ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNOLÓGOS.

CONSTRUCCIÓN DE UNA LIBRETA ELECTRÓNICA DE PRÁCTICAS MEDIANTE VISUAL C# CON MICROCONTROLADORES PIC, USANDO EL PUERTO SERIAL RS-232 Y EL PUERTO USB.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO

EN ELECTRÓNICA Y TELECOMUNICACIONES

LUCÍA CRISTINA GÓMEZ ENCARNACIÓN
golucy0@hotmail.com

DIRECTOR: ING ALCIVAR EDUARDO COSTALES GUADALUPE
eduardo.costales@.epn.edu.ec

Quito, Noviembre 2011

DECLARACIÓN

Yo GÓMEZ ENCARNACIÓN LUCÍA CRISTINA, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

GÓMEZ ENCARNACIÓN LUCÍA CRISTINA

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por la Srta. GÓMEZ ENCARNACIÓN LUCÍA CRISTINA, bajo mi supervisión.

Ing. ALCIVAR COSTALES

DIRECTOR DE PROYECTO

AGRADECIMIENTO

A Dios por permitirme llegar a este momento tan especial en mi vida. Por los triunfos y los momentos difíciles que me han enseñado a valorarte cada día más.

A mis amados padres Wilson y Lorgia a quienes siempre llevo en mi corazón por entregarme todo su amor, por enseñarme a levantarme y seguir caminando a pesar de cualquier adversidad, gracias por estar siempre presentes y ser mi ejemplo, guía y fuente de inspiración.

A mis queridos tíos Clarita y su esposo Guillito por brindarme su apoyo incondicional, por depositar su confianza en cada reto que me ha tocado emprender sin dudar de mi inteligencia y capacidad.

A todos mis hermanos y hermanas por darme esa voz de aliento y colocar su granito de arena en mi crecimiento personal.

A mis primos que son como mis hermanos de quienes también aprendí y disfruté lapsos de tiempo inesperado y extraordinario.

A mis amigos y amigas con quienes compartí y disfruté momentos inolvidables, pero sobre todo gracias por su amistad.

Al Ing. Alcívar Costales por todo su tiempo, colaboración y paciencia ya que sin su ayuda no hubiera sido posible la culminación de este proyecto.

A la Escuela Politécnica Nacional y en especial a la ESFOT que me dieron la oportunidad de formar parte de ellas.

Lucía Cristina Gómez Encarnación.

DEDICATORIA

A todos los futuros estudiantes de la carrera Electrónica y Telecomunicaciones para facilitarles y motivarles a desarrollar su potencial creativo en el área de programación y uso de los PICs.

Lucía Cristina Gómez Encarnación.

ÍNDICE DE CONTENIDOS.

ÍNDICE DE GRÁFICOS.

RESUMEN

En 1980 aproximadamente, los fabricantes de circuitos integrados iniciaron la difusión de un nuevo circuito para control, medición e instrumentación al que llamaron microcomputador en un sólo chip o de manera más exacta MICROCONTROLADOR.

Un microcontrolador es un circuito integrado que contiene toda la estructura (arquitectura) de un microcomputador, o sea CPU, RAM, ROM y circuitos de entrada y salida. Los resultados de tipo práctico, que pueden lograrse a partir de estos elementos, son sorprendentes.

Algunos microcontroladores más especializados poseen además convertidores análogos digital, temporizadores, contadores y un sistema para permitir la comunicación alámbrica como inalámbrica.

Se pueden crear muchas aplicaciones con los microcontroladores. Estas aplicaciones de los microcontroladores son ilimitadas (el límite es la imaginación) entre ellas podemos mencionar: sistemas de alarmas, juego de luces, paneles publicitarios, etc. Controles automáticos para la Industria en general. Entre ellos control de motores DC/AC y motores de paso a paso, control de máquinas, control de temperatura, control de tiempo, adquisición de datos mediante sensores, etc.

A partir de aquí el trabajo se dedica a hacer una descripción breve y en pocos párrafos de algunos de los principales microcontroladores del mercado y podemos ver que no hay mucha diferencia entre unas marcas y otras. Tal vez algunas se especialicen más para algunas aplicaciones mientras que otras lo hacen más en el campo del aprendizaje por ser de propósito más general.

El objetivo principal es construir una Libreta Electrónica de Prácticas mediante Visual C# para verificar programas a implementar con PICs de la familia 18F, usando el puerto serial RS232y el puerto USB; de buena presentación y fácil de usar, además que nos permita mostrar diferentes aplicaciones en nuestro proyecto, utilizando una pantalla GLCD y herramientas de software como Visual C # y CCS.

El proyecto que a continuación se presenta consiste en determinar las características principales de los PICs de la familia 18F, estableciendo las diferencias entre el puerto serial RS232 y el puerto USB, y está basado en el principio de mostrar por pantalla los resultados de cada aplicación y la comunicación entre el PIC y los diferentes puertos.

El comportamiento creciente de aplicaciones que usan microprocesadores para control y monitoreo hace que sea de gran importancia el manejo de estos dispositivos y aprovechar al máximo su potencial, es así, que de esta manera ponemos en consideración este trabajo que puede ayudar a comprender el funcionamiento de un PIC y la forma como interactúa entre medios tangibles.

Se utiliza la pantalla GLCD 128*64 para mostrar los resultados, la cual es controlada por el PIC18F4550.

PRESENTACIÓN

El progreso de la electrónica digital y en general toda la electrónica tiene la tendencia a la reducción del hardware que se utiliza, tomando como una alternativa altamente eficiente y rentable la implementación de dispositivos programables como son los microcontroladores los cuales hacen posible la ejecución de aplicaciones de una manera más sencilla.

Los estudiantes siempre hemos requerido de módulos didácticos actuales y novedosos los cuales nos faciliten el trabajo en el Laboratorio de Microprocesadores.

Por lo tanto en este caso hemos requerido más de una vez, de módulos didácticos que nos ayuden a comprobar nuestra programación realizada y obtener con certeza resultados positivos así evitándonos de perder tiempo en la construcción del módulo y a la vez economizar nuestro bolsillo.

La Libreta electrónica de Prácticas contiene 5 aplicaciones pequeñas pero de gran ayuda las que funcionan individualmente o en conjunto, muy fácil de manipular, cada programa de cada aplicación está guardado en el PIC y desde el software Visual C# enviamos la orden para que se ejecute la aplicación deseada.

1. CAPÍTULO I: FUNDAMENTOS TEÓRICOS.

1.1 MICROCONTROLADOR

1.1.1 INTRODUCCIÓN

Un microcontrolador es un sistema microprogramable que se presenta en un circuito integrado de alta escala de integración, es decir, se trata de un ordenador completo en un solo circuito integrado. En su interior se encuentra una CPU, Unidad E/S y memoria interna, normalmente memoria RAM (volátil) para guardar datos y memoria de programa no volátil (EPROM, EEPROM o Flash) donde reside el programa a ejecutar.
La señal de reloj puede generarse internamente o bien mediante elementos externos, una red RC, un cristal de cuarzo o un resonador.

1.1.1.1 Que es un microprocesador

El microprocesador es la parte de la computadora diseñada para llevar acabo o ejecutar los programas. Este viene siendo el cerebro de la computadora, el motor, el corazón de esta máquina. Este ejecuta instrucciones que se le dan a la computadora a muy bajo nivel haciendo operaciones lógicas simples, como sumar, restar, multiplicar y dividir. El microprocesador, o simplemente el micro, es el cerebro del ordenador. Es un chip, un tipo de componente electrónico en cuyo interior existen miles (o millones) de elementos llamados transistores, cuya combinación permite realizar el trabajo que tenga encomendado el chip.

1.1.1.2 Controlador y microcontrolador

Recibe el nombre de controlador el dispositivo que se emplea para el gobierno de uno o varios procesos. Por ejemplo, el controlador que regula el funcionamiento de un horno dispone de un sensor que mide constantemente su temperatura interna y, cuando traspasa los límites prefijados, genera las señales adecuadas que accionan los actuadores que intentan llevar el valor de la temperatura dentro

del rango estipulado. Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta, posteriormente se emplearon los microprocesadores, que se rodeaban con chips de memoria y E/S sobre una tarjeta de circuito impreso.

Recibe el nombre de microcontrolador el dispositivo que contiene todos los elementos del controlador y consiste en un sencillo pero completo computador contenido en el corazón (chip) de un circuito integrado.
Los productos que para su regulación incorporan un microcontrolador disponen de las siguientes ventajas:
Aumento de prestaciones: Mayor control sobre un determinado elemento.
Aumento de la fiabilidad: Al reemplazar el microcontrolador por un elevado número de elementos disminuye el riesgo de averías y se precisan menos ajustes.
Reducción del tamaño en el producto acabado: Disminuye el volumen, la mano de obra y los stocks.
Mayor flexibilidad: Las características de control están programadas por lo que su modificación sólo necesita cambios en el programa de instrucciones.

1.1.1.3 Arquitectura de un microcontrolador

- Procesador o CPU (Unidad Central de Proceso).
- Memoria Central:
- Memoria tipo ROM/EPROM/EEPROM/Flash.
- Memoria de datos de tipo RAM.
- Buses de control, datos y direcciones.
- Líneas de E/S para comunicarse con el exterior.

Recursos auxiliares (temporizadores, Puertas Serie y Paralelo, Conversores Analógico/Digital, Conversores Digital/Analógico, etc.).
Generador de impulsos de reloj (sincroniza el funcionamiento de todo el sistema).
El microcontrolador es en definitiva un circuito integrado que incluye todos los componentes de un computador. Como se observa en la figura 1.1.

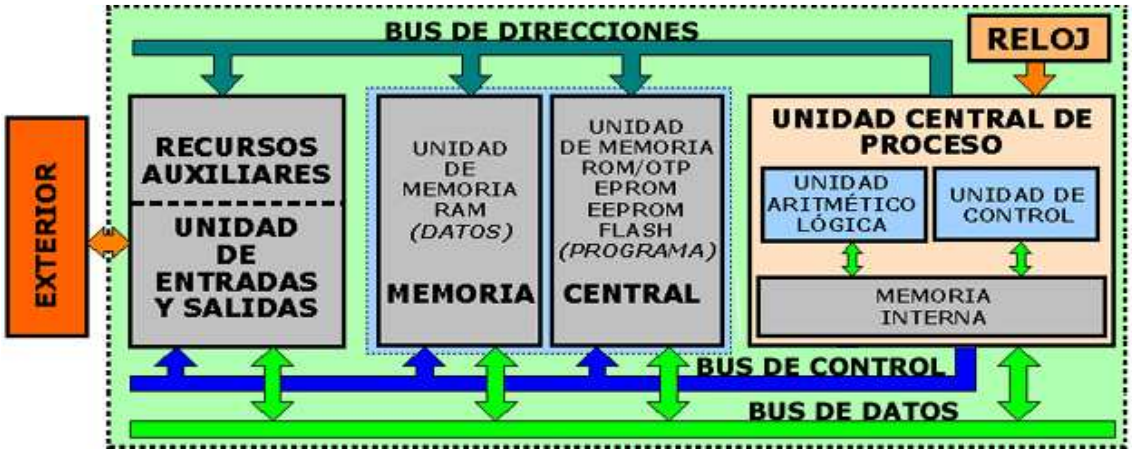


Figura 1.1 Arquitectura de un microcontrolador

1.1.2 DIFERENCIA ENTRE MICROPROCESADOR Y MICROCONTROLADOR.

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (CPU), también llamada procesador, de un computador. La CPU está formada por la Unidad de Control, que interpreta las instrucciones, y la ALU (Unidad Aritmética Lógica), que las ejecuta.

Las patitas de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la Memoria y los Módulos de E/S y configurar un computador implementado por varios circuitos integrados. Se dice que un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine como se observa en la Figura 1.2.

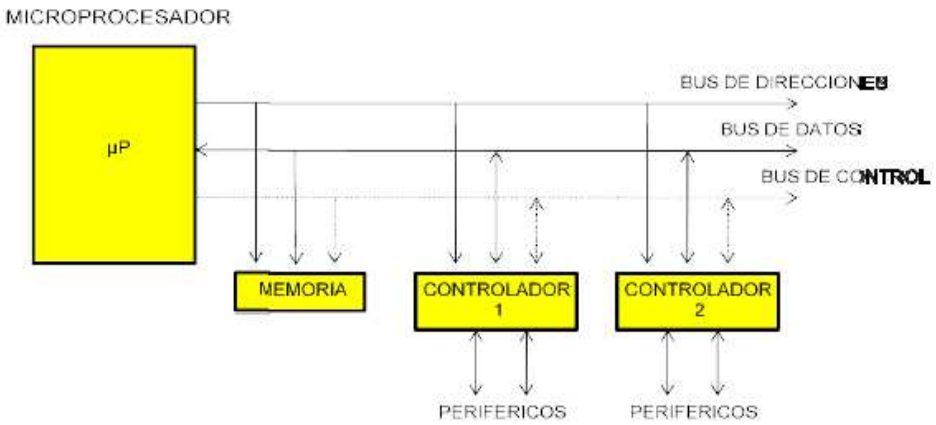


Figura 1.2. Estructura de un sistema abierto basado en un microprocesador.

La disponibilidad de los buses en el exterior permite que se configure a la medida de la aplicación.

Un microcontrolador es un circuito integrado, todas las partes del computador están contenidas en su interior y sólo salen al exterior las líneas que gobiernan los periféricos. Si sólo se dispusiese de un modelo de microcontrolador, éste debería tener muy potenciados todos sus recursos para poderse adaptar a las exigencias de las diferentes aplicaciones. Esta potenciación supondría en muchos casos un despilfarro. Es posible seleccionar la capacidad de las memorias, el número de líneas de E/S, la cantidad y potencia de los elementos auxiliares, la velocidad de funcionamiento, etc. Un aspecto muy destacado del diseño es la selección del microcontrolador a utilizar, en la figura 1.3 se observa el microcontrolador.

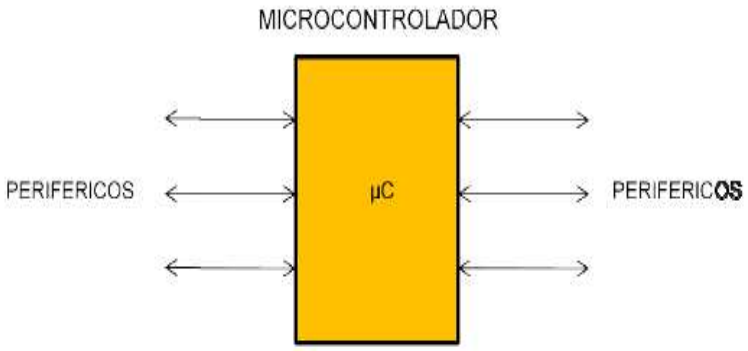


Figura 1.3 El microcontrolador es un sistema cerrado.

1.1.3 HERRAMIENTAS PARA EL DESARROLLO DE APLICACIONES:

Uno de los factores que más importancia tiene a la hora de seleccionar un microcontrolador entre todos los demás es el soporte tanto de software como hardware de que se dispone. Un buen conjunto de herramientas de desarrollo puede ser decisivo en la elección, ya que puede suponer una ayuda inestimable en el desarrollo del proyecto.

Las principales herramientas de ayuda al desarrollo de sistemas basados en microcontroladores son:

- Desarrollo del software:
- Depuración:

1.1.3.1 Desarrollo del software:

Lenguaje Assambler:

El lenguaje ensamblador es un tipo de lenguaje de bajo nivel utilizado para escribir programas informáticos, y constituye la representación más directa del código máquina específico para cada arquitectura de computadoras legible por un programador.

- Programar en lenguaje ensamblador es difícil de aprender, entender, leer, escribir, depurar y mantener, por eso surgió la necesidad de los lenguajes compilados.
- A pesar de perder rendimiento en un proceso de compilación, en la actualidad la mayoría de las computadoras son suficientemente rápidas.
- El lenguaje ensamblador no es portable.
- Programar en lenguaje ensamblador lleva mucho tiempo.
- Los programas hechos en lenguaje ensamblador son generalmente más rápidos. Al programar cuidadosamente en lenguaje ensamblador se pueden crear programas de 5 a 100 veces más rápidos que con lenguajes de alto nivel.
- Los programas hechos en lenguaje ensamblador generalmente ocupan menos espacio. Un buen programa en lenguaje ensamblador puede ocupar casi la mitad de espacio que su contrapartida en lenguaje de alto nivel.
- Con el lenguaje ensamblador se pueden crear segmentos de código imposibles de formar en un lenguaje de alto nivel.

Lenguaje compilador:

La programación en un programa de alto nivel (como el C) permite disminuir el tiempo de desarrollo de un producto. No obstante, si no se programa con cuidado, el código resultante puede ser mucho más ineficiente que el programado en ensamblador. Las versiones más potentes suelen ser muy caras, aunque para los microcontroladores más populares pueden encontrarse versiones “demo” limitadas e incluso compiladores gratuitos.

1.1.3.2 Depuración:

Como los microcontroladores están destinados a controlar dispositivos físicos, será necesario contar con una serie de herramientas que garanticen su correcto funcionamiento cuando son conectados al resto de los circuitos.

- Simulador: son capaces de ejecutar en un PC programas realizados para el microcontrolador. Su gran inconveniente es que es difícil simular la entrada y la salida de datos del microcontrolador y que tampoco cuentan con los posibles ruidos de entradas.
- Placas de evaluación: pequeños sistemas con un microcontrolador ya montado y que suelen conectarse a un PC desde el que se cargan los programas que se ejecutan en el microcontrolador. Suelen incluir visualizadores LCD, teclados, LEDs, fácil acceso a los pines de entrada y salida, etc.
- Emuladores en circuito: dispositivo que se coloca entre el PC anfitrión y el zócalo de la tarjeta de circuitos impresos donde se alojará el microcontrolador definitivo. El programa es ejecutado desde el PC, pero para la tarjeta de aplicaciones es como si lo hiciese el mismo microcontrolador que luego irá en el zócalo. Presenta en pantalla toda la información tal y como sucederá cuando se coloque la cápsula.

1.1.4 APLICACIONES DE LOS MICROCONTROLADORES.

- Cada vez existen más productos que incorporan un microcontrolador con el fin de aumentar sustancialmente sus prestaciones, reducir su tamaño y coste, mejorar su fiabilidad y disminuir el consumo.
- Algunos fabricantes de microcontroladores superan el millón de unidades de un modelo determinado producidas en una semana. Este dato puede dar una idea de la masiva utilización de estos componentes.
- Los microcontroladores están siendo empleados en multitud de sistemas presentes en nuestra vida diaria, como pueden ser juguetes, horno microondas, frigoríficos, televisores, computadoras, impresoras, módems, el sistema de arranque de nuestro coche, etc.

- Y otras aplicaciones con las que seguramente no estaremos tan familiarizados como instrumentación electrónica, control de sistemas en una nave espacial, etc.
- Una aplicación típica podría emplear varios microcontroladores para controlar pequeñas partes del sistema.

