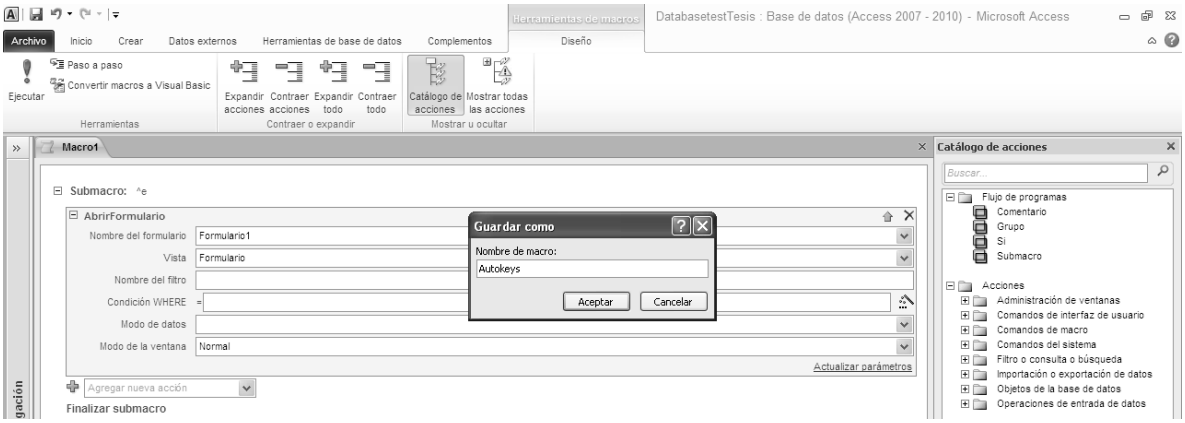


Figura 3.13 Creando macros para la aplicación. Paso 4

.Macro CTR+J

Esta macro realiza la acción de invocar la función “Descolgado()” creada en VBA que cumple la tarea de señalizar y notificar el evento de descolgado dentro de “FormItinerario” de la base. Para crearla, se abre de nuevo la macro “Autokeys” en vista diseño, agregando una nueva submacro y nombrándola ahora como “^j”. Después como acción se selecciona “EjecutarCodigo” y se escribe el nombre de la función denominada “Descolgado ()” y se guarda de nuevo la macro (figura 3.14).

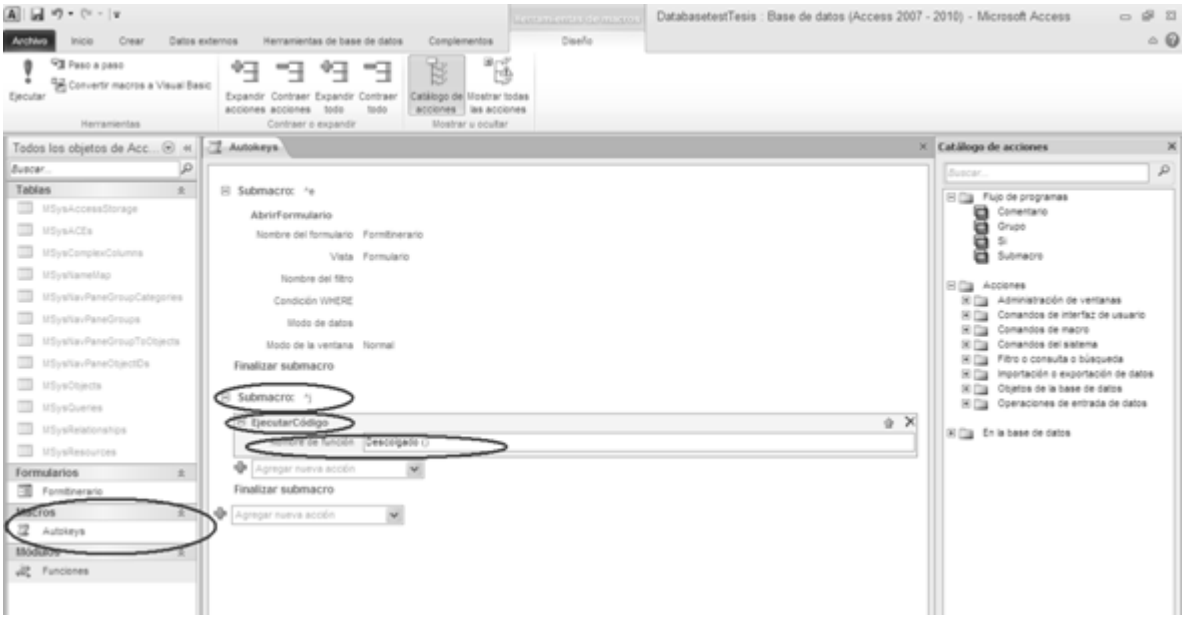


Figura 3.14 Creando macros para la aplicación. Paso 5

.Macro CTR+D.

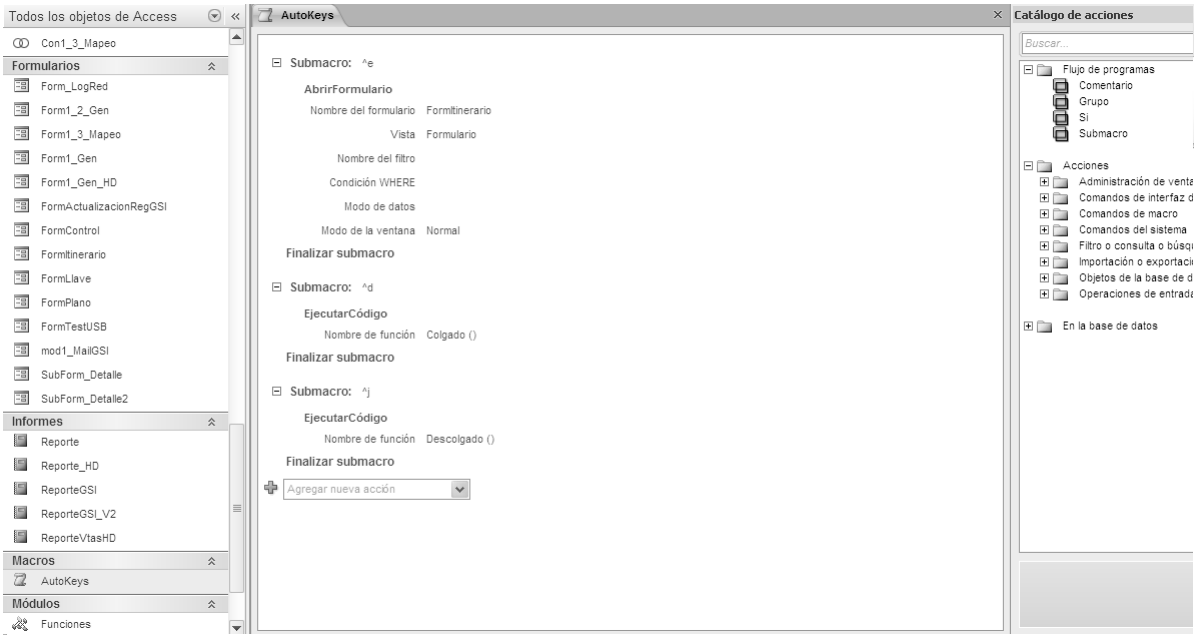


Figura 3.15. Creando macros para la aplicación. Paso 6

Esta macro de igual manera invoca la función “Colgado ()” creada en VBA. Esta función cumple la tarea de ejecutar rutinas sobre “FormItinerario” relativas al evento de colgado telefónico. Para crearla se ejecuta el mismo procedimiento que en “CTR+J”, solo que nombrando la submacro como “^D” y en la acción “EjecutarCodigo” nombrar la función “Colgado ()”. La estructura final de las macros se muestra en la figura 3.15.