1.2 PIC

1.2.1 INTRODUCCIÓN HISTÓRICA

En 1965 GI (Microelectronics Division) formó una división de microelectrónica, destinada a generar las primeras arquitecturas viables de memoria EPROM y EEPROM. GI también creó un microprocesador de 16 bit, denominado CP1600, a principios de los 70. Este fue un microprocesador razonable, pero no particularmente bueno manejando puertos de e/s. Para algunas aplicaciones muy específicas GI diseñó un Controlador de Interface Periférico (PIC) en torno a 1975. Fue diseñado para ser muy rápido, además de ser un controlador de e/s para una máquina de 16 bits pero sin necesitar una gran cantidad de funcionalidades, por lo que su lista de instrucciones fue pequeña.

No es de extrañar que la estructura diseñada en 1975 sea sustancialmente, la arquitectura del actual PIC16C5x. Además, la versión de 1975 fue fabricada con tecnología NMOS y sólo estaba disponible en versiones de ROM de máscara, pero seguía siendo un buen pequeño microcontrolador. El mercado, no obstante, no pensó así y el PIC quedó reducido a ser empleado por grandes fabricantes únicamente.

Durante los 80, GI renovó su apariencia y se reestructuró, centrando su trabajo en sus principales actividades, semiconductores de potencia esencialmente, lo cual siguen haciendo actualmente con bastante éxito. GI Microelectronics Division cambió a GI Microelectronics Inc (una especie de subsidiaria), la cual fue finalmente vendida en 1985 a Venture Capital Investors, incluyendo la fábrica en Chandler, Arizona. La gente de Ventura realizó una profunda revisión de los productos en la compañía, desechando la mayoría de los componentes AY3, AY5 y otra serie de cosas, asumiendo sólo el negocio de los PIC y de las memorias

EEPROM y EPROM. Como parte de esta estrategia, la familia NMOS PIC165x fue rediseñada para emplear algo que la misma compañía fabricaba bastante bien, memoria EPROM. De esta forma nació el concepto de basarse en tecnología CMOS (estructura semiconductora oxido-metal complementaria), OTP (programables una sola vez) y memoria de programación EPROM, naciendo la familia PIC16C5x.

Actualmente Microchip ha realizado un gran número de mejoras a la arquitectura original, adaptándola a las actuales tecnologías y al bajo costo de los semiconductores.

1.2.2 DEFINICIÓN DE PIC.

Los PIC son una familia de microcontroladores programables basados en una arquitectura tipo Harvard y que utilizan un repertorio de instrucciones máquina muy reducido (RISC). Fabricados por Microchip Technology Inc. una empresa líder en microcontroladores junto a Motorola o Intel que se ha abierto un hueco bastante importante en el mercado de los microcontroladores por su reducido coste, su amplia gama y la gran cantidad de información disponible, la figura del Pic se aprecia en la figura 1.4.

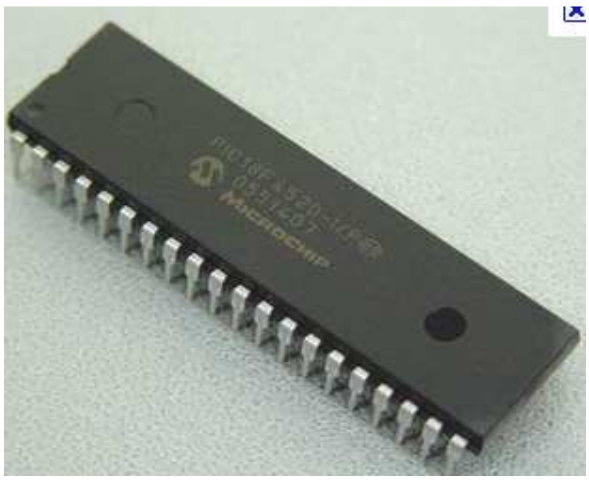


Figura 1.4 PIC.

Los circuitos integrados programables (Programmable Integrated Circuits = PIC) son componentes sumamente útiles en la Electrónica de Consumo. Aun cuando son conocidos desde hace más de veinte años, existen en la actualidad nuevos tipos que cumplen con una serie de requisitos y características sumamente útiles. Como una primera aproximación podemos definir a un PIC como “un chip que me permite obtener un circuito integrado a mi medida”, es decir puedo hacer que el PIC se comporte como un procesador de luminancia o un temporizador o cualquier otro sistema mediante un programa que le grabo en una memoria interna ROM.

RISC: Tanto la industria de los ordenadores como la de los microcontroladores están decantándose hacia la filosofía RISC (Conjunto de Juego de Instrucciones Reducido). En estos procesadores el repertorio de instrucciones máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo. La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

El PIC usa un juego de instrucciones tipo RISC, cuyo número puede variar desde 35 para PICs de gama baja a 70 para los de gama alta. Las instrucciones se clasifican entre las que realizan operaciones entre el acumulador y una constante, entre el acumulador y una posición de memoria, instrucciones de condicionamiento y de salto/retorno, implementación de interrupciones y una para pasar a modo de bajo consumo llamada sleep.

1.2.3 LA ARQUITECTURA DEL PIC ESTA CARACTERIZADA POR LAS SIGUIENTES PRESTACIONES:

- Área de código y de datos separadas (Arquitectura Harvard).
- Un reducido número de instrucciones de largo fijo.
- La mayoría de las instrucciones se ejecutan en un solo ciclo de ejecución (4 ciclos de reloj), con ciclos de único retraso en las bifurcaciones y saltos.
- Un solo acumulador (W), cuyo uso (como operador de origen) es implícito (no está especificado en la instrucción).

- Todas las posiciones de la RAM funcionan como registros de origen y/o de destino de operaciones matemáticas y otras funciones.
- Una pila de hardware para almacenar instrucciones de regreso de funciones.
- Una relativamente pequeña cantidad de espacio de datos direccionable (típicamente, 256 bytes), extensible a través de manipulación de bancos de memoria.
- El espacio de datos está relacionado con el CPU, puertos, y los registros de los periféricos.
- El contador de programa está también relacionado dentro del espacio de datos, y es posible escribir en él (permitiendo saltos indirectos).

A diferencia de la mayoría de otros CPU, no hay distinción entre los espacios de memoria y los espacios de registros, ya que la RAM cumple ambas funciones, y esta es normalmente referida como "archivo de registros" o simplemente, registros.

1.2.4 ARQUITECTURA INTERNA DEL PIC:

1.2.4.1 Arquitectura Von Neumann

Dispone de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control), visto en la figura 1.5.

Figura 1.5 Arquitectura Von Neumann

1.2.4.2 Arquitectura Harvard

Dispone de dos memorias independientes, una que contiene sólo instrucciones, y otra que contiene sólo datos. Ambas disponen de sus respectivos sistemas de

buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias, como se aprecia en la figura 1.6.

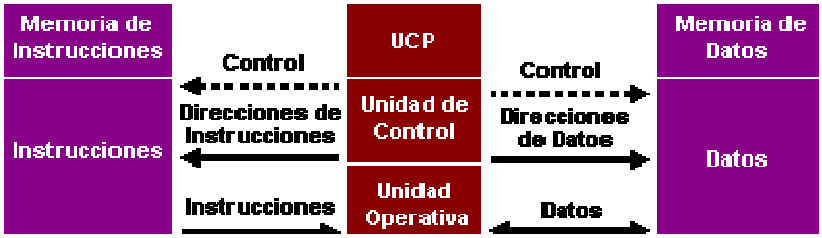


Figura 1.6 Arquitectura Harvard

1.2.4.3 Espacio de datos.

Los microcontroladores PIC tienen una serie de registros que funcionan como una RAM de propósito general. Los registros de propósito específico para los recursos de hardware disponibles dentro del propio chip también están direccionados en la RAM. La direccionabilidad de la memoria varía dependiendo la línea de dispositivos, y todos los dispositivos PIC tienen algún tipo de mecanismo de manipulación de bancos de memoria que pueden ser usados para acceder memoria adicional. Las series más recientes de dispositivos disponen de funciones que pueden cubrir todo el espacio direccionable, independientemente del banco de memoria seleccionado. En los dispositivos anteriores, esto debía lograrse mediante el uso del acumulador.

Para implementar direccionamiento indirecto, un registro de "selección de registro de archivo" (FSR) y de "registro indirecto" (INDF) son usados: Un número de registro es escrito en el FSR, haciendo que las lecturas o escrituras al INDF serán realmente hacia o del registro apuntado por el FSR. La memoria de datos externa no es directamente direccionable excepto en algunos microcontroladores PIC 18 de gran cantidad de pines.

1.2.4.4 Tamaño de palabra

El tamaño de palabra de los microcontroladores PIC es fuente de muchas confusiones. Todos los PICs manejan datos en trozos de 8 bits, con lo que se deberían llamar microcontroladores de 8 bits. Pero a diferencia de la mayoría de CPUs, el PIC usa arquitectura Harvard, por lo que el tamaño de las instrucciones puede ser distinto del de la palabra de datos. De hecho, las diferentes familias de

PICs usan tamaños de instrucción distintos, lo que hace difícil comparar el tamaño del código del PIC con el de otros microcontroladores. Por ejemplo, un microcontrolador tiene 6144 bytes de memoria de programa: para un PIC de 12 bits esto significa 4096 palabras y para uno de 16 bits, 3072 palabras.

1.2.4.5 Programación del PIC

Para transferir el código de un ordenador al PIC normalmente se usa un dispositivo llamado programador. La mayoría de PICs que Microchip distribuye hoy en día incorporan ICSP (In Circuit Serial Programming, programación serie incorporada) o LVP (Low Voltage Programming, programación a bajo voltaje), lo que permite programar el PIC directamente en el circuito destino. Para la ICSP se usan los pines RB6 y RB7 (En algunos modelos pueden usarse otros pines como el GP0 y GP1 o el RA0 y RA1) como reloj y datos y el MCLR para activar el modo programación aplicando un voltaje de 13 voltios. Existen muchos programadores de PICs, desde los más simples que dejan al software los detalles de comunicaciones, a los más complejos, que pueden verificar el dispositivo a diversas tensiones de alimentación e implementan en hardware casi todas las funcionalidades. Muchos de estos programadores complejos incluyen ellos mismos PICs pre programado como interfaz para enviar las órdenes al PIC que se desea programar. El software de programación puede ser el ICprog, muy común entre la gente que utiliza este tipo de microcontroladores. Entornos de programación basados en intérpretes BASIC ponen al alcance de cualquier proyecto que parecieran ser ambiciosos.

1.2.5 CARACTERÍSTICAS DE LOS PICs

- Es ideal para baja potencia en el orden de los nano vatios y aplicaciones de conectividad que se benefician de la disponibilidad de tres puertos seriales: FS-USB (12 Mbit/s), I ² C, SPI (hasta 10 Mbit/s) y uno asincrónico (LIN capaz) de puerto serie (EUSART).
- Tiene grandes cantidades de memoria RAM para el almacenamiento en búfer y la memoria del programa FLASH mejorada; le hacen ideal para aplicaciones de control integrado y seguimiento que requiere conexión

periódica con una computadora personal a través de USB para los datos de carga y descarga o actualización del firmware.

- Mientras esté en funcionamiento hasta 48 MHz, el software y hardware del PIC18F2550 es también en su mayoría compatible con el PIC16C745 USB de baja velocidad.

1.2.5.1 Los PICs actuales vienen con una amplia gama de mejoras hardware incorporados:

- Núcleos de CPU de 8/16 bits con Arquitectura Harvard modificada
- Memoria Flash y ROM disponible desde 256 bytes a 256 kilobytes
- Puertos de E/S (típicamente 0 a 5,5 voltios)
- Temporizadores de 8/16 bits
- Tecnología Nano watt para modos de control de energía
- Periféricos serie síncronos y asíncronos: USART, AUSART, EUSART
- Conversores analógico/digital de 8-10-12 bits
- Comparadores de tensión
- Módulos de captura y comparación PWM
- Controladores LCD
- Periférico MSSP para comunicaciones I²C, SPI, y I²S
- Memoria EEPROM interna con duración de hasta un millón de ciclos de lectura/escritura
- Periféricos de control de motores
- Soporte de interfaz USB
- Soporte de controlador Ethernet
- Soporte de controlador CAN
- Soporte de controlador LIN
- Soporte de controlador Irda

1.2.6 VENTAJAS DE LOS PICs.

- Eficiencia de código.-Permiten una gran compactación de los programas.
- Rapidez de ejecución.
- RISC.- Juego de instrucciones reducido para un fácil aprendizaje.

- Compatibilidad de pines y códigos entre dispositivos de la misma familia.
- Gran variedad de versiones en distintos encapsulados.
- Herramientas de desarrollo tanto en software como en hardware a un bajo costo.

1.3 VISUAL C#

1.3.1 DEFINICIÓN DE VISUAL C#

C# (pronunciado si Sharp en inglés) es un lenguaje orientado a objetos elegante y con seguridad de tipos que permite a los desarrolladores crear una amplia gama de aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Puede utilizar este lenguaje para crear aplicaciones cliente para Windows tradicionales, servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de base de datos, y muchas tareas más. Microsoft Visual C# 2005 proporciona un editor de código avanzado, diseñadores de interfaz de usuario prácticos, un depurador integrado y muchas otras herramientas para facilitar un rápido desarrollo de la aplicación basado en la versión 2.0 del lenguaje C# y en .NET Framework.

Visual Studio admite Visual C# con un editor de código completo, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz y fácil de usar, además de otras herramientas. La biblioteca de clases .NET Framework ofrece acceso a una amplia gama de servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa.

1.3.2 LENGUAJE C#

La sintaxis de C# es muy expresiva, aunque cuenta con menos de 90 palabras clave; también es sencilla y fácil de aprender. La sintaxis de C# basada en signos de llave podrá ser reconocida inmediatamente por cualquier persona familiarizada con C, C++ o Java. Los desarrolladores que conocen cualquiera de estos lenguajes pueden empezar a trabajar de forma productiva en C# en un plazo muy breve. La sintaxis de C# simplifica muchas de las complejidades de C++ y, a la vez, ofrece funciones eficaces tales como tipos de valores que aceptan valores

NULL, enumeraciones, delegados, métodos anónimos y acceso directo a memoria, que no se encuentran en Java. C# también admite métodos y tipos genéricos, que proporcionan mayor rendimiento y seguridad de tipos, e iteradores, que permiten a los implementadores de clases de colección definir comportamientos de iteración personalizados que el código de cliente puede utilizar fácilmente.

Como lenguaje orientado a objetos, C# admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método Main que es el punto de entrada de la aplicación, se encapsulan dentro de definiciones de clase. Una clase puede heredar directamente de una clase primaria, pero puede implementar cualquier número de interfaces. Los métodos que reemplazan a los métodos virtuales en una clase primaria requieren la palabra clave override como medio para evitar redefiniciones accidentales. En C#, una estructura es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite la herencia.

Además de estos principios básicos orientados a objetos, C# facilita el desarrollo de componentes de software a través de varias construcciones de lenguaje innovadoras, entre las que se incluyen:

- Firmas de métodos encapsulados denominadas delegados, que permiten notificaciones de eventos con seguridad de tipos.
- Propiedades, que actúan como descriptores de acceso para variables miembro privadas.
- Atributos, que proporcionan metadatos declarativos sobre tipos en tiempo de ejecución.
- Comentarios en línea de documentación XML.

Si necesita interactuar con otro software de Windows, como objetos COM o archivos DLL nativos de Win32, podrá hacerlo en C# mediante un proceso denominado "interoperabilidad". La interoperabilidad permite que los programas de C# realicen prácticamente lo mismo que una aplicación de C++ nativa. C# admite incluso el uso de punteros y el concepto de código "no seguro" en los casos en que el acceso directo a la memoria es absolutamente crítico.

El proceso de generación de C# es simple en comparación con el de C y C++, y es más flexible que en Java. No hay archivos de encabezado independientes, ni

se requiere que los métodos y los tipos se declaren en un orden determinado. Un archivo de código fuente de C# puede definir cualquier número de clases, estructuras, interfaces y eventos.

1.3.2.1 A continuación se enumeran otros recursos de Visual C#:

- Para disponer de una introducción general al lenguaje, vea Especificación del lenguaje C#.
- Para obtener información detallada sobre aspectos concretos del lenguaje C#, vea Referencia de C#.
- Para tener una comparación entre la sintaxis de C# y las de Java y C++, vea Lenguaje de programación C# para desarrolladores de Java y C# para los desarrolladores de C++.

1.3.3 ARQUITECTURA DE LA PLATAFORMA.

Los programas de C# se ejecutan en .NET Framework, un componente que forma parte de Windows y que incluye un sistema de ejecución virtual denominado Common Language Runtime (CLR) y un conjunto unificado de bibliotecas de clases. CLR es la implementación comercial de Microsoft de Common Language Infrastructure (CLI), norma internacional que constituye la base para crear entornos de ejecución y desarrollo en los que los lenguajes y las bibliotecas trabajan juntos sin problemas.

El código fuente escrito en C# se compila en un lenguaje intermedio (IL) conforme con la especificación CLI. El código de lenguaje intermedio, junto con recursos tales como mapas de bits y cadenas, se almacena en disco en un archivo ejecutable denominado ensamblado, cuya extensión es .exe o .dll generalmente. Un ensamblado contiene un manifiesto que ofrece información sobre los tipos, la versión, la referencia cultural y los requisitos de seguridad del ensamblado.

Cuando se ejecuta un programa de C#, el ensamblado se carga en CLR, con lo que se pueden realizar diversas acciones en función de la información del manifiesto. A continuación, si se cumplen los requisitos de seguridad, CLR realiza una compilación Just In Time (JIT) para convertir el código de lenguaje intermedio

en instrucciones máquinas nativas. CLR también proporciona otros servicios relacionados con la recolección automática de elementos no utilizados, el control de excepciones y la administración de recursos. El código ejecutado por CLR se denomina algunas veces "código administrado", en contraposición al "código no administrado" que se compila en lenguaje máquina nativo destinado a un sistema específico.

1.4. MAX 232

El MAX232 es un transmisor/receptor que incluye un generador de voltaje capacitivo que suministra niveles de voltaje del estándar EIA – 232 desde solo 5V, de esta manera es un interfaz de conexión que cambia los niveles lógicos TTL a los estándares RS232 cuando se realiza una transmisión a la PC desde el circuito en comunicación y cambia los niveles RS- 232 a TTL, cuando tiene una recepción desde la PC. Además este circuito integrado lleva internamente 2 conversores con lo que podremos manejar 4 señales del pórtico serie del PC. El MAX232 es un circuito integrado que convierte los niveles de las líneas de un puerto serie RS232 a niveles TTL y viceversa. Lo interesante es que sólo necesita una alimentación de 5V, ya que genera internamente algunas tensiones que son necesarias para el estándar RS232. Otros integrados que manejan las líneas RS232 requieren dos voltajes, +12V y -12V. En la figura 1.7 se observan la distribución de pines y el elemento físico del MAX232.

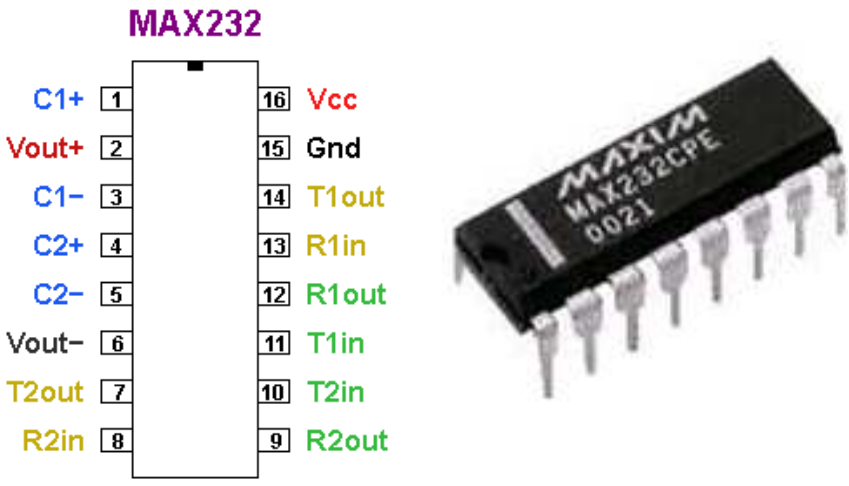


Figura 1.7 Encapsulado del max232

El MAX232 soluciona la conexión necesaria para lograr comunicación entre el puerto serie de una PC y cualquier otro circuito con funcionamiento en base a señales de nivel TTL/CMOS. En la figura 1.7 se observa el encapsulado del max232

1.4.1 FUNCIONAMIENTO:

El circuito integrado lleva internamente 2 conversores de nivel de TTL a RS232 y otros 2 de RS232 a TTL con lo que en total podemos manejar 4 señales del puerto serie del PC, por lo general las más usadas son; TX, RX, RTS, CTS, estas dos últimas son las usadas para el protocolo handshaking pero no es imprescindible su uso. Para que el max232 funcione correctamente debemos poner unos condensadores externos.

Usos: Este integrado es usado para comunicar un microcontrolador o sistema digital con un PC o sistema basado en el bus serie RS232.

1.4.2 CARACTERÍSTICAS DEL MAX232:

- Vcc: de 4,5v a 5,5v.
- Consumo: 4 mA. (15 mA. con carga a la salida de 3 K ohm).
- Entradas compatibles TTL y CMOS.
- Tensión de entrada máxima RS232: +/- 30v.
- Tensión de Salida RS232: +/- 15v.

La tensión de salida típica es de +/-8v con carga nominal de 5 K ohm en RS-232.

- Resistencia entrada RS232: 5 K ohm (a masa).
- Resistencia entrada TTL/CMOS: 400 K ohm (a positivo).
- Las entradas se pueden dejar al aire.

Entrada TTL al aire, se considera un "0" al invertirse en la salida. Entrada RS232 al aire, se considera un "1" al invertirse en la salida.

- Salidas cortocircuitables continuamente:

- Salida RS232: +/- 22 mA. Salida TTL/CMOS: a masa -10 mA., a positivo +30 mA.
- Data Rate: 200 Kbps (mín 116 Kbps).

1.5 PUERTO SERIAL RS- 232

1.5.1 DEFINICIÓN

El puerto serie RS−232 es el que se emplea en las computadoras, PC, módems, conmutadores e impresoras y tiene sus inicios en los años 60´s por la EIA (Electronics Industries Association de los EE.UU), este fue creado para ofrecer una conexión entre aparatos que requieren comunicación de Datos.

Los ordenadores se conectan con cualquier equipo periférico, a través de sus puertos paralelo o serie, o los más recientes como el USB (Universal Serial Bus), es usado aun el puerto serie RS−232 por ser un estándar impuesto en todos los equipos informáticos.

En un ordenador puede haber varios puertos series, a los que normalmente se les denomina COM 1, COM 2, COM 3 (muchas veces los puertos serie a partir del COM 2 se denominan puertos virtuales o son debidos a ampliaciones de los puertos por tarjetas controladoras del tipo PCI), etcétera, por defecto el COM 1 suele pertenecer al ratón usando éste el IRQ 4(canal de interrupción), aunque también es posible encontrarle en el COM 2, así que lo normal es encontrarnos libre el puerto serie del COM 2 utilizando el IRQ 3.

Los equipos terminales de datos (conmutadores, PC, impresoras, etc.), envían señales en 0´s y 1´s lógicos binarios, que el módem debe convertir a señales analógicas y enviarlas por la línea telefónica o canal de comunicación pero también es posible que se comuniquen siempre en digital.

Éste interface o puerto RS−232 trabaja entre +12 voltios y −12 voltios, de manera que un cero lógico es cuando la terminal esté entre +9 y +12 voltios, y un uno lógico cuando esté entre −9 y −12 voltios de manera que un puerto serie que no

está transmitiendo, mantiene el terminal de transmisión en un 1 lógico es decir entre −9 y −12 volts.

Los conectores estándar RS−232 sean estos hembras o machos, son el DB−25 y el DB−9.

1.5.2 LA ESTRUCTURA DE DATOS DEL PUERTO SERIE RS−232.

La comunicación de datos en un puerto serie, se usa normalmente para efectuar comunicaciones asíncronas, ósea sin tiempo preestablecido para iniciarse. Los datos llegan en paquetes de información, normalmente cada paquete es de 8 bits=1 byte, algunos equipos envían carácter por carácter, otros guardan muchos caracteres en la memoria y cuando les toca enviarlos los envían uno tras otro.

Uno de los parámetros más importantes en la comunicación serie, es la velocidad con la que los datos se transmiten, para el caso del RS−232, pueden transmitir de los 300 Baudios (1 Baudio=1 bit/seg) hasta 115,200 Baudios, la velocidad depende de los equipos conectados en el puerto serie y la calidad y longitud de los cables. Otro de los parámetros importantes es el bit de inicio que le indica al puerto receptor que va a llegar un byte de información.

El RS-232 consiste en un conector de 9 pines DB-9 como se muestra en la figura 1.8, es más barato e incluso más extendido para cierto tipo de periféricos (como el ratón serie del PC). Las señales con las que trabaja este puerto serie son digitales, de +12V (0 lógico) y -12V (1 lógico), para la entrada y salida de datos, a la inversa en las señales de control. El estado de reposo en la entrada y salida de datos es -12V. Dependiendo de la velocidad de transmisión empleada, es posible tener cables de hasta 15 metros.

Cada pin puede ser de entrada o de salida, teniendo una función específica cada uno de ellos. Las funciones más importantes son:

Pin	Función
TXD	(Transmitir Datos)
RXD	(Recibir Datos)
DTR	(Terminal de Datos Listo)
DSR	(Equipo de Datos Listo)
RTS	(Solicitud de Envío)
CTS	(Libre para Envío)
DCD	(Detección de Portadora)

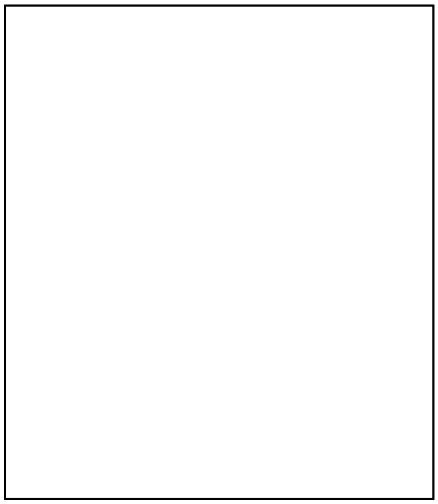


Figura 1.8 Conector DB-9

1.5.3 EL PUERTO SERIE RS-232 EN EL PC.

El ordenador controla el puerto serie mediante un circuito integrado específico, llamado UART (Transmisor-Receptor-Asíncrono Universal). Normalmente se utilizan los siguientes modelos de este chip: 8250 (bastante antiguo, con fallos, solo llega a 9600 baudios), 16450 (versión corregida del 8250, llega hasta 115.200 baudios) y 16550A (con buffers de E/S). A partir de la gama Pentium, la circuitería UART de las placa base son todas de alta velocidad, es decir UART 16550A. De hecho, la mayoría de los módems conectables a puerto serie necesitan dicho tipo de UART, incluso algunos juegos para jugar en red a través del puerto serie necesitan de este tipo de puerto serie.

Mediante los puertos de E/S se pueden intercambiar datos, mientras que las IRQ producen una interrupción para indicar a la CPU que ha ocurrido un evento (por ejemplo, que ha llegado un dato, o que ha cambiado el estado de algunas señales de entrada). La CPU debe responder a estas interrupciones lo más rápido posible, para que dé tiempo a recoger el dato antes de que el siguiente lo sobrescriba. Sin embargo, las UART 16550A incluyen unos buffers de tipo FIFO, dos de 16 bytes (para recepción y transmisión), donde se pueden guardar varios datos antes de que la CPU los recoja. Esto también disminuye el número de interrupciones por segundo generadas por el puerto serie.

El RS-232 puede transmitir los datos en grupos de 5, 6, 7 u 8 bits, a unas velocidades determinadas (normalmente, 9600 bits por segundo o más). Después de la transmisión de los datos, le sigue un bit opcional de paridad (indica si el número de bits transmitidos es par o impar, para detectar fallos), y después 1 o 2 bits de parada. Normalmente, el protocolo utilizado es el 8N1 (que significa, 8 bits de datos, sin paridad y con 1 bit de parada).

Una vez que ha comenzado la transmisión de un dato, los bits tienen que llegar uno detrás de otro a una velocidad constante y en determinados instantes de tiempo. Por eso se dice que el RS-232 es asíncrono por carácter y síncrono por bit. Los pines que portan los datos son RXD y TXD. Las demás se encargan de otros trabajos: DTR indica que el ordenador está encendido, DSR que el aparato conectado a dicho puerto está encendido, RTS que el ordenador puede recibir datos (porque no está ocupado), CTS que el aparato conectado puede recibir datos, y DCD detecta que existe una comunicación o sea presencia de datos.

1.6 PUERTO USB

1.6.1 INTRODUCCIÓN.

Hoy día resulta muy interesante observar cómo los avances tecnológicos nos sorprenden por la evolución tan rápida que presentan y algo que gusta es que cada vez son más fáciles de usar para cualquier persona, es decir, se están volviendo muy amigables y no se necesita ser un experto para poder comprender su funcionamiento, usarlos o instalarlos, este es el caso de Universal Serial Bus, mejor conocido como USB.

1.6.2 HISTORIA

En un principio teníamos la interfaz serie y paralelo, pero era necesario unificar todos los conectores creando uno más sencillo y de mayores prestaciones. Así nació el USB (Universal Serial Bus) con una velocidad de 12Mb/seg. Y como su evolución, USB 2.0, apodado USB de alta velocidad, con velocidades en este momento de hasta 480 Mb/seg, es decir, 40 veces más rápido que las conexiones mediante cables USB 1.1.

USB es una nueva arquitectura de bus o un nuevo tipo de bus desarrollado por un grupo de siete empresas que forman parte de los avances plug-and-play que permite instalar periféricos sin tener que abrir la máquina para instalarle el hardware, es decir, basta con que se conecte dicho periférico en la parte posterior del computador y listo.

1.6.3 USB

Significa "Universal Serial Bus", su traducción al español es línea serial universal de transporte de datos. Es un conector rectangular de 4 terminales que permite la transmisión de datos entre una gran gama de dispositivos externos (periféricos) con la computadora; por ello es considerado puerto; mientras que la definición de la Real Academia Española de la lengua es "toma de conexión universal de uso frecuente en las computadoras".

USB es una interface plug y play entre la PC y ciertos dispositivos tales como teclados, mouses, scanners, impresoras, módems, placas de sonido, cámaras, etc.

Una característica importante es que permite a los dispositivos trabajar a velocidades mayores, en promedio a unos 12 Mbps, esto es más o menos de 3 a 5 veces más rápido que un dispositivo de puerto paralelo y de 20 a 40 veces más rápido que un dispositivo de puerto serial.

1.6.4 CÓMO FUNCIONA EL USB.

Trabaja como interfaz para transmisión de datos y distribución de energía, que ha sido introducida en el mercado de PCs y periféricos para mejorar las lentas interfaces serie (RS-232) y paralelo. Esta interfaz de 4 hilos, 12 Mbps y "plug and play", distribuye 5V para alimentación, transmite datos y está siendo adoptada rápidamente por la industria informática.

Es un bus basado en el paso de un testigo, semejante a otros buses como los de las redes locales en anillo con paso de testigo y las redes FDDI. El controlador USB distribuye testigos por el bus. El dispositivo cuya dirección coincide con la que porta el testigo responde aceptando o enviando datos al controlador. Este también gestiona la distribución de energía a los periféricos que lo requieran.

Emplea una topología de estrellas apiladas que permite el funcionamiento simultáneo de 127 dispositivos a la vez. Esta topología permite a muchos dispositivos conectarse a un único bus lógico sin que los dispositivos que se encuentran más abajo en la pirámide sufran retardo. A diferencia de otras arquitecturas, USB no es un bus de almacenamiento y envío, de forma que no se produce retardo en el envío de un paquete de datos hacia capas inferiores.

1.6.5 COMPONENTES DELUSB.

El USB emplea una topología de estrellas apiladas, en la raíz o vértice de las capas está el controlador o host que controla todo el tráfico que circula por el bus, esto se aprecia en la figura 1.9, y consta de tres componentes:

- Controlador
- Hubs o Concentradores
- Periféricos

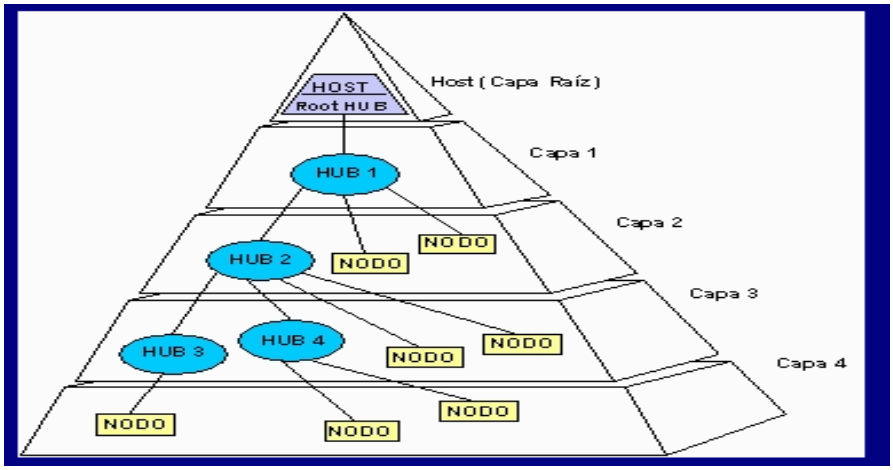


Figura 1.9 Componentes del USB

1.6.5.1 Controlador.

Reside dentro del PC y es responsable de las comunicaciones entre los periféricos USB y la CPU del PC. Es también responsable de la admisión de los periféricos dentro del bus, tanto si se detecta una conexión como una desconexión. Para cada periférico añadido, el controlador determina su tipo y le asigna una dirección lógica para utilizarla siempre en las comunicaciones con el mismo. Si se producen errores durante la conexión, el controlador lo comunica a la CPU, que, a su vez, lo transmite al usuario. Una vez que se ha producido la

conexión correctamente, el controlador asigna al periférico los recursos del sistema que éste precise para su funcionamiento. El controlador también es responsable del control de flujo de datos entre el periférico y la CPU.

1.6.5.2 Hub o Concentradores.

Son distribuidores inteligentes de datos y alimentación, y hacen posible la conexión a un único puerto USB de 127 dispositivos. De una forma selectiva reparten datos y alimentación hacia sus puertas descendentes y permiten la comunicación hacia su puerta de retorno o ascendente. Un hub de 4 puertos, por ejemplo, acepta datos del PC para un periférico por su puerta de retorno o ascendente y los distribuye a las 4 puertas descendentes si fuera necesario. En la figura 1.10 se observa como los hubs proporcionan conectividad a una serie de dispositivos periféricos.

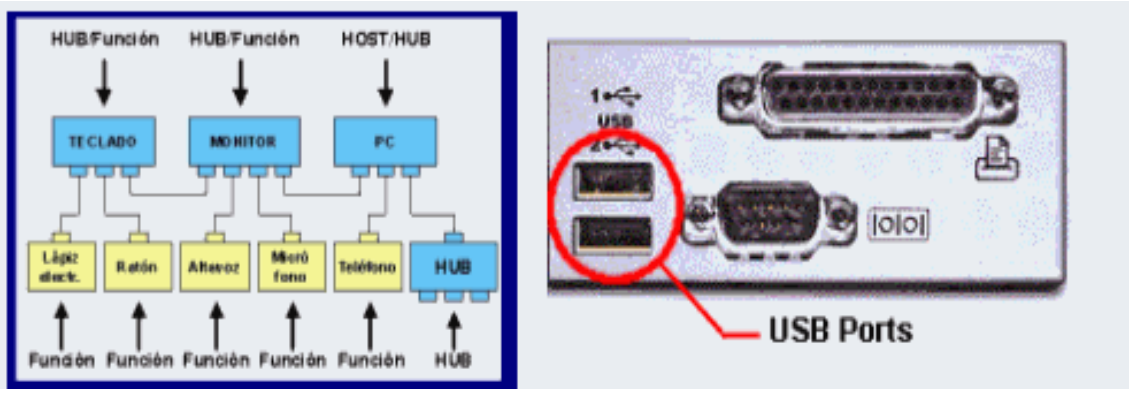


Figura 1.10 Hub y puertos USB

1.6.5.3 Periféricos

USB soporta periféricos de baja y media velocidad. Empleando dos velocidades para la transmisión de datos de 1.5 y 12 Mbps se consigue una utilización más eficiente de sus recursos. Los periféricos de baja velocidad tales como teclados, ratones, joysticks, y otros periféricos para juegos, no requieren 12 Mbps. Empleando para ellos 1,5 Mbps, se puede dedicar más recursos del sistema a periféricos tales como monitores, impresoras, módems, scanners, equipos de audio, que precisan de velocidades más altas para transmitir mayor volumen de datos cuya dependencia temporal es más estricta. En la figura 1.11 podemos observar a varios dispositivos que funcionan mediante el puerto USB.

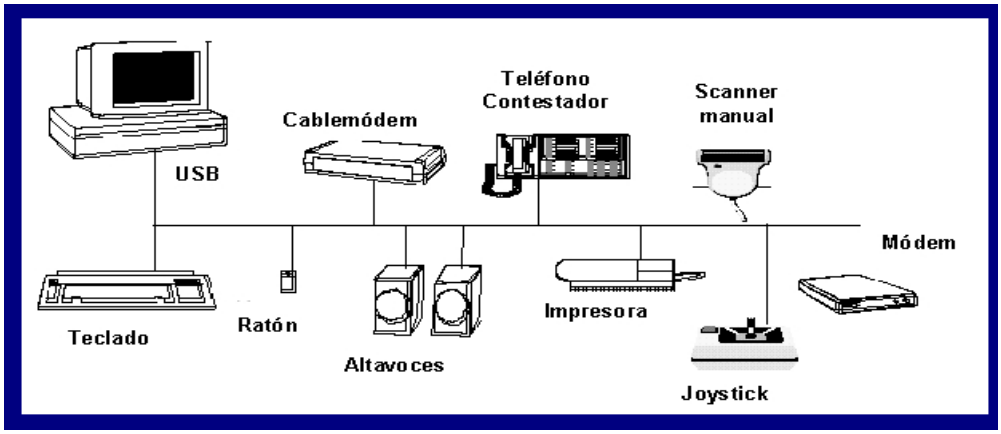


Figura 1.11 Dispositivos periféricos conectados al USB

1.6.6 DIAGRAMA DE CAPAS.

En el diagrama de capas de la figura 1.12 podemos ver cómo fluye la información entre las diferentes capas a nivel real y a nivel lógico.