3.8 Función VBA: Descolgado ()

Esta función se ubica en un módulo llamado “Funciones” del código de VBA para la base de datos. El código de la función en este módulo se muestra a continuación y su resultado en la figura 3.16.

```
Option Compare Database
Public Function Descolgado()

Forms!formItinerario!StatusTel = "Descolgado"
Forms!formItinerario!LampTel.BackColor = &HFF00&
Forms!formItinerario!TimeUp = Now()

Forms!formItinerario!NumTel.SetFocus
Forms!formItinerario!NumTel = ""
Forms!formItinerario!Combo63 = ""
Forms!formItinerario!ComboEstatusLlamada = ""
Forms!formItinerario!Comentarios = ""
Forms!formItinerario!SetFechaprog = ""

DoCmd.GoToRecord , "form_logRed", acNewRec
Forms!form_logRed!Usuario = Forms!formcontrol!Usuario
Forms!form_logRed!Fecha = Now()
Forms!form_logRed!Evento = "Descolgado Tel"
Forms!form_logRed!KeyNum = 7
DoCmd.GoToRecord , "form_logRed", acNewRec

End Function
```

En el formulario FomItinerario:
Pone el texto “Descolgado” en el campo StatusTel.
Cambia a color verde el indicador junto al campo Status Tel y coloca la fecha y la hora del sistema en el campo TimeUp.

Cambia el foco al campo NumTel para que cuando el Nosy detecte una marcación numérica se registren los datos enviados por el Nosy en este campo.
Limpia los campos tipo y estatus de llamada y comentarios para que puedan ser utilizados por el usuario.

En el formulario Form_LogRed:
Crea un nuevo registro.
Escribe en los campos respectivos del registro el usuario, la fecha y hora del sistema, la acción ”Descolgado Tel” principalmente.

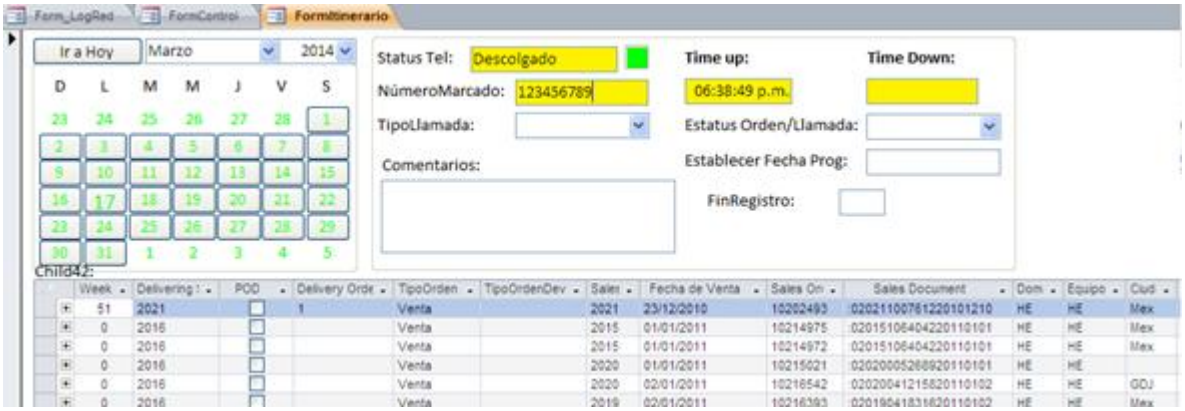


Figura 3.16 Vista del resultado de la ejecución de la función Descolgado ()

Una vez que se ha llamado esta función, el usuario está en posición de marcar un número telefónico lo que es detectado por el *"Nosy"* y enviado secuencialmente como códigos de teclas lo que será registrado en el campo "NúmeroMarcado".

Durante la conversación, el usuario puede realizar otras actividades como poner comentarios incluso pasar a otra aplicación. Esta condición permanecerá hasta que ocurra un evento de Colgado que pondrá al frente la base de datos (CTRL+ALT+F) y pondrá también al frente de cualquier aplicación el "FormItinerario" (CTRL+E), entonces será ejecutada la función Colgado() invocada por la macro ^d debido al comando "CTRL+D" proveniente del *"Nosy"*.

3.9 Función VBA: Colgado ().

Esta función también se ubica en un módulo llamado "Funciones" del código de VBA para la base de datos y su código se ejecuta después de ser invocada la macro ^d.
Su código es el siguiente:

```
Public Function Colgado()
Forms!formItinerario!FinText.SetFocus
Forms!formItinerario!StatusTel = "Colgado"
Forms!formItinerario!LampTel.BackColor = &HFF&
Forms!formItinerario!TimeDown = Now()

Forms!form_logRed!Usuario = Forms!formcontrol!Usuario
Forms!form_logRed!Fecha = Now()
Forms!form_logRed!Evento = "Colgado Tel"
Forms!form_logRed!Dato = "NúmeroMarcado: " & Forms!formIti
Forms!form_logRed!KeyNum = 8
Forms!form_logRed!Estatus = Forms!formItinerario!ComboEstatu
Forms!form_logRed!Comentarios = "Llamada: " & Forms!formItinerario!Combo63 & " ;Comentario: " &
Forms!formItinerario!Comentarios
DoCmd.GoToRecord , "form_logRed", acNewRec

End Function
```

En el formulario FomItinerario:
Pone el texto "Colgado" en el campo StatusTel.
Cambia a color rojo el indicador junto al campo Status Tel y coloca la fecha y la hora del sistema en el campo TimeDown.

En el formulario Form_LogRed:
Escribe en un nuevo registro el evento de Colgado registrando la fecha, hora e incluso el número marcado y los comentarios.