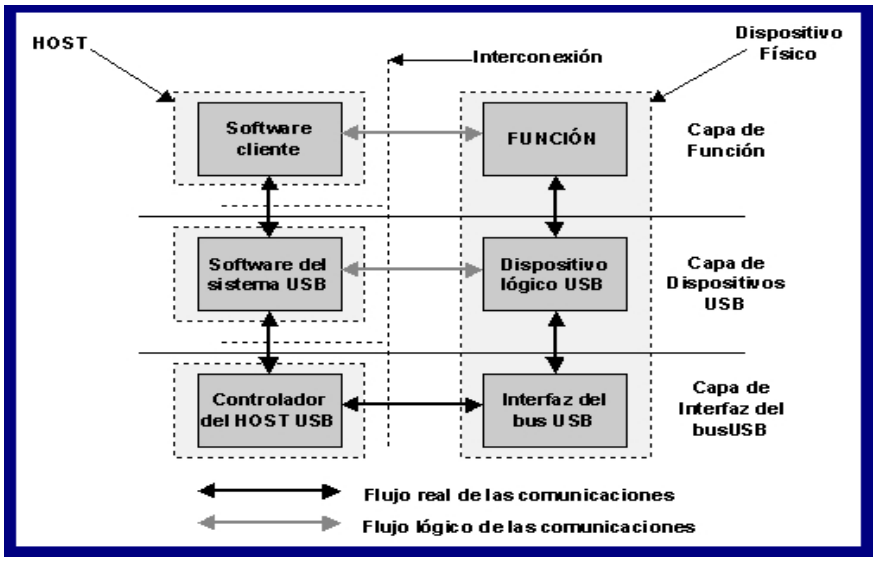


Figura 1.12 Diagrama de capas

El software cliente se ejecuta en el host y corresponde a un dispositivo USB; se suministra con el sistema operativo o con el dispositivo USB. El software del sistema USB, es el que soporta USB en un determinado sistema operativo y se suministra con el sistema operativo independientemente de los dispositivos USB o del software cliente.

El controlador anfitrión USB está constituido por el hardware y el software que permite a los dispositivos USB ser conectados al anfitrión, la conexión entre un

host y un dispositivo requiere la interacción entre las capas. La capa de interfaz de bus USB proporciona la conexión física entre el host y el dispositivo. La capa de dispositivo USB es la que permite que el software del sistema USB realice operaciones genéricas USB con el dispositivo.

La capa de función proporciona capacidades adicionales al host vía una adecuada capa de software cliente. Las capas de función y dispositivos USB tienen cada una de ellas una visión de la comunicación lógica dentro de su nivel, aunque la comunicación entre ellas se hace realmente por la capa de interfaz de bus USB.

1.7 SENSOR.

1.7.1 INTRODUCCIÓN.

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser por ejemplo: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, pH, etc. Un sensor se diferencia de un transductor en que el sensor está siempre en contacto con la variable de instrumentación con lo que puede decirse también que es un dispositivo que aprovecha una de sus propiedades con el fin de adaptar la señal que mide para que pueda interpretar otro dispositivo. Áreas de aplicación de los sensores: Industria automotriz, Industria aeroespacial, Medicina, Industria de manufactura, Robótica, etc. En la figura 1.13 se observa varias clases de sensores.

Figura 1.13 Clases de sensores.

1.7.2 CLASES DE SENSORES.

1.7.2.1 Sensores de presión y fuerza

Son pequeños, ofrecen una excelente repetitividad y una alta precisión y fiabilidad bajo condiciones ambientales variables. Presentan unas características operativas constantes en todas las unidades y una intercambiabilidad sin recalibración.

1.7.2.2 Sensores de humedad.

Están configurados con circuitos integrados que proporcionan una señal acondicionada, contienen un elemento sensible capacitivo en base de polímeros que interacciona con electrodos de platino.

1.7.2.3 Sensores de Movimientos.

Son importantes en robótica, ya que nos da información sobre las evoluciones de las distintas partes que forman el robot, podemos controlar con un grado de precisión elevada la evolución del robot en su entorno de trabajo.

1.7.2.4 Sensores magnéticos.

Ofrecen una alta sensibilidad. Entre las aplicaciones se incluyen brújulas, control remoto de vehículos, detección de vehículos, realidad virtual, sensores de posición, sistemas de seguridad e instrumentación médica.

1.7.2.5 Sensores de temperatura.

Consisten en una fina película de resistencia variable con la temperatura y están calibrados por láser para una mayor precisión e intercambiabilidad. Las salidas lineales son estables y rápidas.

1.7.3 CARACTERÍSTICAS DE LOS SENSORES.

- Rango de medida: dominio en la magnitud medida en el que puede aplicarse el sensor.
- Precisión: es el error de medida máximo esperado.
- Offset o desviación de cero: es valor de la variable de salida cuando la variable de entrada es nula. Si el rango de medida no llega a valores nulos

de la variable de entrada, se establece otro punto de referencia para definir el offset.

- Linealidad o correlación lineal.
- Sensibilidad de un sensor: relación entre la variación de la magnitud de salida y la variación de la magnitud de entrada.
- Resolución: mínima variación de la magnitud de entrada que puede apreciarse a la salida.
- Rapidez de respuesta: puede ser un tiempo fijo o depender de cuánto varíe la magnitud a medir. Depende de la capacidad del sistema para seguir las variaciones de la magnitud de entrada.
- Repetibilidad: error esperado al repetir varias veces la misma medida.

1.8 MOTOR DE CORRIENTE CONTINÚA.

Son de los más comunes y económicos como se aprecia en la figura 1.14, se pueden encontrar en la mayoría de los juguetes a pilas, constituidos, por lo general, por dos imanes permanentes fijados en la carcasa y una serie de bobinados de cobre ubicados en el eje del motor.

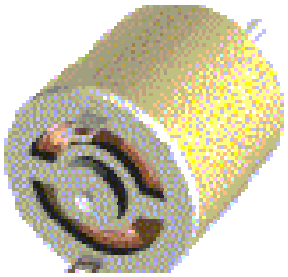


Figura 1.14 Gráfico de un Motor de CC.

Accionar un motor DC es muy simple y solo es necesario aplicar la tensión de alimentación entre sus bornes. Para invertir el sentido de giro basta con invertir la alimentación y el motor comenzará a girar en sentido opuesto.

El motor de corriente continua está compuesto de 2 piezas fundamentales: que son el rotor y el estator como se ilustran en las figuras 1.15 y 1.16.

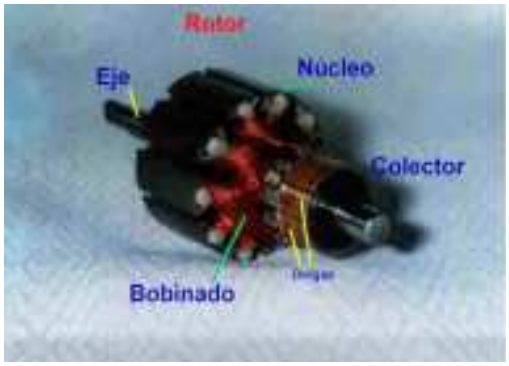


Figura 1.15 Rotor *Figura 1.16 Estator*

Dentro de éstas se ubican los demás componentes como:

- Escobillas y porta escobillas
- Colector
- Eje
- Núcleo y devanado del rotor
- Imán Permanente
- Armazón
- Tapas o campanas

Rotor. Constituye la parte móvil del motor, proporciona el torque para mover a la carga.

Estator. Esta parte de la máquina no se mueve y es la carcasa de la máquina.

Eje. Formado por una barra de acero fresada. Imparte la rotación al núcleo, devanado y al colector.

Núcleo. Se localiza sobre el eje. Fabricado con capas laminadas de acero, su función es proporcionar un trayecto magnético entre los polos para que el flujo magnético del devanado circule.

Para la obtención de la velocidad y la posición del motor se consideró la utilización de los componentes de un ratón mecánico-óptico ya que estos dispositivos utilizan un codificador óptico el que consta de una rueda perforada en 36 ranuras que representan los 360 grados del círculo, esta se mueve por medio de las rotaciones producidas por una esfera de acero recubierta por una capa de hule, y el codificador, el que se encarga de generar los pulsos transmitidos a un controlador que permite la interconexión entre la computadora y el periférico, es el

transductor de los impulsos generados por el circuito conectado a la parte mecánica.

1.8.1 FUENTE PWM.

Debido a que los motores de corriente continua necesitan ser "alimentados" con un pulso de magnitud determinada, se utiliza una fuente de Control de Ancho de Pulso (PWM Pulse Width Modulated de sus siglas en inglés) para que se pueda regular la velocidad de encendido y funcionamiento de una manera uniforme ya que en el momento de implementar este tipo de motores es necesario obtener error mínimo en el movimiento desde el momento de inicio del motor y durante los posibles cambios que se harán en el circuito. La fuente PWM produce los pulsos con un ciclo de trabajo determinado porque en el momento de que el circuito necesite menos tiempo de trabajo realiza la variación.
El diagrama de bloques de un motor DC se observa en la Figura 1.17.

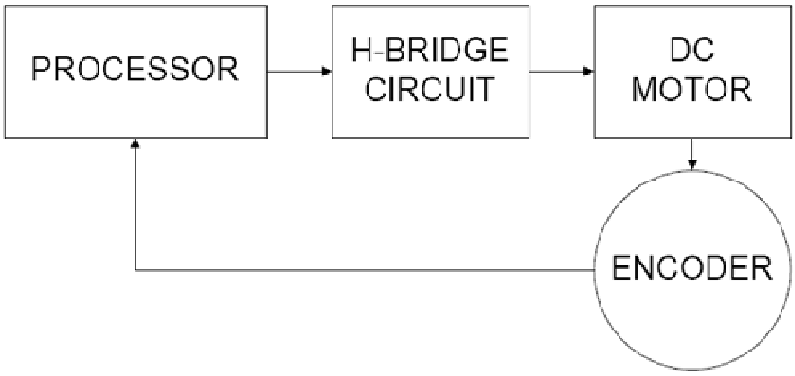


Figura 1.17 Diagrama de bloques de motor DC

1.8.2 TIPOS DE MOTORES D.C.

Los motores D.C se clasifican de acuerdo al tipo de bobinado de campo como motores Serie, Shunt, Shunt estabilizado, Compuesto. Sin embargo algunos de ellos pueden ser auto excitados o de excitación separada o pueden tener campos de imán permanente. Ellos muestran curvas muy diferentes de torque-velocidad y se conectan en diferentes configuraciones para diferentes aplicaciones. Los motores de imán permanente tienen la ventaja de no requerir una fuente de

potencia para el campo, pero tienen la desventaja de ser susceptibles a la desmagnetización por cargas de choque eléctricas o mecánicas.

1.8.2.1 Motor Shunt.

El motor Shunt o motor de excitación en paralelo es un motor eléctrico de corriente continua cuyo bobinado inductor principal está conectado en derivación o paralelo con el circuito formado por los bobinados inducido e inductor auxiliar.
A voltaje nominal y campo completo, la velocidad del motor Shunt aumentará 5% a medida que la corriente de carga disminuya de plena carga a sin carga.

1.8.2.2 Motor serie.

En un motor serie, el flujo del campo es una función de la corriente de la carga y de la curva de saturación del motor. A medida que la corriente de la carga disminuye desde plena carga, el flujo disminuye y la velocidad aumenta. La tasa de incremento de velocidad es pequeña al principio pero aumenta a medida que la corriente se reduce.

1.8.2.3 Motor Shunt Estabilizado.

Para vencer la potencial inestabilidad de un motor recto Shunt y reducir la “caída” de velocidad de un motor Compound, un ligero devanado serie es arrollado sobre el devanado Shunt. El flujo del devanado serie aumenta con la corriente de carga y produce un motor estable con una característica de caída de velocidad para todas las cargas. La regulación de velocidad de un motor Shunt estabilizado es típicamente menor al 15%

1.8.2.4 Motor compuesto.

Los motores compuestos tienen un campo serie sobre el tope del bobinado del campo Shunt. Este campo serie, consiste en pocas vueltas de un alambre grueso, es conectado en serie con la armadura y lleva la corriente de armadura.

El flujo del campo serie varia directamente a medida que la corriente de armadura varia, y es directamente proporcional a la carga. El campo serie se conecta de manera tal que su flujo se añade al flujo del campo principal Shunt.

1.8.3 FUNCIONAMIENTO DEL MOTOR DE CC.

Se basa en la interacción entre el campo magnético del imán permanente y el generado por las bobinas, ya sea una atracción o una repulsión hacen que el eje del motor comience su movimiento.

Cuando una bobina es recorrida por la corriente eléctrica, esta genera un campo magnético y como es obvio este campo magnético tiene una orientación es decir dos polos un polo NORTE y un polo SUR, si el núcleo de la bobina es de un material ferromagnético los polos en este material se verían como se indica en la figura 1.18.

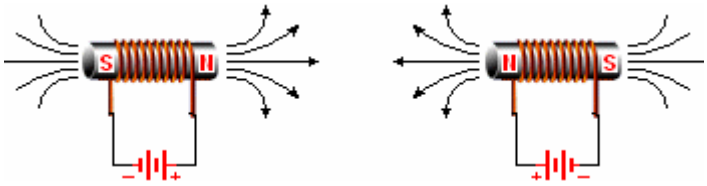


Figura 1.18 Gráfico de los polos en una bobina.

Estos polos pueden ser invertidos fácilmente con sólo cambiar la polaridad de la bobina, por otro lado al núcleo de las bobinas las convierte en un electroimán, ahora bien, si tienes nociones del efecto producido por la interacción entre cargas, recordarás que cargas opuestas o polos opuestos se atraen y cargas del mismo signo o polos del mismo signo se repelen, esto hace que el eje del motor gire produciendo un determinado torque, que es la fuerza de giro.

1.8.3.1 Control de Sentido de Giro para Motores-CC.

Existen varias formas de lograr que estos motores inviertan su sentido de giro una es utilizando una fuente simétrica o dos fuentes de alimentación con un interruptor simple de dos contactos y otra es utilizar una fuente común con un interruptor doble es decir uno de 4 contactos, en todos los casos es bueno conectar también un capacitor en paralelo entre los bornes del motor, para amortiguar la inducción que generan las bobinas internas del motor.

Ahora bien, estos Drivers que se mencionan son circuitos integrados que facilitan el diseño de nuestros circuitos, tales como el UCN5804, el BA6286, el L293B, L297, L298 o también se pueden ingeniar con el ULN2803 o el ULN2003, estos dos últimos son arreglos de transistores. Usaremos el Driver es el L293B

1.8.3.2 Ventajas de los Drivers L293B.

- Cada canal es capaz de entregar hasta 1A de corriente.
- Posee una entrada de alimentación independiente que alimenta los 4 drivers, es decir la que requieren los motores.
- El control de los drivers es compatible con señales TTL es decir con 5 voltios (estamos hablando de señales lógicas).
- Cada uno de los 4 drivers puede ser activado de forma independiente (por su terminal de entrada), o habilitado de dos en dos con un sólo terminal (Enable).

En la figura 1.19 se observa una imagen del integrado y su tabla de verdad.

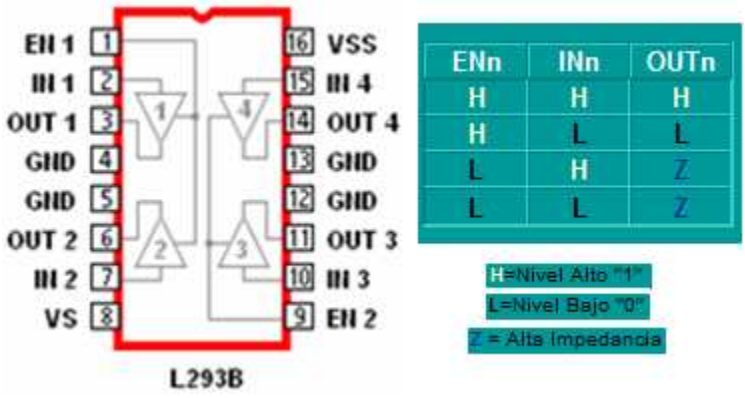


Figura 1.19 Pines del driver L293B.

Cada canal o driver es controlado por medio de una señal de control compatible TTL y los canales se habilitan de dos en dos por medio de las señales de control EN1 (canal 1 y 2) y EN2 (canal 3 y 4), en la tabla de la figura 1.19, vemos el funcionamiento de las entradas y cómo responden las salidas. Así pues, vemos que poniendo a nivel alto la entrada de habilitación "EN" del driver, la salida de este "OUT" pasa de alta impedancia al mismo nivel que se encuentre la entrada del driver "IN" pero amplificado en tensión y en corriente, siendo esta de 1A

máximo. La tensión de alimentación del circuito integrado no es la misma que se aplica a las carga conectada a las salidas de los drivers, y para estas salidas se alimenta el driver por su patita número 8 (Vs), la tensión máxima aplicable a esta patita es de 36V

1.9 REGULADOR DE TENSIÓN.

Un regulador de tensión (a veces traducido del inglés como regulador de voltaje) es un dispositivo electrónico diseñado para proteger aparatos eléctricos y electrónicos sensibles a variaciones de diferencia de potencial o voltaje y ruido existente en la corriente alterna de la distribución eléctrica. Los reguladores de voltaje están presentes en las fuentes de alimentación de corriente continua reguladas, cuya misión es la de proporcionar una tensión constante a su salida.
En la figura 1.20 se observa el regulador de voltaje con el símbolo lógico.

Figura 1.20 Reguladores de voltaje L7805 y LM317.

1.9.1 REGULACIÓN CON DIODO ZENER.

El diodo Zener es un tipo especial de diodo preparado para trabajar en la zona inversa. Cuando se alcanza la denominada tensión Zener en polarización inversa, el diodo recorta la onda de tensión, de este modo mantiene la tensión constante entre sus terminales dentro de ciertos márgenes. Si la corriente es muy pequeña la tensión empezará a disminuir, pero si es excesiva puede destruir el diodo.
Esta propiedad hace que el diodo Zener sea utilizado como regulador de tensión en las fuentes de alimentación.

1.9.2 REGULADORES CONMUTADOS.

Poseen mayor rendimiento de conversión, ya que los transistores funcionan en conmutación, reduciendo así la potencia disipada en estos y el tamaño de los

disipadores. Se pueden encontrar este tipo de fuentes en los ordenadores personales, en electrodomésticos, reproductores DVD, etc. Una desventaja es la producción de ruido electromagnético producido por la conmutación a frecuencias elevadas, teniendo que apantallar y diseñar correctamente la PCB (Placa de Circuito Impreso) del convertidor.

1.9.3 REGULADORES CON CIRCUITO INTEGRADO.

Son componentes muy parecidos a los transistores de potencia, suelen tener tres terminales, uno de entrada, un común o masa, y uno de salida, tienen una capacidad de reducción del rizado muy alta y normalmente sólo hay que conectarles un par de condensadores. Existen circuitos reguladores con un gran abanico de tensiones y corrientes de funcionamiento. La serie más conocida de reguladores integrados es la 78xx y la serie 79xx para tensiones negativas. Los de mayor potencia necesitarán un disipador de calor, este es el principal problema de los reguladores serie lineales tanto discreto como integrado, al estar en serie con la carga las caídas de tensión en sus componentes provocan grandes disipaciones de potencia.

1.10 ZUMBADOR

Zumbador, buzzer en inglés, es un dispositivo electrónico que produce un sonido o zumbido continuo o intermitente de un mismo tono. Sirve como mecanismo de señalización o aviso, y son utilizados en múltiples sistemas como en automóviles o en electrodomésticos. Inicialmente este dispositivo estaba basado en un sistema electromecánico que era similar a una campana eléctrica pero sin el badajo (es la parte interna de la campana) metálico, el cual imitaba el sonido de una campana, su forma física se muestra en la figura 1.21.

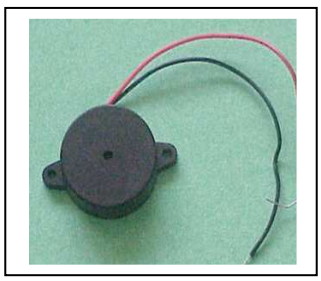


Figura 1.21 Buzzer

Su construcción consta de dos elementos, un electroimán y una lámina metálica de acero. El zumbador puede ser conectado a circuitos integrados especiales para así lograr distintos tonos. Cuando se acciona, la corriente pasa por la bobina del electroimán y produce un campo magnético variable que hace vibrar la lámina de acero sobre la armadura.

1.11 POWER LOGIC 8 BIT SHIFT REGISTER.

1.11.1 DESCRIPCIÓN.

En la figura 1.22 se observa la imagen física del TPIC6B595N, el cual es monolítico, de potencia media y alta tensión de 8 bits, diseñado para usarse en sistemas que requieren relativamente potencia de carga alta. El dispositivo contiene una construcción de voltaje fijo en las salidas de inducción transitoria de protección. Puede manejar aplicaciones de transmisión de potencia incluidos relés, selenoides y otras cargas de potencia media o alto voltaje. Este dispositivo tiene una entrada serial y una salida paralela de 8 bits, el registro de desplazamiento se alimenta de una memoria de almacenamiento de 8 bits, un registro tipo D.

Las transferencias de datos son a través de ambos registros de desplazamiento y de almacenamiento en el flanco ascendente del reloj del registro de desplazamiento (SRCK) y el reloj de registro (RCK), respectivamente. El registro de almacenamiento para la transferencia de datos en el búfer de salida se limpia cuando el Registro de desplazamiento es alto (SRCLR). Cuando SRCLR es bajo, el registro de desplazamiento de entrada se borra.

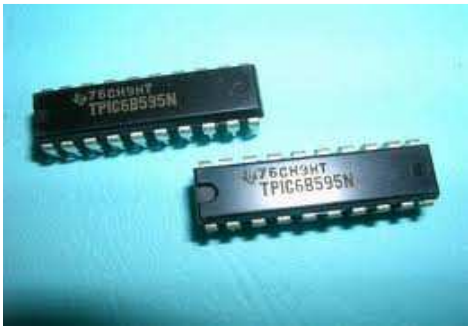


Figura 1.22 Imagen TPIC.

Cuando la salida de habilitación (G) se mantiene en alto, todos los datos en el buffer de salida se mantiene en bajo y todas las salidas de drenaje están apagadas.

Cuando G se mantiene en bajo, los datos del registro de almacenamiento son transparentes para los buffers de salida. Cuando los datos en los buffers de salida son bajos, el transistor DMOS está apagado. Cuando los datos están en alto, las salidas del transistor DMOS tienen una capacidad de caída de corriente.

La salida de serie (SER OUT) permite la conexión en cascada de datos del registro de desplazamiento a otros dispositivos. La salida serial se registró fuera del dispositivo al borde de la caída SRCK para proporcionar el tiempo de retención adicional para la aplicación en cascada.

1.11.2 DIAGRAMA LOGICO DEL REGISTRO DE DESPLAZAMIENTO.

En la figura 1.23 se muestra el diagrama lógico del registro de desplazamiento con sus 8 salidas de drenaje y como trabaja el reloj.

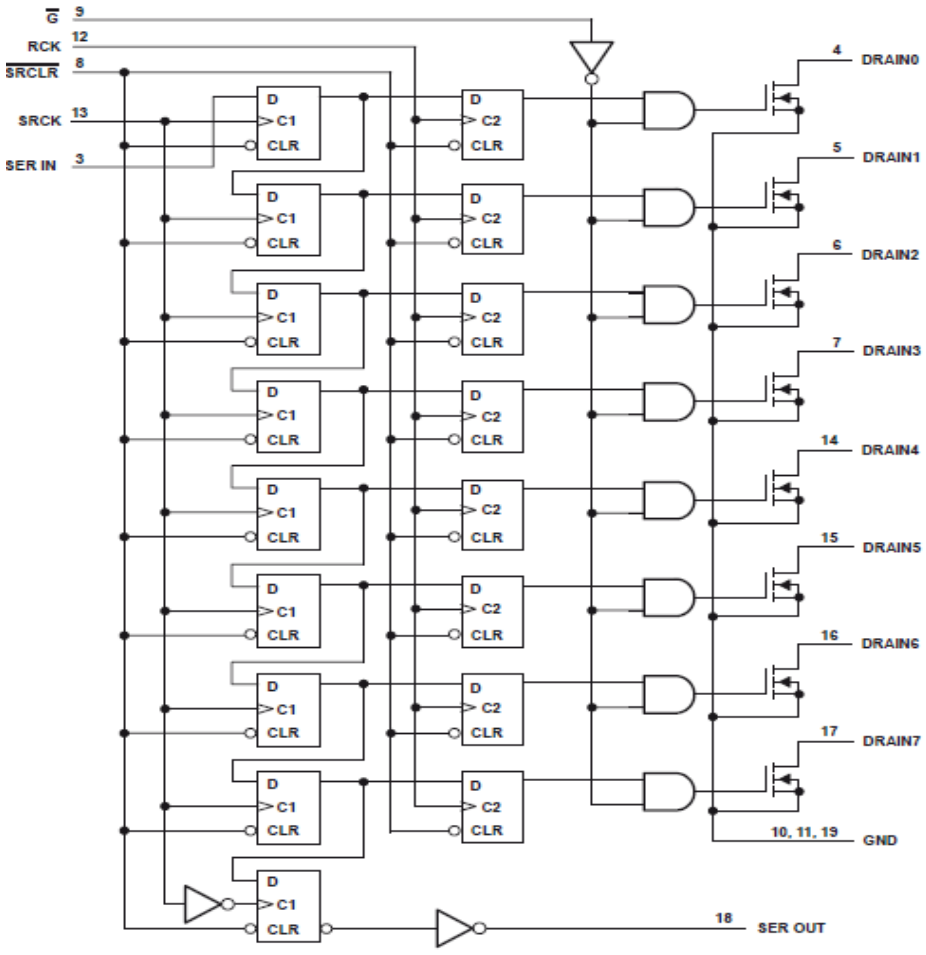


Figura 1.23 Diagrama lógico del TPIC6B595N.

2. CAPÍTULO II: DISEÑO Y CONSTRUCCIÓN DE LA TARJETA (DESARROLLO DEL HARDWARE).

2.1 CONSTRUCCIÓN DEL HARDWARE.

Esta Libreta Electrónica de Prácticas está dirigida a los estudiantes que estén aprendiendo a programar, que deseen optimizar su tiempo y satisfacer la necesidad de realizar programas usando microcontroladores PIC con Visual C # mediante comunicación USB y RS-232.

Como podemos apreciar la figura 2.1, el proyecto consta de varias etapas: Una de ellas es la comunicación externa, que consiste en enviar datos desde la PC, mediante la comunicación USB o RS-232, la siguiente etapa es el procesamiento de los datos y consiste en el acondicionamiento de la información a transmitirse para que llegue al microprocesador y de aquí se envíe la información y la última etapa que es la presentación o visualización en cada una de las aplicaciones.

Siempre debe haber un maestro y un esclavo en la comunicación, el maestro siempre empieza la comunicación la misma que se inicia enviando números en los buffers, como tenemos 8 buffers, el buffer 0 envía los comandos y en el resto de buffers se envían los demás parámetros necesarios del comando.

El uC recibe y decodifica los comandos y una vez listos los comandos se envían a sus respectivos periféricos o aplicaciones para ser observados.
El puerto USB administra 8 buffers en el proyecto al igual que el puerto RS232.
Las principales características que se han tomado en cuenta para desarrollar el proyecto son:

- Interfaz novedosa y llamativa de Visual C#.
- Fácil acceso al sistema por parte del usuario.
- Costos no muy elevados en la construcción de la Libreta Electrónica.

A continuación se muestra en la figura 2.1 el diagrama de bloques del proyecto planteado.

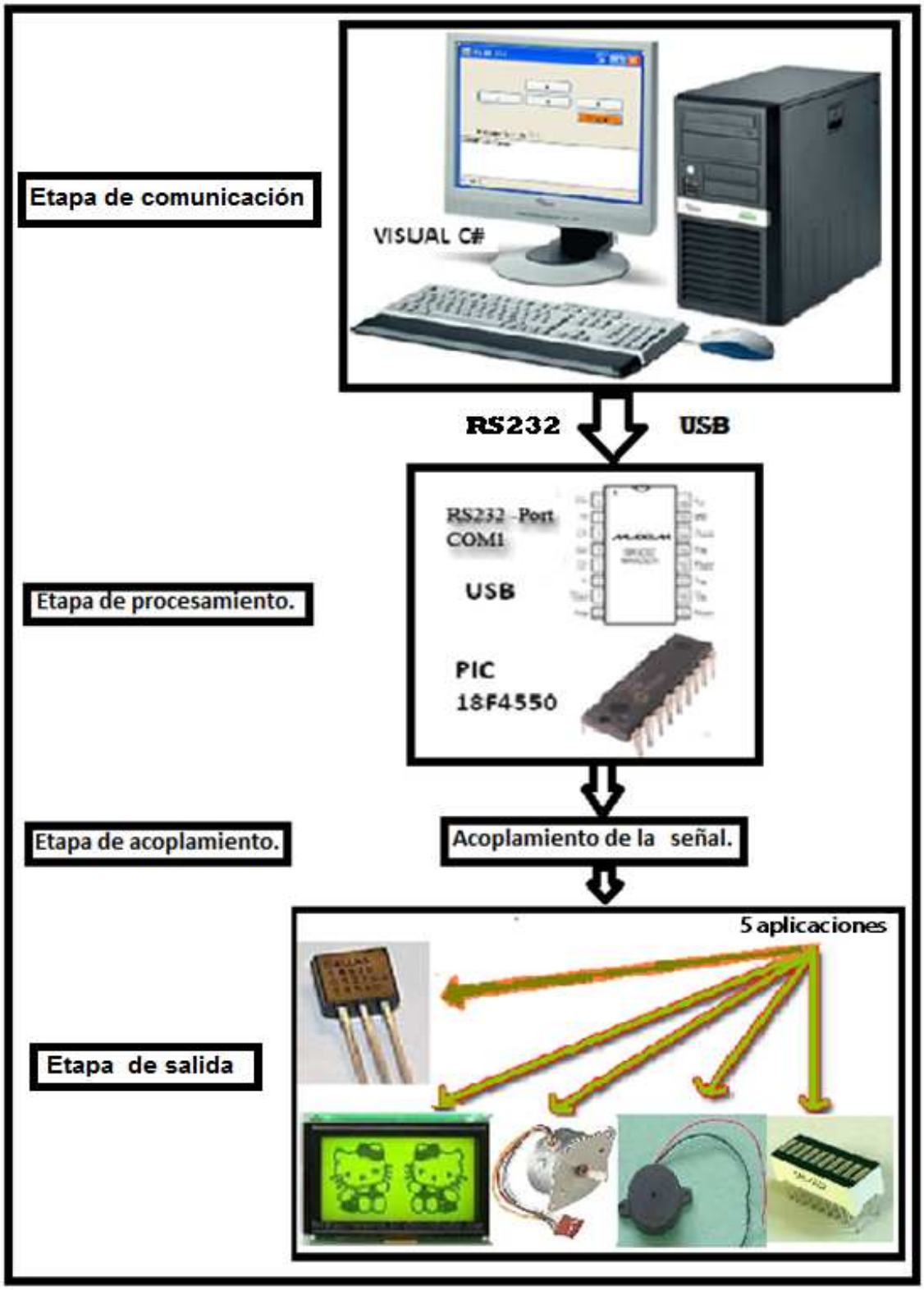


Figura 2.1 Diagrama de bloques del Proyecto.

2.2 REQUERIMIENTOS DEL HARDWARE.

Una vez que se han definido las características de la Libreta Electrónica de prácticas se ha iniciado un estudio necesario y se ha concluido en realizar un circuito general que comprende 5 aplicaciones.
La Libreta Electrónica es una tarjeta que consta de los conectores USB para la transmisión de datos mediante el puerto USB y el conector DB9 para la transmisión de datos mediante el puerto RS-232, los cuales son usados para la transmisión de datos hacía el PIC; además contiene un reloj en tiempo real en el que se puede ver la hora enviada desde la PC.

Aplicaciones que comprende la Libreta Electrónica Prácticas:

- GLCD
- BUZZER
- SENSOR
- MOTORES DE CC
- BARRA DE LEDs

GLCD: Podremos apreciar el dato enviado desde la PC, el texto que enviemos o leer la temperatura del sensor, además podremos comprobar qué motor está funcionando, observar el valor enviado del motor es decir, lo que varía el ancho de pulso, también observaremos cómo rota el cero lógico o el uno lógico en la barra de LEDs. Además será posible observar el tipo de comunicación que estemos usando.

BUZZER: Es un dispositivo mediante el cual podremos escuchar diferentes sonidos, pero en este caso produce un sonido o zumbido continuo o intermitente de un mismo tono.

SENSOR: Mediante este dispositivo podremos tomar medidas de cambios de temperatura ambiente y esta medida la podremos observar en el GLCD.

MOTORES DE CC: Usamos 2 motores de corriente continua ya que en el PIC tenemos 2 pines PWM, los motores de c.c. giran en el sentido horario y anti

horario y además podemos aumentar o disminuir la velocidad de los mismos modificando la programación.

BARRA DE LEDs: Mediante esta barra podremos observar cómo se desplaza la luz en cada LED más rápido o más lento, también es posible invertir el sentido de giro de la luz o formar números que se puedan leer en binario.

2.3 COMPONENTES DEL HARDWARE.

Para la construcción de la Libreta Electrónica de Prácticas se ha tomado en cuenta los componentes necesarios que se usarán en cada aplicación.
A continuación describiremos el funcionamiento y las características principales de cada dispositivo.

2.3.1 SENSOR DE TEMPERATURA DS18B20.

El sensor DS18B20 digital ofrece cambios de temperatura con una precisión de 9 bits a 12 bits para mediciones de temperatura en grados Celsius y tiene una función de alarma programable no volátil, en el punto superior e inferior de disparo por el usuario. El DS18B20 comunica un cable bus que, por definición, sólo requiere una línea de datos (y tierra) para la comunicación con un microprocesador central. Tiene un rango de temperatura de -55ºC a +125ºC y tiene una precisión de - 0,5ºC en el rango de -10ºC a ±85ºC. Además, el DS18B20 puede obtener energía directamente de la línea de datos (potencia parásito) eliminando la necesidad de una fuente de alimentación externa.
Características:

- Interfaz de un cable para la comunicación.
- Cada dispositivo tiene un código serial único de 64-bit guardado en una memoria ROM interna.
- No requiere componentes externos.
- Puede ser alimentado a través de la línea de datos. El rango de voltajes de alimentación es de 3V a 5.5V
- Realiza mediciones desde –55℃ hasta +125℃ (–67℉ hasta +257℉)

- ±0.5℃ de exactitud desde–10℃ hasta +85℃
- Resolución seleccionable de 9 a 12 bits.
- Convierte la temperatura a una palabra digital de 12 bits en 750ms (máximo.)
- Alarmas configurables por el usuario en memoria no volátil
- Aplicaciones de control térmico, sistemas industriales, productos finales, termómetros y cualquier otro sistema que sea sensible térmicamente.

En la figura 2.2 se observa la configuración de pines del sensor DS18B20 y en la figura 2.3 se muestra una tabla con la función que cumple cada pin.

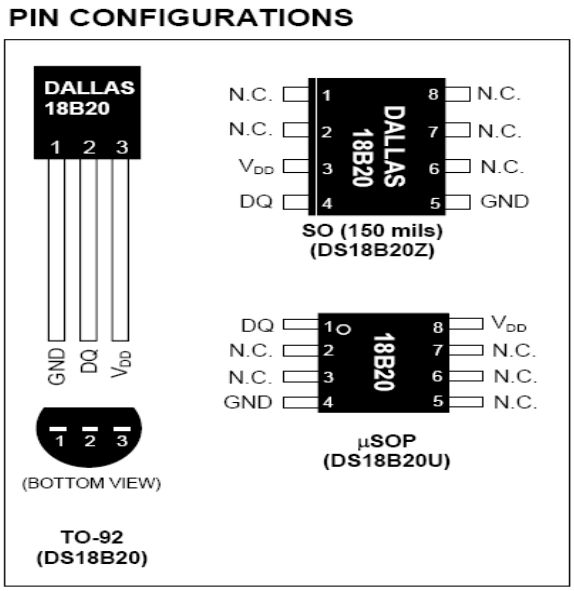


Figura 2.2 Configuración de pines del sensor DS18B20.

PIN DESCRIPTION

PIN			NAME	FUNCTION
SO	µSOP	TO-92		
1, 2, 6, 7, 8	2, 3, 5, 6, 7	—	N.C.	No Connection
3	8	3	V_{DD}	Optional V_{DD}. V_{DD} must be grounded for operation in parasite power mode.
4	1	2	DQ	Data Input/Output. Open-drain 1-Wire interface pin. Also provides power to the device when used in parasite power mode (see the *Powering the DS18B20* section.)
5	4	1	GND	Ground

Figura 2.3 Descripción de pines del sensor DS18B20.

En la figura 2.4 podemos observar un diagrama de bloques del sensor DS18B20.