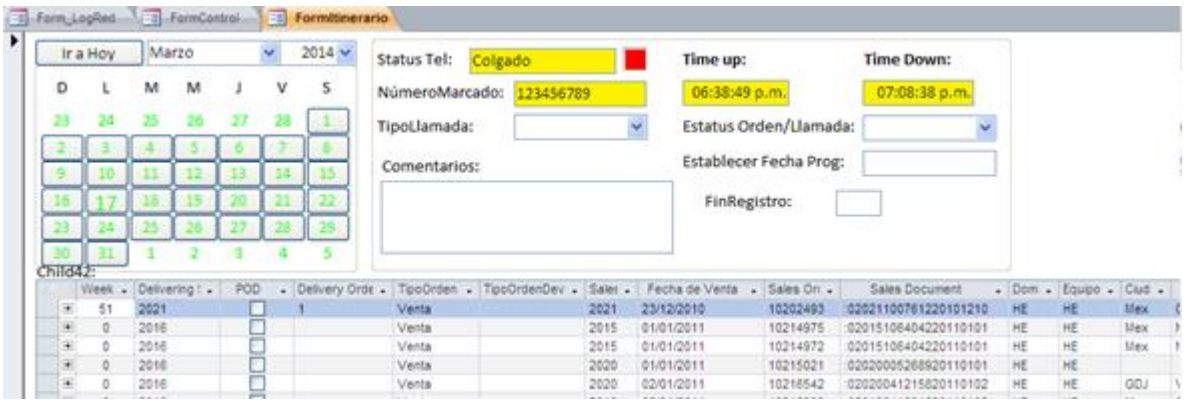
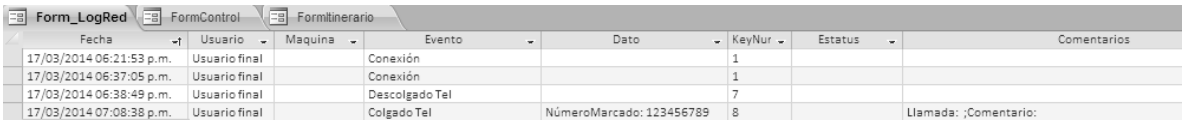


Figura 3.17 Vista del resultado de la ejecución de la función Colgado ()

Los registros del log después de un evento de descolgado y colgado queda como se muestra en la figura 3.18.



Fecha	Usuario	Maquina	Evento	Dato	KeyNur	Estatus	Comentarios
17/03/2014 06:21:53 p.m.	Usuario final		Conexión		1		
17/03/2014 06:37:05 p.m.	Usuario final		Conexión		1		
17/03/2014 06:38:49 p.m.	Usuario final		Descolgado Tel		7		
17/03/2014 07:08:38 p.m.	Usuario final		Colgado Tel	NúmeroMarcado: 123456789	8		Llamada: ;Comentario:

Figura 3.18 Vista final del formulario "Log" después de concluida una llamada telefónica

3.10 El monitoreo vía red al servidor de datos.

La base de datos mostrada hasta ahora comenzó a ser utilizada dividida conceptualmente en dos partes. Una parte de la aplicación funcionaba como una terminal de datos montada en cada computadora de los usuarios u operadores de call center que efectuaban los cambios en los registros afectando en tiempo real la otra parte conocida como matriz y que residía en una ubicación específica de un servidor de la red de datos. Con el tiempo y debido a las dificultades obtenidas principalmente por la velocidad de trabajo de Access en red, se modificó la forma en que intercambiaban datos entre las terminales con la base matriz transformándolas a un modo conocido como "sincronización de réplicas". Bajo este modo, los datos se vaciaban hasta que sincronizaran tanto réplica como base matriz dándole mayor velocidad de ejecución a cada terminal. Sin embargo, tanto en los modos de tiempo real como en sincronización de réplicas solía ocurrir que cuando la comunicación con la base del servidor se perdía, la base de datos de cada terminal dejaba de funcionar e incluso se cerraba. Regularmente, cuando existe un fallo en Access, este envía un mensaje de error indicando la posible falla de lo ocurrido, sin embargo la mayoría de estos mensajes se encontraban en inglés lo cual dificultaba el entendimiento de lo ocurrido a algunos de los usuarios reportando la incidencia como "base inservible o dañada". Las fallas realmente ocurrían ya sea porque la comunicación de la red local o algún nodo no funcionaba, o porque el servidor donde reside la base matriz se encontraba fuera de línea.

Como una medida de otorgar a los usuarios de la base visibilidad cuando la red no estaba funcionando y específicamente cuando la comunicación con el servidor de la base de datos era nula, se buscó la forma de incorporar alguna herramienta que mostrara a los usuarios la conexión con el servidor.

De la experiencia obtenida en Ciasa años atrás con el diagnóstico, instalación y prueba de cajeros automáticos en sucursales bancarías, bajo ambiente de MSDOS se utilizaban una serie de comandos del protocolo de comunicación TCP/IP para probar la comunicación del cajero con un servidor central. Uno de esos comandos es el "Ping". El comando Ping envía una llamada a un equipo remoto e informa si se logra establecer comunicación o no con él.

Una muestra del uso de este comando en MSDOS es como se muestra en la siguiente figura 3.19.

Donde se ejecuta el comando ping a la dirección IP de la página www.terra.com.mx con dirección IP “208.70.188.180”. Después de este comando, en la figura se muestra que el comando envió 4 paquetes de datos y recibió 4 sin perder ninguno como respuesta del “208.70.188.180”.

En base lo anterior, se requería lograr lo mismo pero con código VBA sin tener que abrir la consola de MSDOS.

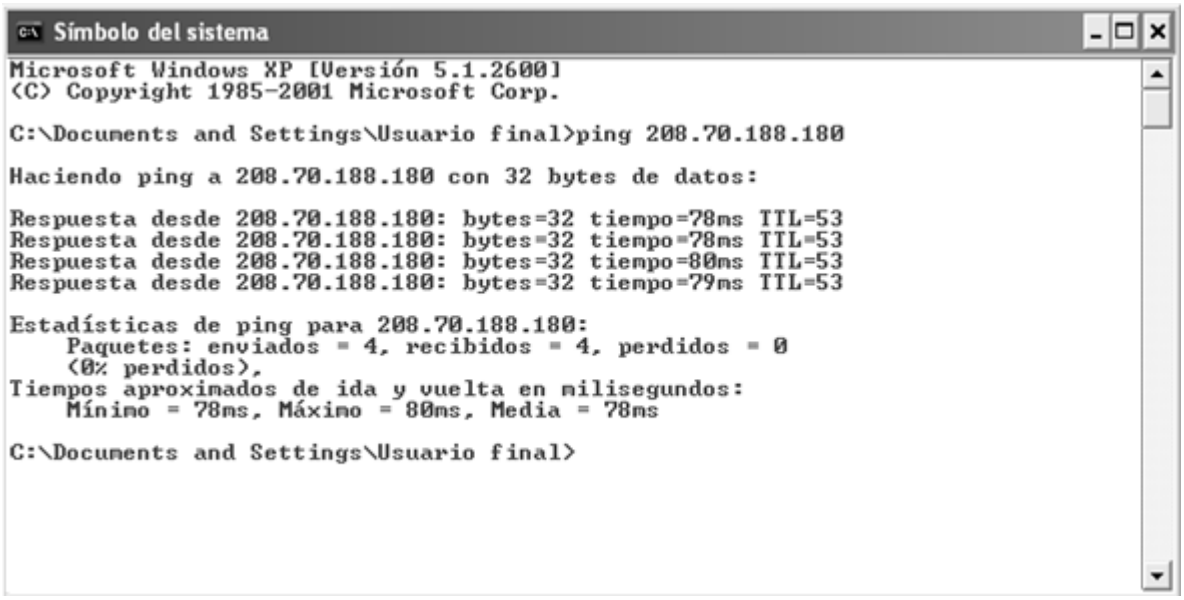


Figura 3.19 Ejemplo de ejecución del comando Ping en MSDOS

Para llevar a cabo la tarea, se utilizaron funciones API en conjunto con VBA de la siguiente manera:

En la base datos Access en el formulario “FormControl” se agregó al principio del evento “Form_Load()” el siguiente código VBA:

```
Private Sub Form_Load()
'Set timer
Me.TimerInterval = 3000
.
.
.
```

El código “Me.TimerInterval=3000” establece un temporizador por un intervalo de 3 segundos. Si se escribe código dentro del evento “Form_Timer()” de dicho formulario y siempre y cuando éste se encuentre abierto, cada 3 segundos se ejecutará el código contenido en “Form_Timer()”.