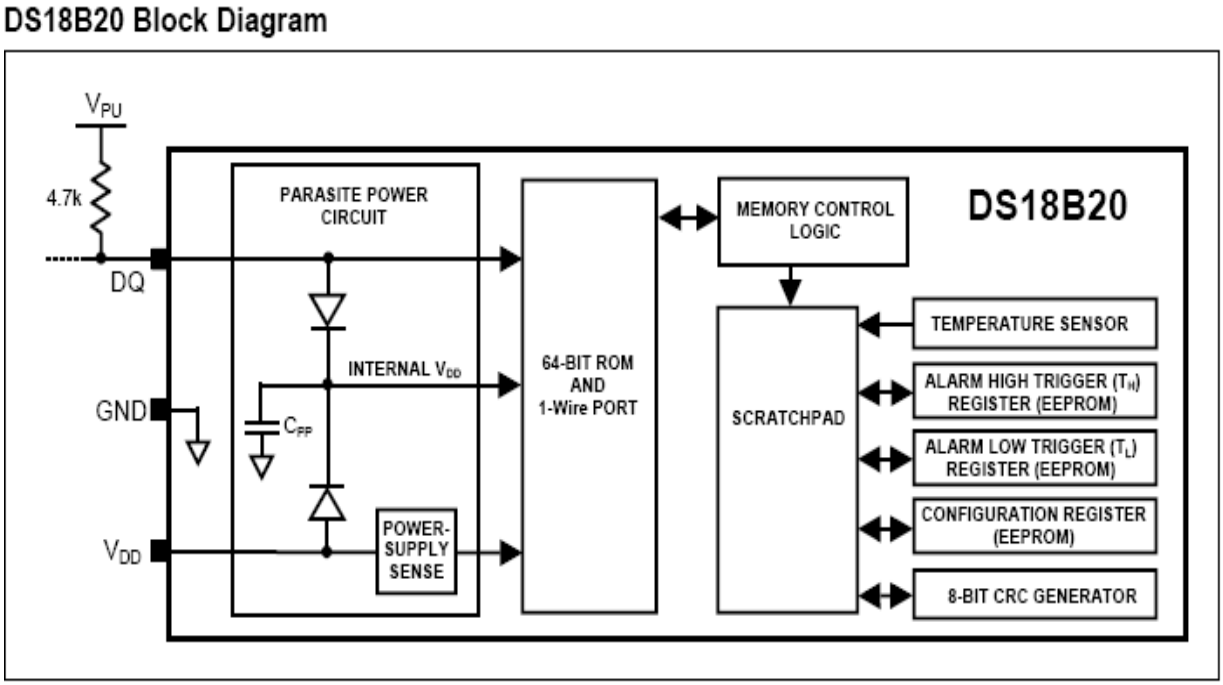


Figura 2.4 Diagrama de bloques del sensor DS18B20.

Más información del Sensor de temperatura se puede encontrar en el Anexo B.

2.3.2 BUZZER

2.3.2.1 Introducción.

Zumbador, buzzer en inglés, es un dispositivo electrónico que produce un sonido o zumbido continuo o intermitente de un mismo tono. Un zumbador electrónico es un “miniparlante” (minibocina) de bajo costo que se utiliza para hacer sonidos.
El sonido generado por el zumbador puede cambiarse alterando las señales electrónicas suministradas por el microcontrolador.

Inicialmente este dispositivo estaba basado en un sistema electromecánico que era similar a una campana eléctrica pero sin el badajo metálico, el cual imitaba el sonido de una campana.

En la figura 2.5 se muestra una imagen con las diferentes formas del buzzer.

Figura 2.5 Formas del buzzer.

2.3.2.2 ¿Para Qué se Utilizan los Zumbadores?

Los zumbadores se utilizan en una gran variedad de diferentes productos para dar "retroalimentación" al usuario, este sonido da retroalimentación al usuario para indicarle que se recibió la señal del botón presionado. Sirven como mecanismo de señalización o aviso, y son utilizados en múltiples sistemas como en automóviles o en electrodomésticos. Un buen ejemplo de esto es una máquina expendedora, la cual emite un sonido cada vez que se presiona un botón para escoger un refresco. Otro tipo de zumbador se utiliza a menudo, en tarjetas musicales de cumpleaños, para tocar una melodía cuando se abre la tarjeta.

2.3.2.3 ¿Cómo se usan los Zumbadores?

Simplemente conecte un cable al pin de salida del zumbador y el otro cable a 0V (tierra). Tome en cuenta que los zumbadores más económicos no tienen cubierta plástica exterior. En estos casos es necesario montar el zumbador sobre una sección del circuito impreso (con cinta adhesiva de doble contacto) para crear un sonido que se pueda escuchar.

El circuito impreso actúa como una "caja de resonancia" (baffle) y amplifica el sonido emitido por el zumbador.

Tiene dos contactos con polaridad, como se aprecia en la figura 2.6. El positivo suele estar marcado con un signo (+) mientras que el negativo va en negro o con un signo (-).

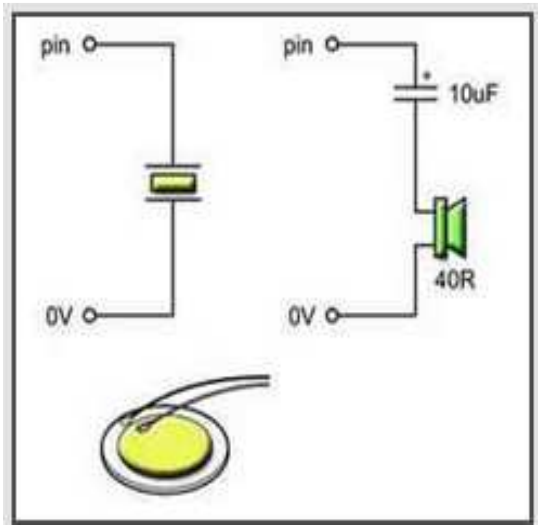


Figura 2.6 Conexión del buzzer

2.3.2.4 Características del Buzzer:

- Zumbador redondo 11,5x10mm
- Nivel sonoro 40 dB máximo.
- Alimentación 5 a 15 Voltios CC.
- Consumo de 15 mA.

2.3.3 GLCD.

Una Pantalla Gráfica de Cristal Líquido, está formada por una matriz de píxeles monocromos colocados delante de una fuente de luz o reflectora. Hay versiones de pantallas con diferentes controladores, para este proyecto se usó el controlador KS0108. En la figura 2.7 se observa el gráfico de un GLCD.

Figura 2.7 Pantalla de cristal líquido Gráfica.

El GLCD tiene pines de control y de datos. Los de control son: CS1, CS2, DI, RW, E y Reset. Los de datos son del DB0 al DB7. Este GLCD se divide en 2 zonas de

64 * 64 pixeles. Con los pines de selección CS1 y CS2 seleccionamos qué zona queremos usar. Los pines DI y RW sirven para controlar el display y decirle si queremos leer o escribir datos desde o hacia él display, ejecutar alguna instrucción o leer el estado del GLCD. El pin E (enable) sirve como forma de decirle al display que lea los otros pines de control y ejecute lo que se le indica por medio de estos. Con los pines de datos se puede enviar o recibir datos del display y leer el estado del chip de control del display.

El GLCD dispone de una memoria RAM interna del mismo tamaño de la capacidad que dispone la pantalla, por ejemplo si una pantalla tiene un tamaño de 128 pixeles de largo por 64 pixeles de alto (128x64) tiene una memoria RAM interna de la misma capacidad (128x64). Por lo general son manejados por microcontroladores para la configuración y utilización de la misma.

2.3.3.1 Características de un GLCD 128x64

- Conformado por una matriz de puntos de visualización de 128 pixeles de largo por 64 pixeles de alto.
- Su iluminación de fondo está entre verde-amarillo cuando se enciende.
- Fácil manejo con microprocesadores de 8-Bits.
- Bajo poder de consumo.
- Contiene dos controladores internos un KS0108B y KS0107B.

2.3.3.2 Descripción de los pines de conexión de una GLCD 128x64.

- PIN 1: VSS (Conexión a tierra)
- PIN 2: VDD (Conexión de alimentación - +5V)
- PIN 3: V0 (Voltaje de salida negativa, por lo general es usado con un potenciómetro con el PIN 18 para el ajuste del contraste de los pixeles)
- PIN 4: D/I (Datos de E/S para el cambio de registro)
- PIN 5: R/W (Determina si los datos se van a leer o escribir)
- PIN 6: E (Habilita la comunicación con la GLCD)
- PIN 7 - 14 (Especifica un dato de 8-Bits de información)
- PIN15: CS1 (Indica si se selecciona la primera mitad de la pantalla, pixeles 0-63)

- PIN16: CS2 (Indica si se selecciona la segunda mitad de la pantalla, pixeles 64-127)
- PIN17: RESETB (Señal de reinicio, funciona de varias forma dependiendo de la ocasión)
- PIN18: VEE (Conexión de ajuste de contraste de los pixeles)
- PIN19: A (Conexión positiva de la luz de fondo, por lo general son +5V)
- PIN20: K (Conexión negativa de la luz de fondo, por lo general es tierra)

2.3.4 BARRA DE LEDS.

Figura 2.8 Gráfico de la Barra de LEDs.

En la figura 2.8 se observa la barra de LEDs, en un principio cuando fueron creados los LEDs, estos fueron diseñados con el único propósito de ser utilizados como señalizadores luminosos que indicaban el funcionamiento electrónico su potencia y cantidad de luz era muy pequeña, pero para el efecto utilizado era más que suficiente. Con el paso de los tiempos estos LEDs han ido beneficiándose de muchos avances tecnológicos, por lo que se han creado LEDs de alta luminosidad hasta llegar a los LEDs RGB que no son más que tres diodos emisores

2.3.4.1 Descripción de la barra de LEDs:

Esta es una barra de 10 segmentos de LEDs que puede ser empleada en una gran cantidad de proyectos. Tiene un footprint compacto, es fácil de conectar y son excelentes para prototipos o productos terminados. Esencialmente son 10 LEDs individuales encapsulados cada uno con conexiones individuales de ánodo y cátodo.

Esta barra de 10 LEDs rojos ultra brillantes puede utilizarse para contadores, vúmetros y cualquier proyecto que requiera presentación de LEDs en un formato compacto.

2.3.5 CIRCUITO INTEGRADO L293B.

Un motor no puede ser conectado directamente al microcontrolador ya que el consumo de corriente es elevado, dañaría el PIC, por esta razón se hace uso del driver (L293B) para controlar el motor.

2.3.5.1 Circuito Integrado L293B.

El L293B es un driver de 4 canales capaz de proporcionar una corriente de salida de hasta 1A por canal. Cada canal es controlado por señales de entrada compatibles TTL y cada pareja de canales dispone de una señal de habilitación que desconecta las salidas de los mismos.

2.3.5.2 Diagrama de Bloques

En la Figura 2.9, se muestra el diagrama de bloques del L293B. La señal de control EN1 activa la pareja de canales formada por los drivers 1 y 2. La señal EN2 activa la pareja de drivers 3 y 4. Las salidas OUTN se asocian con las correspondientes OUTn. Las señales de salida son amplificadas respecto a las de entrada tanto en tensión (hasta +Vss) como en corriente (máx. 1 A).

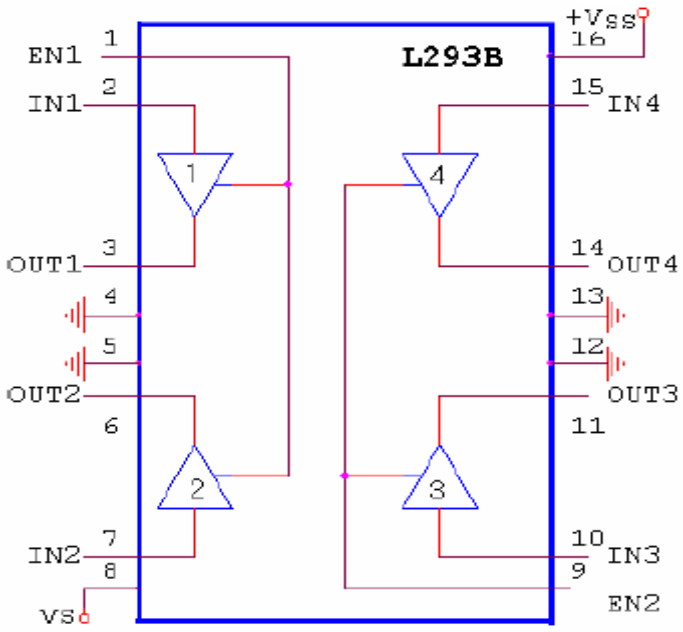


Figura 2.9 Diagrama de bloques L293B

2.3.5.3 Control del giro de un motor en los dos sentidos.

El circuito de la Figura 2.10 permite controlar el doble sentido de giro del motor. Cuando la entrada C está a nivel bajo y la D a nivel alto, el motor gira hacia la izquierda. Cambiando la entrada C a nivel alto y la D a nivel Bajo, se cambia el sentido de giro del motor hacia la derecha.

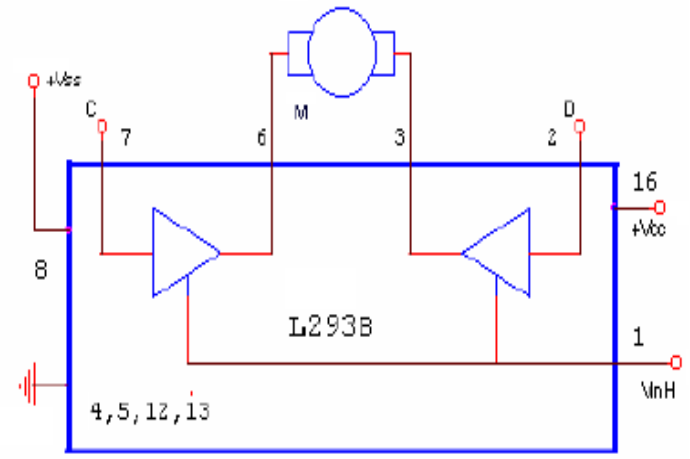


Figura 2.10 Gráfico de los polos en una bobina.

2.3.5.4 Conexión de diodos en el motor.

Para proteger el circuito contra posibles picos de corriente inversa cuando se arranca el motor, se debe conectar unos diodos tal y como se muestra en la figura 2.11

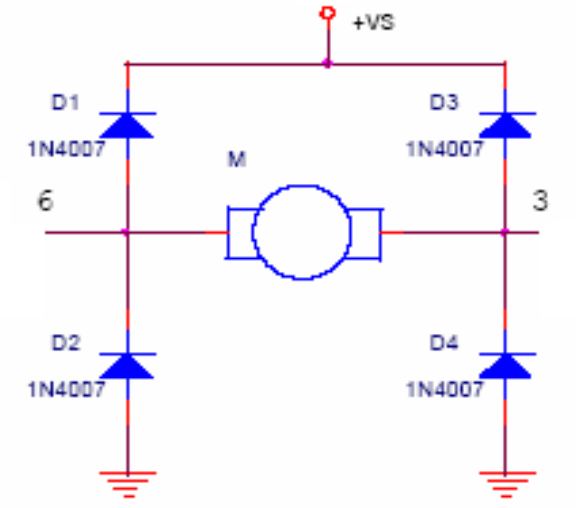


Figura 2.11 Conexión de los diodos para el motor.

2.3.5.5 Control de velocidad para un motor mediante el PWM.

El control de velocidad se realiza mediante la modulación de ancho de pulso o PWM. Consiste en hacer variar el ancho del pulso o ciclo de trabajo de una señal. Mientras el pulso sea más ancho, la velocidad del motor será más rápida y en cambio si el ancho del pulso es más pequeño la velocidad disminuirá. De esta manera se controla la cantidad de corriente, la tensión no varía y en consecuencia el torque del motor se mantiene.

2.3.6 REGISTRO DE DESPLAZAMIENTO TPIC6B595N.

Este dispositivo contiene una serie de 8 bits con entrada serial y salida paralela. La transferencia de datos es a través de ambos registros; de desplazamiento y de almacenamiento en el flanco ascendente del reloj del registro de desplazamiento (SRCK) y el registro de reloj (RCK), respectivamente. En el registro de almacenamiento la transferencia de datos en el búfer de salida se limpian cuando el registro de desplazamiento es alto (SRCLR) y cuando SRCLR es bajo, el registro de desplazamiento de entrada se borra.
Cuando la salida de habilitación (G) se mantiene en alto, todos los datos en el buffer de salida se mantiene en bajo y todas las salidas de drenaje están apagadas. Mayor información se puede encontrar en el Anexo E.

2.3.6.1 Características del TPIC6B595N.

- Registro de desplazamiento de 8 bits.
- Rango de voltaje de 4,5 V a 5,5 V
- Temperatura de Funcionamiento: -40 °C a +125 °C
- Temperatura máxima de funcionamiento: 125 °C
- Capacidad de corriente: 500mA.
- Bajo consumo de energía.
- Voltaje de salida fijo 50V.

En la figura 2.12 se muestra la configuración de pines del TPIC6B595N y el símbolo lógico para su mayor comprensión.

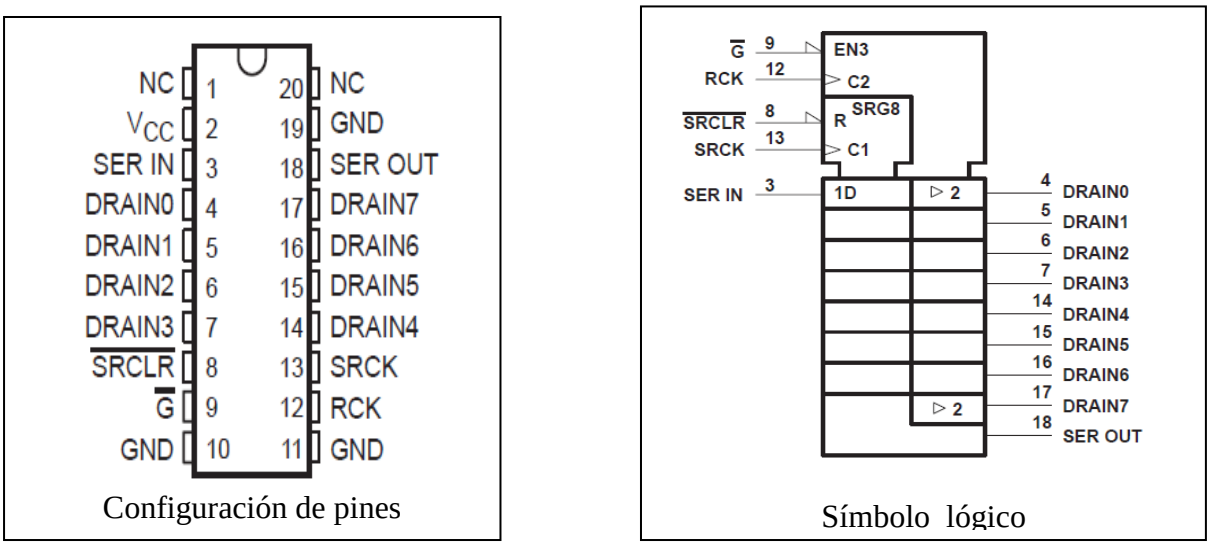


Figura 2.12 Configuración de pines y símbolo lógico del TPIC.