En el evento “Form_Timer()” del formulario de “FormControl” se escribe el código que se muestra a continuación y que básicamente lo que hace es llamar una función ping para la dirección IP

establecida en la variable “TargetIP”. Despendiendo de la respuesta del valor retornado por Ping, cambiará de color un cuadro denominado “Conexión”. Rojo para falla en conexión y verde para éxito:

```
Private Sub Form_Timer()
'*****RUTINA PING A SERVIDOR**************
   ' estructura con la información para hacer el Ping
   Dim ECHO As ICMP_ECHO_REPLY
   Dim TargetIP As String ' dirección ip
   Dim Ret As String

   TargetIP = "10.7.233.1"

   If Len(TargetIP) > 0 Then

      ' hace el Ping
      ''''''''''''''''''''''''''''''''''''''''''''''''''''''
      Call Ping(TargetIP, ECHO)

      ' Estado del Ping
       If ECHO.status = 0 Then
         Conexion.BackColor = &HFF00&        'Semáforo se pone de color verde
       Else
         ' error al hacer ping
         Conexion.BackColor = &HFF&          'Semáforo se pone de color rojo
       End If
   Else
      ' error
   End If
      '****FIN DE RUTINA PING A SERVIDOR********
End Sub
```

La rutina anterior no sería posible sin la implementación de un módulo adicional dentro de la base datos que contiene la declaración de las funciones API basadas en la librería “icpm.dll” y las funciones que son llamadas por “Form_Timer()”. La definición de la función Ping se encuentra dentro de dicho módulo y es construida a base de otras funciones API declaradas en dicho módulo y pertenecientes a la librería “icpm.dll”. El código VBA del módulo se muestra completo en el Anexo 2.

Hasta este punto se ha descrito en su totalidad todo el desarrollo para la implementación del circuito electrónico *“Nosy”* y la forma en que se logró la interacción del dispositivo con una base de datos para obtener el funcionamiento deseado. No todo funcionó a la primera. Hubieron ciertas dificultades, pero estas y otras cosas son comentadas en la sección “Conclusiones”.

CONCLUSIONES

La aplicación descrita a lo largo de esta memoria representa un ejemplo de soluciones relativamente económicas en el ámbito operativo de una empresa y que pueden ayudar a salir del paso para ciertos tipos de procesos. Por otro lado, implementar soluciones de este tipo tal vez no resulte tan sencillo, ya que implica del conocimiento de diversas áreas como comunicaciones, programación en diversos lenguajes, programación de microcontroladores, etc. Por lo que podría ser un campo mejor explotado por los egresados de Ingeniería.

Como se ha mencionado antes, aprovechar herramientas tan a la mano y relativamente sencillas como Microsoft Access permite crear un sin fin de soluciones. En el pizarrón y en el escritorio se quedaron algunos proyectos e ideas sin iniciar o autorizar como la implementación de una base datos que registrara el flujo de servicios y que respondiera automáticamente a mensajes SMS (Short Message Service) enviados por los clientes solicitando información sobre los mismos lo que reduciría potencialmente el costo de una sección de call center dedicado a ello además de dar prontitud de respuesta a los clientes mejorando el servicio.

Otro proyecto que se quedó sin realizar sería la implementación de un proceso y sistema para registrar en video el conteo de las transferencias de mercancía entre el Centro de recuperación de devoluciones y tiendas mezclando tanto video real digitalizado con caracteres provenientes de la lectura del código de barras del producto lo que mejoraría la aclaración de diferencias y reducción de mermas en la empresa y aunque no utiliza este planteamiento Access, si requería para su desarrollo del diseño electrónico y programación.

También del dispositivo *"Nosy"* se requerían mejoras a futuro, ya que durante la etapa de diseño se encontraron algunos inconvenientes. Era necesario el intercambio de eventos entre el *"host"* y el *"Nosy"* para obtener una mejor respuesta de parte de Windows. Sucedía que a veces Windows se cuelga en alguna tarea y no es tan rápido para responder a las instrucciones de *"Nosy"* debido a la capacidad del procesador, de su memoria o de la cantidad y tamaño de aplicaciones abiertas a la vez. Durante el diseño, se compenso sensiblemente esto proporcionándole a *"Nosy"* retardos entre cada instrucción transmitida de hasta 0.75 segundos para darle tiempo a Windows de abrir o colocar en primer plano la base de datos antes de recibir la siguiente instrucción y aunque ello dio resultado, cabía la posibilidad aunque mínima de que el proceso pudiera fallar. También requeriría mejoras en la sección de audio con la finalidad de reducir lo ruidos inducidos en las conversaciones telefónicas debido al *"Nosy"*, en parte dependientes de las fluctuaciones de energía eléctrica provenientes de la alimentación de +5V a través del puerto USB y debido al aislamiento del amplificador de audio y la construcción de la placa de circuito impreso del *"Nosy"* que inducía apenas perceptibles ruidos debido a las capacidades parásitas en la placa. Es incuestionable esto, ya que el mismo circuito montado en principio en la placa de prueba no generaba dichos ruidos. Afortunadamente, debido a que la calidad de audio obtenida de las redes Cisco es muy superior comparada con la obtenida en la red telefónica pública convencional, el efecto llegaría a ser

despreciable, pero si se tratase de implementar para teléfonos de la red pública, las mejoras deberían ser efectuadas en este punto.

Es posible también reducir y optimizar el código del programa principal concentrando algunas rutinas adicionales en funciones para ser llamadas solo cuando sean necesarias, reduciendo el espacio de memoria que ocupan actualmente.

Un requerimiento también que se tenía proyectado era el implementar la digitalización y grabación de las conversaciones telefónicas para fines de calidad y monitoreo del proceso de call center. Este podría ser implementado físicamente utilizando los convertidores analógico a digital del mismo PIC18F4550 sin tener que realizar grandes cambios en la placa de circuito impreso. Lo que requeriría de mayor esfuerzo sería la realización del programa para codificación y almacenamiento del audio.

Entre las dificultades que se encontraron para la utilización final y presentación del *"Nosy"* fue la fabricación del Chasis o cubierta. Para darle presentación se fabricó de manera artesanal un chasis de pasta utilizada en artes para modelaje. Se elaboró a partir de ese chasis un molde de silicón y se hizo un vaciado en él de resina plástica mezclada con polvo de almidón coloreado en negro. El objetivo de utilizar el polvo de almidón fue el de reducir la estática en el chasis de resina y proteger los dispositivos de descargas eléctricas lo que parece haber dado buenos resultados. El acabado se muestra en el anexo 3.