2.3.7 RTC RELOJ EN TIEMPO REAL DS1307.

2.3.7.1 Descripción.

El DS1307 Real-Time-Clock Serie, es un dispositivo de bajo consumo de energía, completo con código binario decimal (BCD), reloj/calendario más 56 bytes de NV SRAM. Las direcciones y datos son transferidos a través de 2 hilos serie, bus bi-direccional.

El reloj calendario provee información de, segundos, minutos, horas, días, fechas, meses y años. La fecha de cada mes se ajusta automáticamente durante meses menores de 31 días, incluyendo correcciones para el año bisiesto. El reloj funciona en cualquier formato de 24 horas o 12 horas con indicador AM/PM, El reloj tiene incorporado un sensor de tensión que detecta fallas de energía y cambia automáticamente al suministro de batería de respaldo, en la figura 2.13 se muestra la figura.

Figura 2.13 Reloj en Tiempo Real.

2.3.7.2 Características.

- Reloj en tiempo real (RTC) cuenta segundos, minutos, horas, fecha del mes, mes, día de la semana, y año con año bisiesto, compensación válido hasta 2100.
- 56-Bytes, con respaldo de batería, no volátil (NV) de RAM para almacenamiento de datos.
- Interface Serie I2C.
- Onda-Cuadrada programable de la señal de salida.
- Detector automático de fallo de energía y circuito de conmutación.
- Consume menos de 500nA en la batería -- Modo de copia de seguridad con el oscilador funcionando.
- Rango de temperatura industrial opcional: -40℃ a +85℃.
- Disponible en 8 pines.
- Reconocido Underwriters Laboratory (UL).

2.3.7.3 Asignación de pines.

En la figura 2.14 se muestra la configuración de pines y el diagrama lógico del RTC.

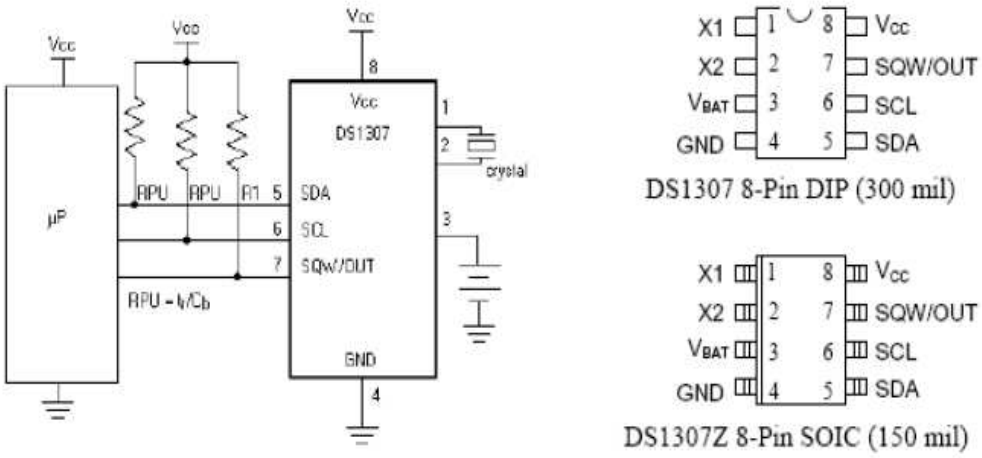


Figura 2.14 Configuración de pines y diagrama lógico del RTC.

2.3.7.4 Operación del RTC.

El DS1307 funciona como un dispositivo esclavo en el bus serie. El acceso se obtiene mediante la aplicación de una condición de START (Inicio) y la prestación de un código de identificación del dispositivo seguido de una dirección de registro.

Se puede acceder a registros posteriores de forma secuencial hasta que es ejecutada una condición STOP (parada).

Cuando VCC cae por debajo de 1,25 x VBAT (Voltaje de batería), un dispositivo en curso invalida el acceso y restablece el contador de dirección del dispositivo. En este momento, pueden no ser reconocidas las entradas al dispositivo para evitar que se escriban datos erróneos en el mismo, por fuera de la tolerancia del sistema. Cuando VCC cae por debajo de VBAT el dispositivo conmuta a batería de baja corriente a modo de seguridad. Tras el encendido, el dispositivo conmuta de la batería a VCC cuando es mayor que VBAT + 0,2 V y reconoce las entradas, cuando VCC es mayor de 1,25 x VBAT. Mayor información se puede encontrar en el anexo C.

2.3.8 MICROCONTROLADOR PIC18F4550.

El PIC 18F4550 es un microcontrolador de propósito general muy versátil y muy económico. Cuenta con interface USB 2.0 full speed y 48MHZ de frecuencia de oscilación para un rango de transferencia de datos de 1.5 Mbps mínimo y 12 Mbps como máximo, incluye un controlador USB interno, cuenta con 2 pines de salida para conectar directo al PC las líneas de transmisión de datos USB (D+ y D-) sin la necesidad de circuitería externa.

La imagen del PIC18F4550 se muestra en la figura 2.15.

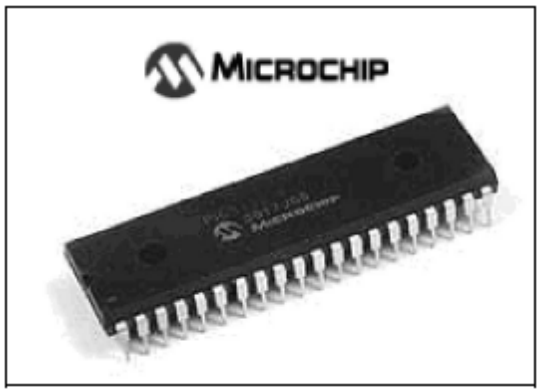


Figura 2.15 Circuito integrado 18F4550.

2.3.8.1 Características.

- Posee 32 Kb de Memoria Flash para almacenar los programas.
- 2 Kb de SRAM para Memoria volátil.
- 256 bytes de EEPROM (memoria no-volátil) para almacenamiento permanente de datos.
- Cuenta con un número de ciclos de escritura/borrado en la Memoria Flash de 100.000. Más de 40 años de retención de datos Flash/EEPROM.
- 1.000.000 de ciclos de escritura/borrado en la Memoria EEPROM.
- Las instrucciones son de 1 byte de longitud con la excepción de algunas que ocupan 2 bytes.
- Las características de la interface USB 2.00 Full Speed incluyen: Un transmisor receptor en el dispositivo y un puerto paralelo para transferir datos directamente a periféricos externos con una mínima carga al CPU.
- Soporta hasta 32 endpoints (16 bidireccionales). El endpoints es un buffer que almacena múltiples bytes o un bloque de datos o un registro en el micro.
- Tiene un set de instrucciones extendido.
- 1KB de la RAM, puede ser dedicado al buffer USB.
- Watchdog timer extendido: programable de 41ms a 131ms
- Cuatro temporizadores (3 de 16 bits y uno de 8 bits)
- Módulo EUSART que soporta interface serie RS232
- Puerto SSP de comunicación serie I2C y SPI
- Convertidor A/D de 10 bits de resolución con más de 12 canales.
- 2 comparadores analógicos con multiplexión de entrada.
- 2 Módulos Captura/Comparación/PWM con 16 bits de resolución.
- Circuito detector de bajo voltaje, reset por caída de tensión.

2.3.8.2 Distribución de pines.

En la figura 2.16 se observa la distribución de pines del PIC 18F4550.

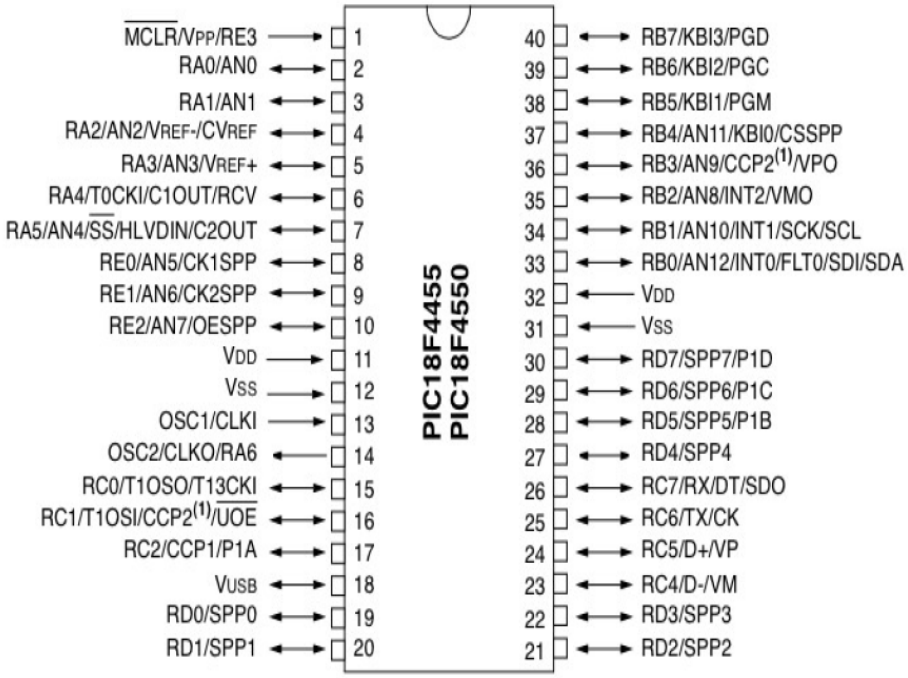


Figura 2.16 Distribución de pines del PIC 18F4550.

Se puede encontrar más información sobre el PIC 18F4550 en el Anexo A.

2.4 EMSAMBLAJE DEL HARDWARE.

Para el ensamblaje de la Libreta Electrónica debemos tener todos los dispositivos y herramientas necesarias para su armado, también es necesario tener el diseño del circuito y el software.

Una vez que tengamos claro el funcionamiento del circuito y sus características procedemos a realizar el ensamblaje del hardware. Para esta etapa usaremos la información recopilada anteriormente de los elementos electrónicos a utilizarse y los anexos que están detallados más adelante.

2.4.1 DESARROLLO DEL CIRCUITO.

2.4.1.2 Lista de Materiales.

En la figura 2.17 se describen los elementos usados en el desarrollo del circuito.

- Dos conectores USB
- Dos connector DB9
- Tres LEDs
- Un regulador de voltaje 7805
- Siete capacitores de 10uF
- Un capacitor de 1000uF
- Dos capacitores de 22pF
- Tres capacitores de 0.1uF
- Un capacitor de 47uF/25v
- Un capacitor de 1000uF/50v
- Una resistencia de 820Ω
- Una resistencia de 3.3kΩ
- Cuatro resistencias de 10K Ω
- Diez resistencias de 470 Ω
- Dos resistencias de 4.7KΩ
- Una resistencia de 100KΩ
- Un potenciómetro de 20KΩ
- Un transistor 2N3904
- Un transistor TIP127
- Diez diodos 1N4007
- Un Max 232
- Dos registros TPIC6B595N
- Un buzzer
- Un sensor de temperatura DS18B20
- Una barra de 10 LEDs
- Un integrado DS1307
- Un PIC 18F4550
- Un driver L293B
- Un GLCD 128*64
- Dos motores DC 12V
- Una batería de 3V
- Un cristal de 12Mhz.
- Un cristal de 20Mhz.
- Cuatro switch
- Un jack
- Cuatro conectores
- Un zócalo de 40 pines
- Un pulsador
- Un zócalo de 8 pines
- Un zócalo Zip de 40 pines

Figura 2.17. Componentes electrónicos de la Tarjeta electrónica.

La construcción del presente circuito nos permite observar el resultado de diferentes aplicaciones como: la temperatura ambiente mediante el sensor, la hora del PC en el GLCD, el sonido del Buzzer, el desplazamiento de la luz en la barra de LEDs, el giro de los motores en diferentes sentidos y la lectura de ciertos parámetros que se visualizan en el GLCD.

El puerto USB usa 5V, el buzzer y los motores usan 9V, el reloj puede trabajar con 3V, el microcontrolador con 5V, los TPIC, diodos y así sucesivamente la mayoría de elementos usan 5V, entonces la fuente interna requerida es de 5V. Por lo tanto usamos una fuente swithchig de 7.5V con 1.2A de tal manera que el regulador de voltaje 7805 regule el voltaje a 5V para la fuente interna y entregue más de 5V a los dispositivos que usen más voltaje.
Cabe indicar que también se diseñó una fuente externa de 12V, en caso de que se desee colocar motores mayores a 5V.

Usamos un switch para manejar los motores DC y también para indicar que fuente estoy usando, la fuente interna o la fuente externa, de manera que la fuente externa es de 12V y puede variar hasta 36V para motores más grandes; pudiendo así cambiar el motor que desee.
También, basándonos en que el PIC18F4550 posee 5 pórticos bidireccionales (A, B, C, D y E), contiene 40 líneas o pines, 35 pines configurables como entradas o salidas de datos, los otros 5 pines son para polarizar el micro. Se han distribuido de la siguiente manera: 5 pines son para polarizar el circuito, 3 pines para configurar el cristal, 14 líneas para el GLCD, 1 línea para el buzzer, 6 líneas para los motores, 1 línea para el sensor, 10 líneas para la barra de LEDs, 2 líneas para el reloj, 2 líneas para el puerto RS232 y para el puerto USB; por lo tanto suman 44 líneas cuando el PIC solo tiene 40 pines, es por esta razón que usamos los TPIC o Shift Register ya que los TPIC usan 4 líneas del micro mientras que la barra de LEDs usa 10 líneas del micro de esta manera nos ayudan a ahorrar líneas y así no usamos otro microcontrolador ahorrando espacio en la tarjeta y dinero. El microprocesador toma las señales digitales y las procesa según el programa que esté en su memoria para más adelante enviarlas a los diferentes periféricos. Dichos datos son posibles visualizarlos en un computador con la

ayuda de la interfaz gráfica generada por Visual C # que tiene la capacidad de transmitir datos vía USB y serial RS232 a un dispositivo externo.

2.4.2 DESCRIPCIÓN DE LAS CONEXIONES DE CADA APLICACIÓN DE LA LIBRETA ELECTRÓNICA DE PRÁCTICAS Y TAMBIÉN COMO SE DISTRIBUYEN LOS PINES DEL PIC.

Podemos observar las conexiones de cada aplicación y como se distribuyen los pines del PIC para cada aplicación.

2.4.2.1 Diagrama de la fuente interna.

En la figura 2.18 se observa el diagrama de la fuente interna, nos provee de energía para alimentar el circuito con 5V y consta de los siguientes elementos:

- Un regulador de voltaje 7805
- Un capacitor de 10uF
- Una resistencia de 820Ω
- Un capacitor de 470uF/50V
- Un switch
- Un diodo 1N4007
- Un jack o connector
- Un LED

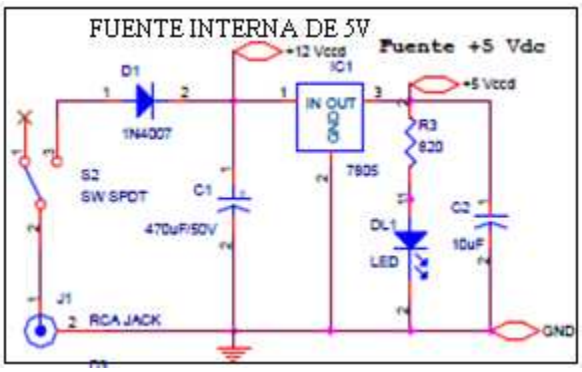


Figura 2.18.Conexión de la fuente interna.

2.4.2.2 Diagrama de los motores DC.

Para proteger el driver L293B de los picos de corriente inversa cuando arranca el motor se coloca diodos como se observa en la figura 2.19, además se observa el diagrama de conexión de los motores, tomamos el pórtico E y C, de los cuales

usamos 6 pines ya que son 2 motores de cc y estos son: Desde E0 hasta E2 y desde C0 hasta C2. Estos pórticos son entradas y salidas.

TRISE=%00000111

TRISC=%10000000

Y usamos los siguientes elementos:

- Dos motores DC
- Un driver L293B
- Ocho diodos 1N4007
- Dos conectores

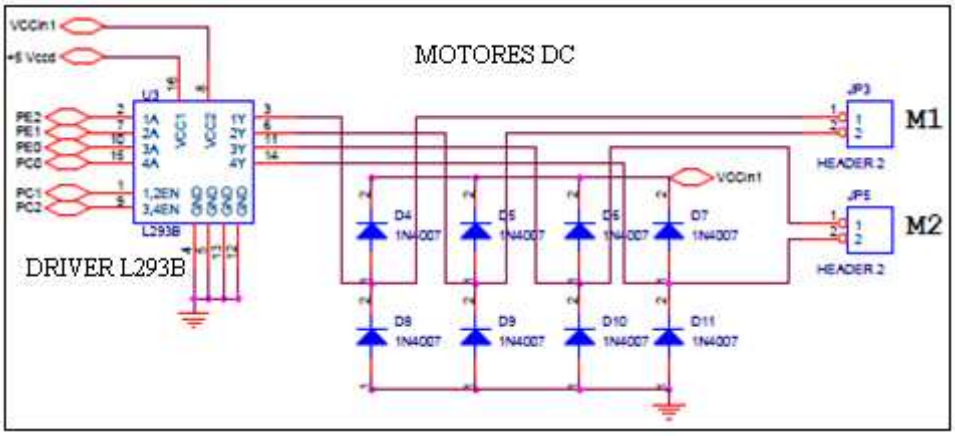


Figura 2.19. Conexión de los motores.

2.4.2.3 Diagrama de la fuente externa.

En la figura 2.20 se observa el diagrama de la fuente externa, es importante explicar que los motores además de las conexiones anteriores indicadas tienen otras conexiones que sirven como la fuente externa cuando se desee conectar motores mayores a 5 voltios.

Y usamos los siguientes elementos:

- Un capacitor de 1000uF/50V
- Dos switch
- Un diodo 1N5406
- Dos conectores o jack
- Un LED
- Una resistencia de 3.3KΩ

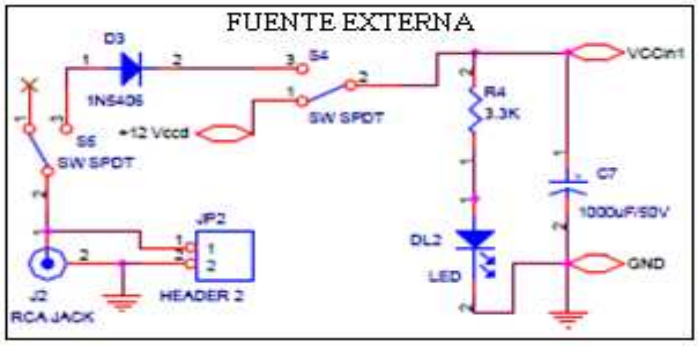


Figura 2.20. Conexión de la fuente externa.

2.4.2.4 Diagrama del sensor de temperatura.

Para el sensor de temperatura definimos el pórtico A4 como entrada y solo necesitamos un pin del PIC. TRISA=%00001000. En la figura 2.21 se muestra las conexiones del sensor. Y usamos los siguientes elementos:

- Una resistencia de 4.7KΩ
- Un sensor DS18B20

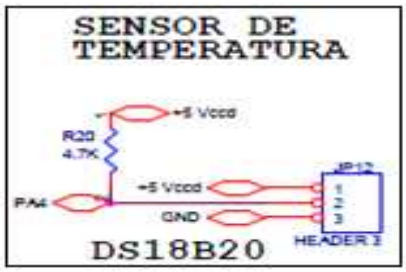


Figura 2.21. Conexión del sensor.