Se tuvo el prototipo 2 del dispositivo finalmente bajo pruebas piloto cerca de tres meses en dos fases, cada una con un usuario diferente. Durante la primera fase se hicieron adecuaciones y modificaciones necesarias tanto al dispositivo como a la base de datos de acuerdo a los requerimientos del usuario 1 que es la versión que se muestra en la presente memoria. Ya habiendo cumplido con los requerimientos de la fase 1 se sometió a la segunda fase de pruebas y aunque de manera personal observaba que el funcionamiento del dispositivo era excelente, existían frecuentemente observaciones por parte del usuario 2 relacionados con los cambios y disciplina requerida para la operación en conjunto con la base de datos, pues les representaba un verdadero paradigma a la acostumbrada operación que el área tenía anteriormente. Pero el dispositivo realmente registraba los eventos y daba visibilidad clara de las actividades que se realizaban por el personal de call center permitiendo contar con métricas más precisas.

El dispositivo en sí no representaba ninguna complejidad en su uso e instalación, además de que tenían los usuarios a su alcance la facultad de reinicializarlo en caso de que este fallara.

Como se demuestra a lo largo de este texto, se dio solución a una problemática administrativa y de operación de un área específica en una empresa aplicando de forma práctica conocimientos tecnológicos y de ingeniería.

ANEXO 1

Código completo de Keyboard.c

```
#ifndef KEYBOARD_C
#define KEYBOARD_C

#ifndef  _18CXX
#define _18CXX

/** INCLUDES *******************************************************/
#include "p18f4550.h"
#include "./USB/usb.h"
#include "HardwareProfile.h"
#include "./USB/usb_function_hid.h"
#include "delays.h"

/** CONFIGURATION **************************************************/
  // Configuration bits based on PIC18F4550
     #pragma config PLLDIV   = 2                         // (8 MHz crystal para 4 Mhz entrada a PLL)
     #pragma config CPUDIV   = OSC1_PLL2                 // CPU Clock= 96MHz / 2 = 48 MHz (96MHz from PLL)
     #pragma config USBDIV   = 2               // Clock source from 96MHz PLL/2 para obtener 48MHz
      #pragma config FOSC    = HSPLL_HS      // HS oscillator,PLL enable, HS used by USB
     #pragma config FCMEN    = OFF
     #pragma config IESO     = OFF
     #pragma config PWRT     = OFF
     #pragma config BOR      = ON
     #pragma config BORV     = 3
     #pragma config VREGEN   = ON              //USB Voltage Regulator
     #pragma config WDT      = OFF
     #pragma config WDTPS    = 32768
     #pragma config MCLRE    = ON
     #pragma config LPT1OSC  = OFF
     #pragma config PBADEN   = OFF                       //Entradas RB0:RB4 digitales
//   #pragma config CCP2MX   = OFF
      #pragma config STVREN   = ON                       //Stack full reset
      #pragma config LVP      = OFF
//    #pragma config ICPRT    = OFF             // Dedicated In-Circuit Debug/Programming
      #pragma config XINST    = OFF             // Extended Instruction Set
      #pragma config CP0      = OFF
      #pragma config CP1      = OFF
//    #pragma config CP2      = OFF
//    #pragma config CP3      = OFF
      #pragma config CPB      = OFF
//    #pragma config CPD      = OFF
      #pragma config WRT0     = OFF
      #pragma config WRT1     = OFF
//    #pragma config WRT2     = OFF
//    #pragma config WRT3     = OFF
      #pragma config WRTB     = OFF             // Boot Block Write Protection
      #pragma config WRTC     = OFF
//    #pragma config WRTD     = OFF
      #pragma config EBTR0    = OFF
      #pragma config EBTR1    = OFF
//    #pragma config EBTR2    = OFF
//    #pragma config EBTR3    = OFF
      #pragma config EBTRB    = OFF


/** VARIABLES ******************************************************/

#pragma udata

BYTE oldSWstd;
```

```
BYTE oldCOMPARADOR;
int CUENTA;
unsigned char KEY1;
unsigned char KEY2;
unsigned char KEY3;
char buffer[8];
unsigned char OutBuffer[8];
USB_HANDLE lastINTransmission;
USB_HANDLE lastOUTTransmission;
BOOL Keyboard_out;

/** PRIVATE PROTOTYPES *********************************************/
BOOL SWstdIsPressed(void);
void ProcessIO(void);
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();
void USBCBSendResume(void);
void USBHIDCBSetReportComplete(void);
void Keyboard(void);

/**********poner aquí servicios de interrupción***********/
/** VECTOR REMAPPING ***********************************************/
#if defined(__18CXX)
	//On PIC18 devices, addresses 0x00, 0x08, and 0x18 are used for
	//the reset, high priority interrupt, and low priority interrupt
	//vectors.  However, the current Microchip USB bootloader
	//examples are intended to occupy addresses 0x00-0x7FF or
	//0x00-0xFFF depending on which bootloader is used.  Therefore,
	//the bootloader code remaps these vectors to new locations
	//as indicated below.  This remapping is only necessary if you
	//wish to program the hex file generated from this project with
	//the USB bootloader.  If no bootloader is used, edit the
	//usb_config.h file and comment out the following defines:
	//#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
	//#define PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOADER

	#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
		#define REMAPPED_RESET_VECTOR_ADDRESS			0x1000
		#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS	0x1008
		#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS	0x1018
	#elif defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
		#define REMAPPED_RESET_VECTOR_ADDRESS			0x800
		#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS	0x808
		#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS	0x818
	#else
		#define REMAPPED_RESET_VECTOR_ADDRESS			0x00
		#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS	0x08
		#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS	0x18
	#endif

	#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(
	PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOA DER)
	extern void _startup (void);        // See c018i.c in your C18 compiler dir
	#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
	void _reset (void)
	{
	  _asm goto _startup _endasm
	}
	#endif
	#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
	void Remapped_High_ISR (void)
	{
	   _asm goto YourHighPriorityISRCode _endasm
	}
	#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
	void Remapped_Low_ISR (void)
```

```
        {
            _asm goto YourLowPriorityISRCode _endasm
        }

        #if  defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(
PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
        //Note: If this project is built while one of the bootloaders has
        //been defined, but then the output hex file is not programmed with
        //the bootloader, addresses 0x08 and 0x18 would end up programmed with 0xFFFF.
        //As a result, if an actual interrupt was enabled and occured, the PC would jump
        //to 0x08 (or 0x18) and would begin executing "0xFFFF" (unprogrammed space).  This
        //executes as nop instructions, but the PC would eventually reach the REMAPPED_RESET_VECTOR_ADDRESS
        //(0x1000 or 0x800, depending upon bootloader), and would execute the "goto _startup".  This
        //would effective reset the application.

        //To fix this situation, we should always deliberately place a
        //"goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS" at address 0x08, and a
        //"goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS" at address 0x18.  When the output
        //hex file of this project is programmed with the bootloader, these sections do not
        //get bootloaded (as they overlap the bootloader space).  If the output hex file is not
        //programmed using the bootloader, then the below goto instructions do get programmed,
        //and the hex file still works like normal.  The below section is only required to fix this
        //scenario.
        #pragma code HIGH_INTERRUPT_VECTOR = 0x08
        void High_ISR (void)
        {
            _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
        }
        #pragma code LOW_INTERRUPT_VECTOR = 0x18
        void Low_ISR (void)
        {
            _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
        }
        #endif      //end of "#if  defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(
        PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOADER)"

        #pragma code


        //These are your actual interrupt handling routines.
        #pragma interrupt YourHighPriorityISRCode
        void YourHighPriorityISRCode()
        {
                    //Check which interrupt flag caused the interrupt.
                    //Service the interrupt
                    //Clear the interrupt flag
                    //Etc.
    #if defined(USB_INTERRUPT)
                USBDeviceTasks();
    #endif

        }           //This return will be a "retfie fast", since this is in a #pragma interrupt section
        #pragma interruptlow YourLowPriorityISRCode
        void YourLowPriorityISRCode()
        {
                    //Check which interrupt flag caused the interrupt.
                    //Service the interrupt
                    //Clear the interrupt flag
                    //Etc.

        }           //This return will be a "retfie", since this is in a #pragma interruptlow section

#elif defined(__C30__)
  #if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
    /*
     *      ISR JUMP TABLE
```

```
     *
     *      It is necessary to define jump table as a function because C30 will
     *      not store 24-bit wide values in program memory as variables.
     *
     *      This function should be stored at an address where the goto instructions
     *      line up with the remapped vectors from the bootloader's linker script.
     *
     *  For more information about how to remap the interrupt vectors,
     *  please refer to AN1157.  An example is provided below for the T2
     *  interrupt with a bootloader ending at address 0x1400
     */
//      void __attribute__ ((address(0x1404))) ISRTable(){
//
//          asm("reset"); //reset instruction to prevent runaway code
//          asm("goto %0"::"i"(&_T2Interrupt));  //T2Interrupt's address
//      }
  #endif
#endif


/** DECLARATIONS ***************************************************/

#pragma code


/*********************************************************************/
/*********************************************************************
void main(void)
{
  //******INICIO INICIALIZACION DE SISTEMA***********


TRISB=0x1F;    //Port B as Input
//PullUp=0;

TRISA=0x0F;

cmConf0=1;                      //CMCONbits.CM0=1;      //bit 0 CMCON
cmConf1=0;                      //CMCONbits.CM1=0;      //bit 1 CMCON
cmConf2=0;                      //CMCONbits.CM2=0;      //bit2 CMCON
cmInvC1=1;                      //CMCONbits.C1INV=1;    //bit4 CMCON

//   UserInit:

  //Initialize all of the push buttons

  oldSWstd = SWstd;
  oldCOMPARADOR = COMPARADOR;

  //initialize the variable holding the handle for the last
  // transmission

  lastINTransmission = 0;
  lastOUTTransmission = 0;

  // Initializes USB module SFRs and firmware variables to known states.

          CUENTA=0;

  USBDeviceInit();          //usb_device.c.

// FIN DE INICIALIZACION DE SISTEMA***********


  #if defined(USB_INTERRUPT)
    USBDeviceAttach();
```

```
  #endif


  while(1)
  {
          Keyboard();

  }//end while
}//end main

//********************************************************************************
//********************************************************************************

void Keyboard(void)
{
          //Check if the IN endpoint is not busy, and if it isn't check if we want to send
          //keystroke data to the host.
  if(!HIDTxHandleBusy(lastINTransmission))
  {
    if(COMPARADOR != oldCOMPARADOR)
    {
                                if (COMPARADOR==1)
                                {
                                        switch (CUENTA)
                                        {
                                        case 0:
                                        KEY1=0X05;
                                        KEY3=0x09;              // Ctrl+Alt+F
                                        CUENTA=1;
                                        break;
                                        case 1:
                                        KEY1=0x1;
                                        KEY3=0x08;              // Ctrl+E
                                        CUENTA=2;
                                        break;
                                        case 2:
                                        KEY1=0x01;
                                        KEY3=0x0D;                                      // Ctrl+J
                                        oldCOMPARADOR = COMPARADOR;
                                        CUENTA=0;
                                        break;
                                        default:
                                        break;
                                        }

                        //Load the HID buffer
                        hid_report_in[0] = KEY1;
                        hid_report_in[1] = 0;
                        hid_report_in[2] = KEY3;
                        hid_report_in[3] = 0;
                        hid_report_in[4] = 0;
                        hid_report_in[5] = 0;
                        hid_report_in[6] = 0;
                        hid_report_in[7] = 0;
                        //Send the 8 byte packet over USB to the host.
                        lastINTransmission = HIDTxPacket(HID_EP, (BYTE*)hid_report_in, 0x08);

                                        switch (CUENTA)
                                        {
                                        case 0:
                                        Delay10KTCYx(10);    //DELAY
                                        break;
                                        case 1:
                                        Delay10KTCYx(100);   //DELAY
                                        Delay10KTCYx(100);   //DELAY
```

```
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                break;
                case 2:
                Delay10KTCYx(10);   //DELAY
                break;
                default:
                break;
                }

                return;

        }
        else if (COMPARADOR==0)
        {

                switch (CUENTA)
                {
                case 0:
                KEY1=0X05;
                KEY3=0x09;              // Ctrl+Alt+F
                CUENTA=1;
                break;
                case 1:
                KEY1=0x1;
                KEY3=0x08;              // Ctrl+E
                CUENTA=2;
                break;
                case 2:
                KEY1=0x01;
                KEY3=0x07;                                      // Ctrl+D
                oldCOMPARADOR = COMPARADOR;
                CUENTA=0;
                break;
                default:
                break;
                }

//Load the HID buffer
hid_report_in[0] = KEY1;
hid_report_in[1] = 0;
hid_report_in[2] = KEY3;
hid_report_in[3] = 0;
hid_report_in[4] = 0;
hid_report_in[5] = 0;
hid_report_in[6] = 0;
hid_report_in[7] = 0;
//Send the 8 byte packet over USB to the host.
lastINTransmission = HIDTxPacket(HID_EP, (BYTE*)hid_report_in, 0x08);

                switch (CUENTA)
                {
                case 0:
                Delay10KTCYx(10);   //DELAY
                break;
                case 1:
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
                Delay10KTCYx(100);  //DELAY
```

```
                                        Delay10KTCYx(100);  //DELAY
                                        Delay10KTCYx(100);  //DELAY
                                        Delay10KTCYx(100);  //DELAY
                                        Delay10KTCYx(100);  //DELAY
                                        break;
                                        case 2:
                                        Delay10KTCYx(10);   //DELAY
                                        break;
                                        default:
                                        break;
                                        }

                                        return;
                                }
                                else
                                return;

} else if(COMPARADOR == oldCOMPARADOR)
{
                                        if(oldCOMPARADOR==1)          // && CUENTA==2)
                                        {
                                                                if(SWstdIsPressed())
                                                                {
                                                                switch (PORTB)
                                                                {
                                                                case 0x11:
                                                                KEY2=0x1E;
                                                                break;
                                                                case 0x12:
                                                                KEY2=0x1F;
                                                                break;
                                                                case 0x13:
                                                                KEY2=0x20;
                                                                break;
                                                                case 0x14:
                                                                KEY2=0x21;
                                                                break;
                                                                case 0x15:
                                                                KEY2=0x22;
                                                                break;
                                                                case 0x16:
                                                                KEY2=0x23;
                                                                break;
                                                                case 0x17:
                                                                KEY2=0x24;
                                                                break;
                                                                case 0x18:
                                                                KEY2=0x25;
                                                                break;
                                                                case 0x19:
                                                                KEY2=0x26;
                                                                break;
                                                                case 0x1A:
                                                                KEY2=0x27;
                                                                break;
                                                                case 0x1B:
                                                                KEY2= 0x04;
                                                                break;
                                                                case 0x1C:
                                                                KEY2= 0x05;
                                                                break;
                                                                default:
                                                                break;
                                                                }
                                        //Load the HID buffer
                                        hid_report_in[0] = 0;
```

```
                                    hid_report_in[1] = 0;
                                    hid_report_in[2] = KEY2;                    //envía CARACTER TECLEADO
                                    hid_report_in[3] = 0;
                                    hid_report_in[4] = 0;
                                    hid_report_in[5] = 0;
                                    hid_report_in[6] = 0;
                                    hid_report_in[7] = 0;
                                    //Send the 8 byte packet over USB to the host.
                                    lastINTransmission = HIDTxPacket(HID_EP, (BYTE*)hid_report_in, 0x08);

                                                KEY2=0;
                                                }
                                                else
                                                {
                                    hid_report_in[0] = 0;
                                    hid_report_in[1] = 0;
                                    hid_report_in[2] = 0;
                                    hid_report_in[3] = 0;
                                    hid_report_in[4] = 0;
                                    hid_report_in[5] = 0;
                                    hid_report_in[6] = 0;
                                    hid_report_in[7] = 0;
                                    //Send the 8 byte packet over USB to the host.
                                    lastINTransmission = HIDTxPacket(HID_EP, (BYTE*)hid_report_in, 0x08);
                                                }

                                    }else if(oldCOMPARADOR==0)
                                    {
                                    hid_report_in[0] = 0;
                                    hid_report_in[1] = 0;
                                    hid_report_in[2] = 0;
                                    hid_report_in[3] = 0;
                                    hid_report_in[4] = 0;
                                    hid_report_in[5] = 0;
                                    hid_report_in[6] = 0;
                                    hid_report_in[7] = 0;
                                    //Send the 8 byte packet over USB to the host.
                                    lastINTransmission = HIDTxPacket(HID_EP, (BYTE*)hid_report_in, 0x08);
                                    }
                                    return;
      }//end if
   }
return;
}//end keyboard()



/********************************************************************************
 * Function:       BOOL SWstdIsPressed(void)
********************************************************************************/
BOOL SWstdIsPressed(void)
{
   if(SWstd != oldSWstd)
   {
      oldSWstd = SWstd;             // Save new value
      if(SWstd == 1)                // If pressed
         return TRUE;               // Was pressed
   }//end if
   return FALSE;                    // Was not pressed
}//end SWstdIsPressed
```

ANEXO 2

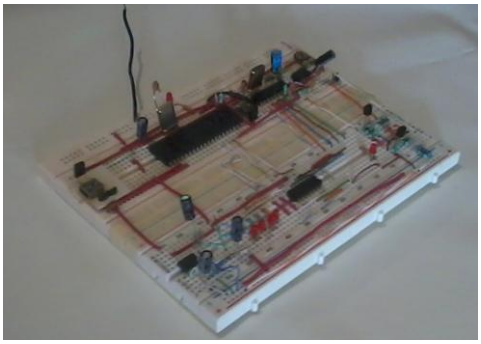
Código VBA para el módulo de comunicaciones

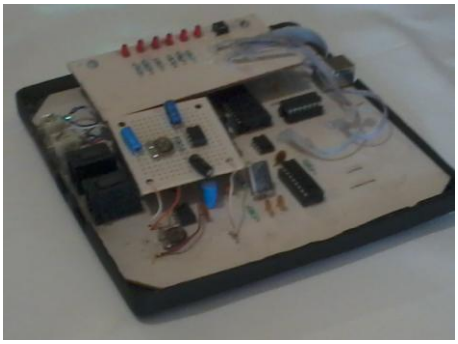
```
Option Compare Database

Public Const PING_TIMEOUT = 400 'default is 200

Public Type ICMP_OPTIONS
   Ttl            As Byte
   Tos            As Byte
   Flags          As Byte
   OptionsSize    As Byte
   OptionsData    As Long
End Type

Public Type ICMP_ECHO_REPLY
   Address        As Long
   status         As Long
   RoundTripTime  As Long
   DataSize       As Integer
   Reserved       As Integer
   DataPointer    As Long
   Options        As ICMP_OPTIONS
   Data           As String * 250
End Type

Public Declare Function IcmpCreateFile Lib "icmp.dll" () As Long

Public Declare Function IcmpCloseHandle Lib "icmp.dll" _
  (ByVal IcmpHandle As Long) As Long

Public Declare Function IcmpSendEcho Lib "icmp.dll" _
  (ByVal IcmpHandle As Long, _
   ByVal DestinationAddress As Long, _
   ByVal RequestData As String, _
   ByVal RequestSize As Integer, _
   ByVal RequestOptions As Long, _
   ReplyBuffer As ICMP_ECHO_REPLY, _
   ByVal ReplySize As Long, _
   ByVal Timeout As Long) As Long

Public Function Ping(szAddress As String, ECHO As ICMP_ECHO_REPLY) As Long

  Dim hPort As Long
  Dim dwAddress As Long
  Dim sDataToSend As String
  Dim iOpt As Long

  sDataToSend = "Echo This"
  dwAddress = AddressStringToLong(szAddress)

  hPort = IcmpCreateFile()

  If IcmpSendEcho(hPort, _
             dwAddress, _
             sDataToSend, _
             Len(sDataToSend), _
             0, _
             ECHO, _
             Len(ECHO), _
             PING_TIMEOUT) Then

      Ping = ECHO.RoundTripTime
```

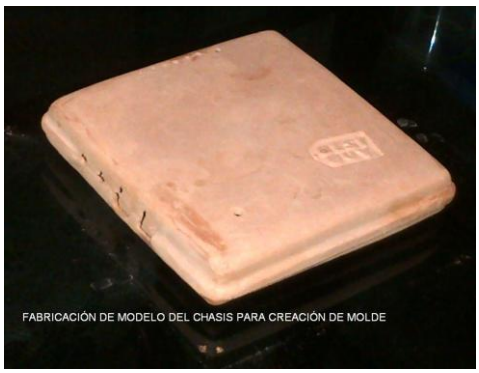
```
  Else: Ping = ECHO.status * -1
  End If

  Call IcmpCloseHandle(hPort)

End Function

Function AddressStringToLong(ByVal tmp As String) As Long

  Dim i As Integer
  Dim parts(1 To 4) As String

  i = 0
  While InStr(tmp, ".") > 0
    i = i + 1
    parts(i) = Mid(tmp, 1, InStr(tmp, ".") - 1)
    tmp = Mid(tmp, InStr(tmp, ".") + 1)
  Wend

  i = i + 1
  parts(i) = tmp

  If i <> 4 Then
    AddressStringToLong = 0
    Exit Function
  End If
  AddressStringToLong = Val("&H" & Right("00" & Hex(parts(4)), 2) & _
                Right("00" & Hex(parts(3)), 2) & _
                Right("00" & Hex(parts(2)), 2) & _
                Right("00" & Hex(parts(1)), 2))

End Function
```

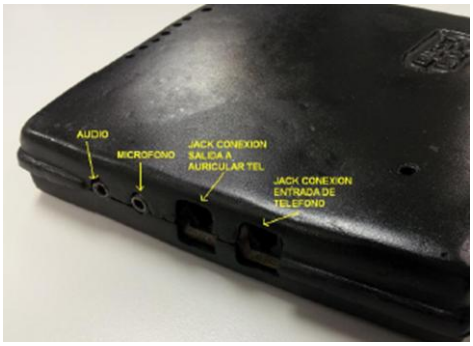
ANEXO 3

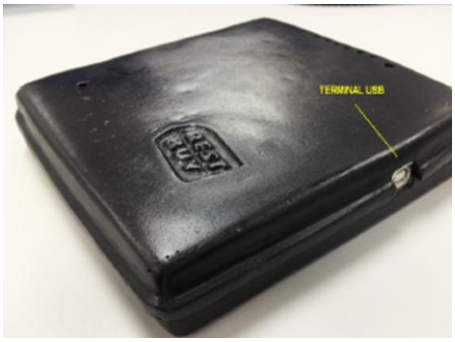
EVIDENCIAS FOTOGRAFICAS DEL PROYECTO

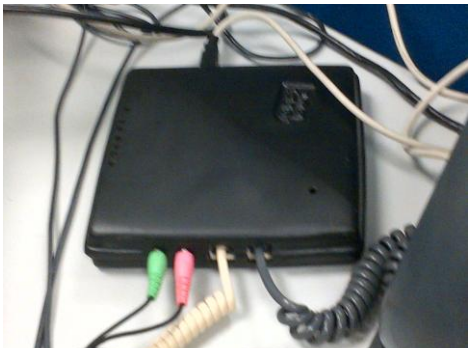












ANEXO 4

RECONOCIMIENTO BRAD ANDERSON LEGACY STOCK DEBIDO AL DESARROLLO DEL DISPOSITIVO NOSY

ANEXO 5

LISTA DE ACRÓNIMOS Y TECNICISMOS INGLESES UTILIZADOS

Acrónimos:

API	Application Programming Interface / Interface para programación de aplicaciones.
ATM	Automatic Teller Machine / Cajero automático
BOR	Brown Out Reset / Reinicio por fallo en alimentación
CCP	Capture – Compare – PWM / Captura – Compara – Modulación de ancho de pulsos
CPU	Central Processing Unit / Unidad central de procesamiento
DTMF	Dual Tone Multi Frecuency / Protocolo de multifrecuencia de doble tono
EAN13	European Article Number 13 / Codificación para código de barras de artículos Europeo Nó 13
EEPROM	Electrically Erasable Programmable Read-Only Memory / Memoria de solo lectura programmable electricamente borrable
GPR	General Proposal Register / Registros de propósito general
ICD/ICSP	In-circuit debugging/In-circuit serial / Depuración en circuito serial
IP	Internet Protocol / Protocolo de Internet
MCLR	Master Clear / Borrado o reinicio maestro
MSDOS	Microsoft Disk Operating System / Sistema Operativo de disco de Microsoft
OEM	Original Equipment Manufacturer / Fabricante de equipos originales
OST	Oscillator Start-up Timer / Temporizador Arranque Oscilador
PC	Personal Computer / Computadora personal
PDF	Portable Document Format / Formato de documento portable
PLL	Phase Locked Loop / Bucle de fase bloqueado (Divisor de frecuencia)
POR	Power On Reset / Reinicio al encender el dispositivo
PWM	Modulación de ancho de pulsos
PWRT	Power-up Timer / Temporizador de encendido
RAM	Random Access Memory / Memoria de acesso aleatorio
SAP	*Systemanalyse und Programmentwicklung* / Sistemas de Análisis y Desarrollo de Programas
SMS	Short Message Service / Servicio de mensajes cortos
SPR	Specific Proposal Register / Registro de propósito específico
TCP	Transmission Control Protocol / Protocolo de control de transmisión
TTL	Transistor-Transistor Logic / Lógica de Transistor-Transistor
UPC	Universal Product Code / Código Universal de producto
USART	Universal Synchronous Asynchronous Receiver/Transmitter
USB	Universal serial Bus / Bus Universal Serial
USB-IF	USB Implementer´s Forum
VBA	Visual Basic Access
WDT	Watch Dog Timer / Temporizador Perro guardián

Tecnicismos ingleses:

Firmware.- Código de programa contenido en la memoria del microcontrolador y que ejecuta cuando le es suministrada energía eléctrica.

Buffer.- En este texto tiene dos significados:

1. Se refiere a un espacio de memoria de datos físico utilizado para la transmisión y/o recepción de datos.
2. En electrónica, se refiere a la etapa o módulo que sirve de intermediaria entre otras dos etapas para que no entren en contacto.

Pull Down.- Se refiere a la conexión de resistores entre la salida tierra o común de un dispositivo lógico.

USB Stack.- Se refiere al set o conjunto de archivos fuente proporcionados por Microchip Inc que contienen todas las utilerías para la implementación de la comunicación USB en sus microcontroladores.

Input Report.- Reporte de datos que se transmite del dispositivo USB al host.

Input Endpoint.- Espacio de memoria de datos reservado para transmitir datos del dispositivo al host.

Output Report.- Reporte de datos que se transmite desde el host al dispositivo USB.

Output Endpoint.- Espacio de memoria de datos reservado para trasmitir datos del host al dispositivo USB.

main(void).- Función principal que según la sintaxis del lenguaje de programación C, representa el inicio o punto de partida para la ejecución de un programa.

Polling.- Operación de consulta constante hacia el dispositivo sin el uso de interrupciones.

BIBLIOGRAFÍA

[1] **USB COMPLETE**
Jan Axelson
Lakeview Research LLC

[2] **ADVANCED PIC MICROCONTROLLER PROJECTS IN C**
Dogan Irahim
Editorial Newnes

[3] **PIC18F2455/2550/4455/4550 Data Sheet**
Microchip Technology Inc.

[4] **MPLAB C18 C COMPILER USER´S GUIDE**
Microchip Technology Inc.

[5] **MPLAB C18 C COMPILER LIBRARIES**
Microchip Technology Inc.

[6] **UNIVERSAL SERIAL BUS: Device Class Definition for Human Interface devices (HID)**
USB Implementers´Forum

[7] **UNIVERSAL SERIAL BUS: HID Usage Tables Versión 1.11**
USB Implementers´Forum

[8] **REDES LOCALES**
José Luis Raya
Editorial Alfaomega – RA-MA

[9] **MICROSOFT ACCESS RUNNING**
John L. Viescas
Editorial McGraw Hill

[10] **MICROSOFT ACCESS DESARROLLO DE SOLUCIONES**
Timothy M. O´Brien, Steve J. Pogge
Editorial McGraw Hill