2.4.2.5 Diagrama de la comunicación USB.

Para la comunicación USB usamos dos pines del pórtico C: C6 y C7 y son el de TX y RX. En la figura 2.22 se aprecia el diagrama de la comunicación USB.

Y usamos los siguientes elementos:

- Un conector USB
- Cable para la comunicación

Figura 2.22.Conexión USB.

2.4.2.6 Diagrama de la comunicación serial RS232.

Para la comunicación serial RS232 necesitamos 2 pines del pórtico C: C6 y C7

En la figura 2.23 se observa el diagrama de la comunicación serial RS232. Y usamos los siguientes elementos:

- Cuatro capacitores de 10uF
- Un connector DB9
- Un Max 232

Figura 2.23. Conexión Serial RS232.

2.4.2.7 Diagrama del buzzer.

Para el buzzer se usa el pórtico C, del cual solo se usa y es el pórtico C4 que se define como salida. En la figura 2.24 se muestra el diagrama. TRISC=%1000000.

Y usamos los siguientes elementos:

- Un buzzer
- Una resistencias de 4.7KΩ
- Un transistor 2N3904

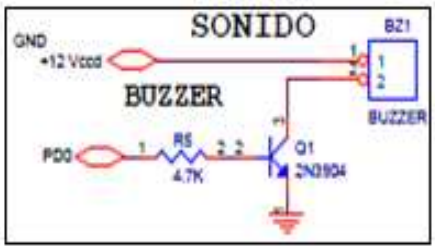


Figura 2.24. Conexión del buzzer.

2.4.2.8 Diagrama del RTC.

El diagrama del RTC se observa en la figura 2.25. Para el Reloj en tiempo real empleamos el pórtico A del PIC, de los cuales usamos los pórticos A0 y A1.

Y usamos los siguientes elementos:

- Un integrado DS1307

- Dos resistencias de 10K Ω
- Una resistencias de 820 Ω
- Un cristal de 12Mhz.
- Un LED
- Una batería de 3v

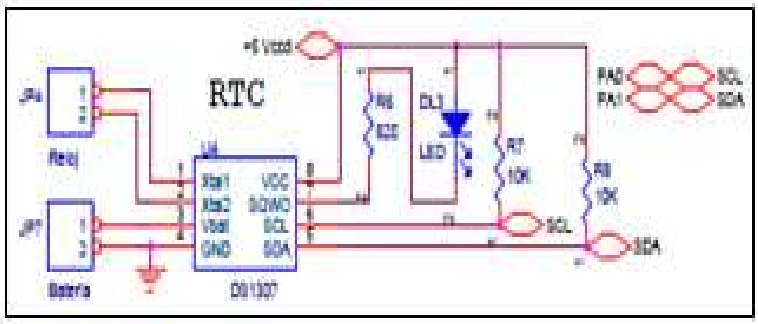


Figura 2.25.Conexión del RTC.

2.4.2.9 Diagrama del GLCD.

El diagrama del GLCD se muestra en la figura 2.26. El GLCD usa 14 pines para funcionar correctamente con el microcontrolador, de los cuales se usa todo el pórtico B, que tiene 8 pines, el pórtico D desde el pin D.2 hasta el pin D.7 que son 6 pines, en total los 14 pines.

Y usamos los siguientes elementos:

- Un GLCD
- Un potenciómetro de 25KΩ
- Un switch

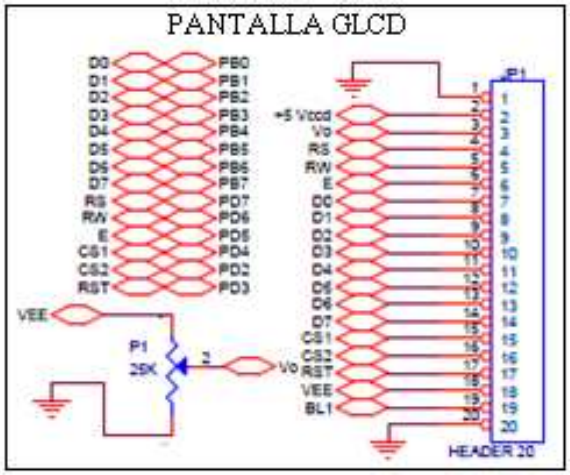


Figura 2.26. Conexión del GLCD.

2.4.2.10 Diagrama del TPIC6B595N

Para los TPIC usamos los pórticos A y D pero solo usamos 4 pines y estos son los pórticos A2, A3, A5 y D0: En la figura 2.27 se observa el diagrama de los TPIC. Y usamos los siguientes elementos:

- Dos capacitores de 0.1uF
- Dos TPIC6B595N

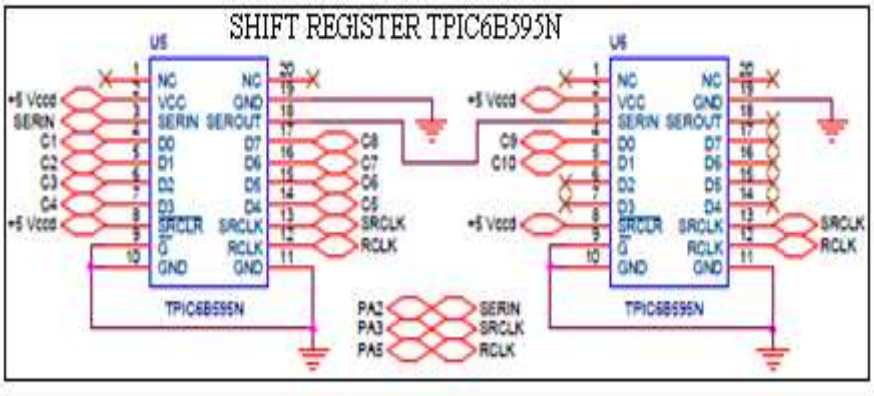


Figura 2.27. Conexión de los TPIC.

2.4.2.11 Diagrama de la barra de LEDs.

Para la barra de LEDs usamos el Pórtico D.1 que se conecta a la resistencia del transistor TIP127 y este a su vez a la barra. El diagrama se muestra en la figura 2.28.

Y usamos los siguientes elementos:

- Un barra de 10 LEDs
- Un transistor TIP127
- Una resistencias de 10KΩ
- Diez resistencias de 470 Ω

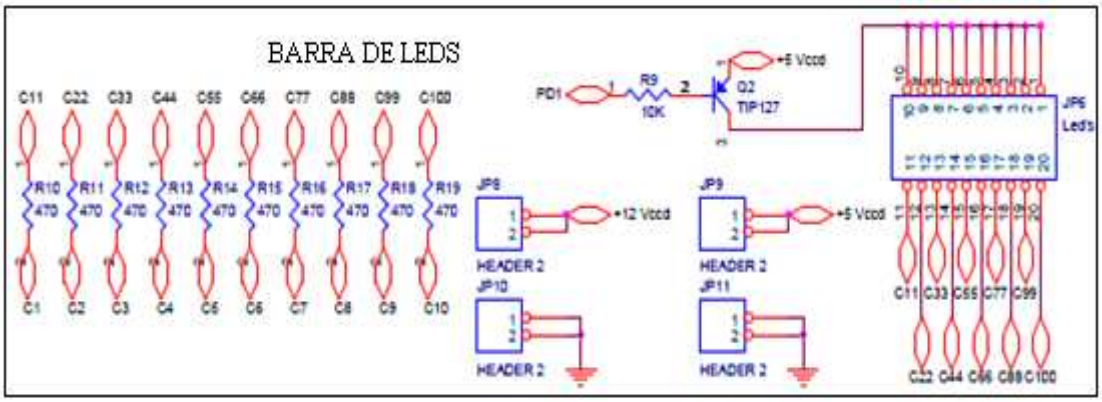


Figura 2.28. Conexión de la barra de LEDs.

2.4.2.12 Diagrama de las conexiones del PIC18F4550

En la figura 2.29 se muestra el diagrama del PIC18F4550 de 40 pines, de los cuales 32 ya se distribuyeron a diferentes aplicaciones que explicamos en diagramas anteriores, para la polarización del PIC se usan 5 pines y 3 pines para conectarse a un cristal como se indica en el diagrama, y el RESET es para el diagrama en general. Usamos los siguientes elementos:

- Un PIC 18F4550
- Una resistencia de 100KΩ
- Un diodo 1N4007
- Un capacitor de 10uF
- Un pulsador
- Un cristal de 20Mhz.
- Dos capacitores de 22pF
- Un capacitor de 47uF/25v

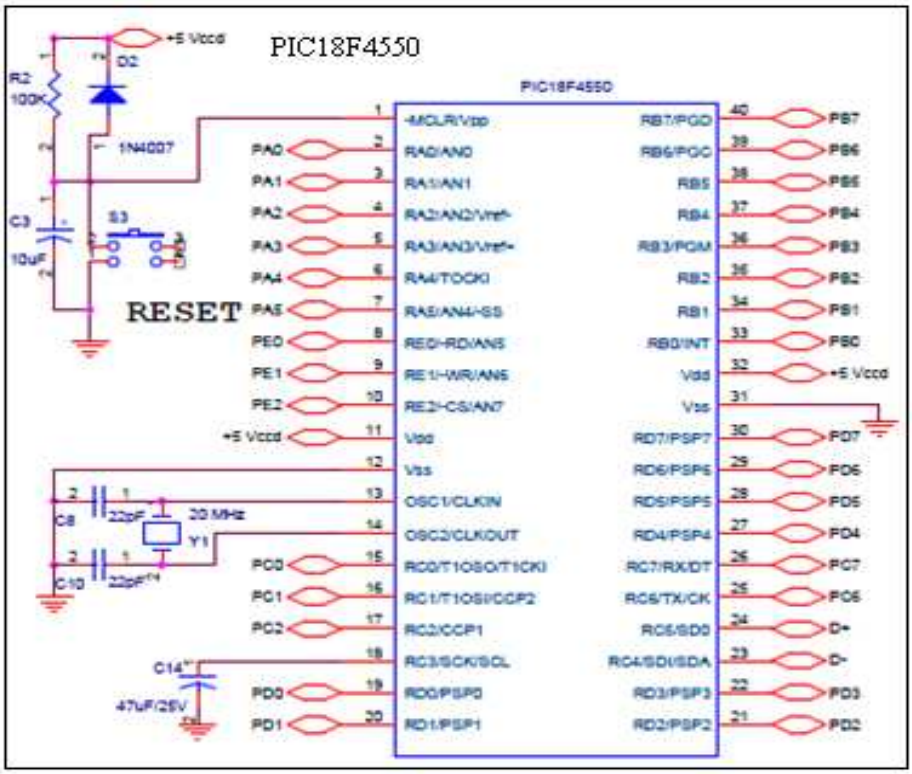


Figura 2.29. Conexión del PIC18F4550.

2.4.3 PRESENTACIÓN DEL DIAGRAMA LÓGICO.

Después de explicar las conexiones de cada aplicación presentamos el diagrama lógico en general de la Libreta Electrónica de Prácticas en la figura 2.30.

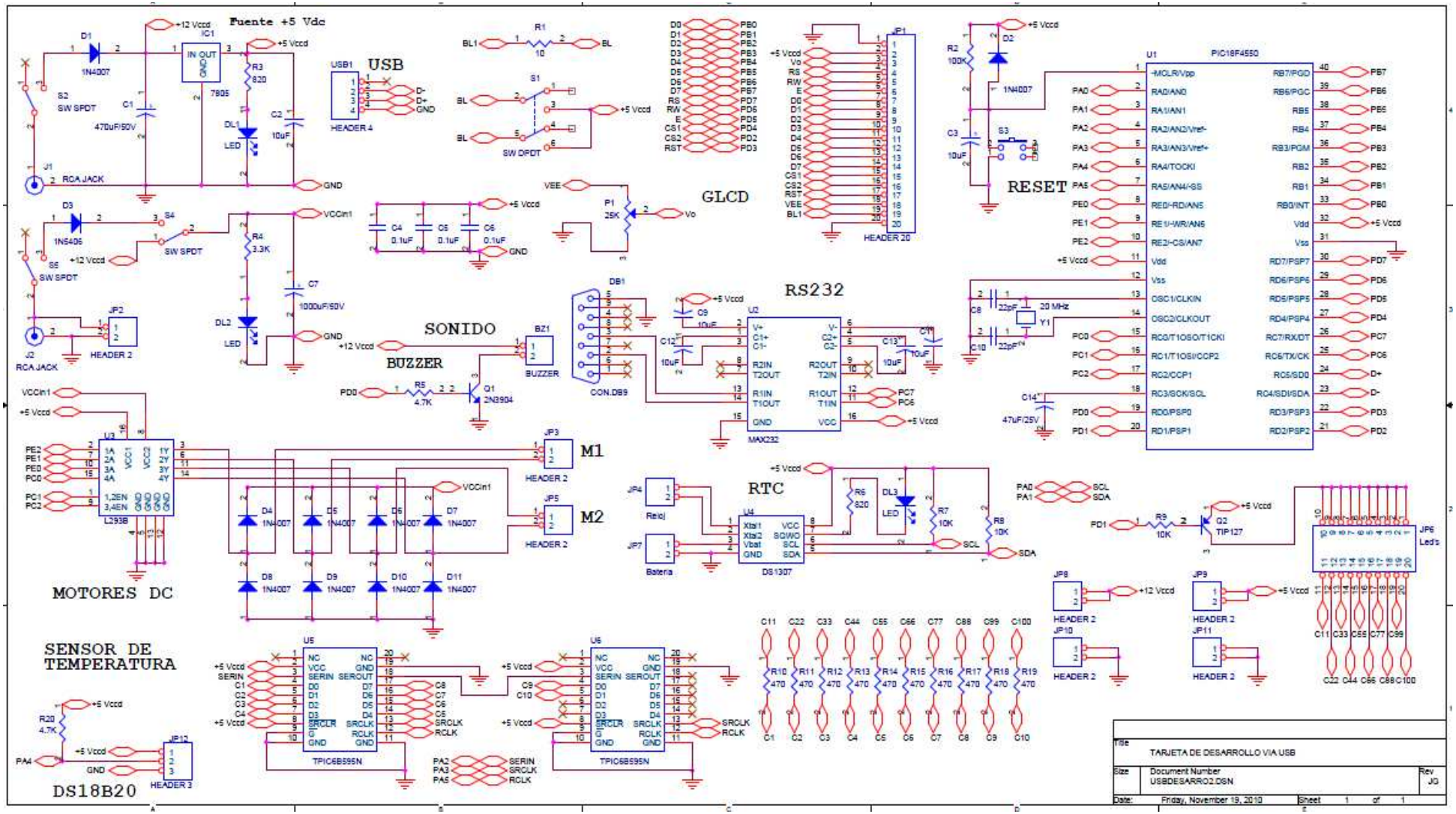


Figura 2.30. Diagrama lógico de La Libreta Electrónica de Prácticas.

Por lo tanto, queda ya finalizado el diagrama en general, el cual muestra los elementos distribuidos en el circuito tanto activos como pasivos, permitiéndonos comprender el funcionamiento del mismo.

El diagrama lógico de la Libreta Electrónica de Prácticas es de gran ayuda para realizar las conexiones y nos ayuda a visualizar los elementos en orden para no confundirnos con tantas líneas y partes que se acumulan conforme se va realizando el diagrama.

2.5 DESARROLLO DEL CIRCUITO IMPRESO

Para realizar el diagrama lógico y el ruteo de pistas del circuito usamos el programa llamado OrCAD, presentamos el entorno en la figura 2.31.

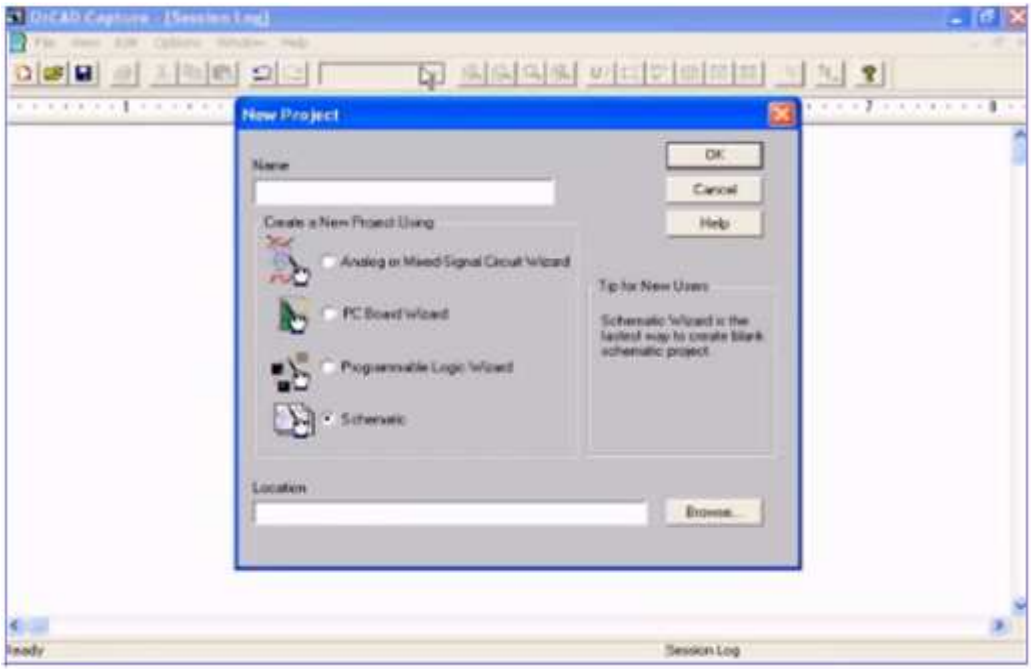


Figura 2.31 Ventana principal de Orcad Capture for Windows.

El Capture de OrCAD tiene un entorno de diseño electrónico potente que nos permite crear páginas y diseños esquemáticos, los diagramas se almacenan en una librería que pueden ser copiados y editados fácilmente posee una diversidad de librerías con muchos dispositivos.

El OrCAD Layout es una herramienta potente a la hora de trazar las pistas en el diseño de PCBs al disponer de muchas funciones, nos permite el ruteo de pistas del circuito de forma manual o automática, diseñado el diagrama lógico

este archivo se guarda con una extensión .DSN y el archivo del ruteo de pistas se guarda con una extensión .PBC

2.5.1 CIRCUITO IMPRESO DE LA LIBRETA ELECTRÓNICA.

A continuación en la figura 2.32 se observa el diagrama de pistas y en la figura 2.33 la distribución, la ubicación de cada elemento en la Libreta Electrónica de Prácticas, siendo estos diagramas muy útiles para ubicar los elementos y proceder a soldarlos en la placa.

Es muy importante antes de colocar los elemento en la placa verificar que no existan cortos circuitos en el diagrama de pistas.

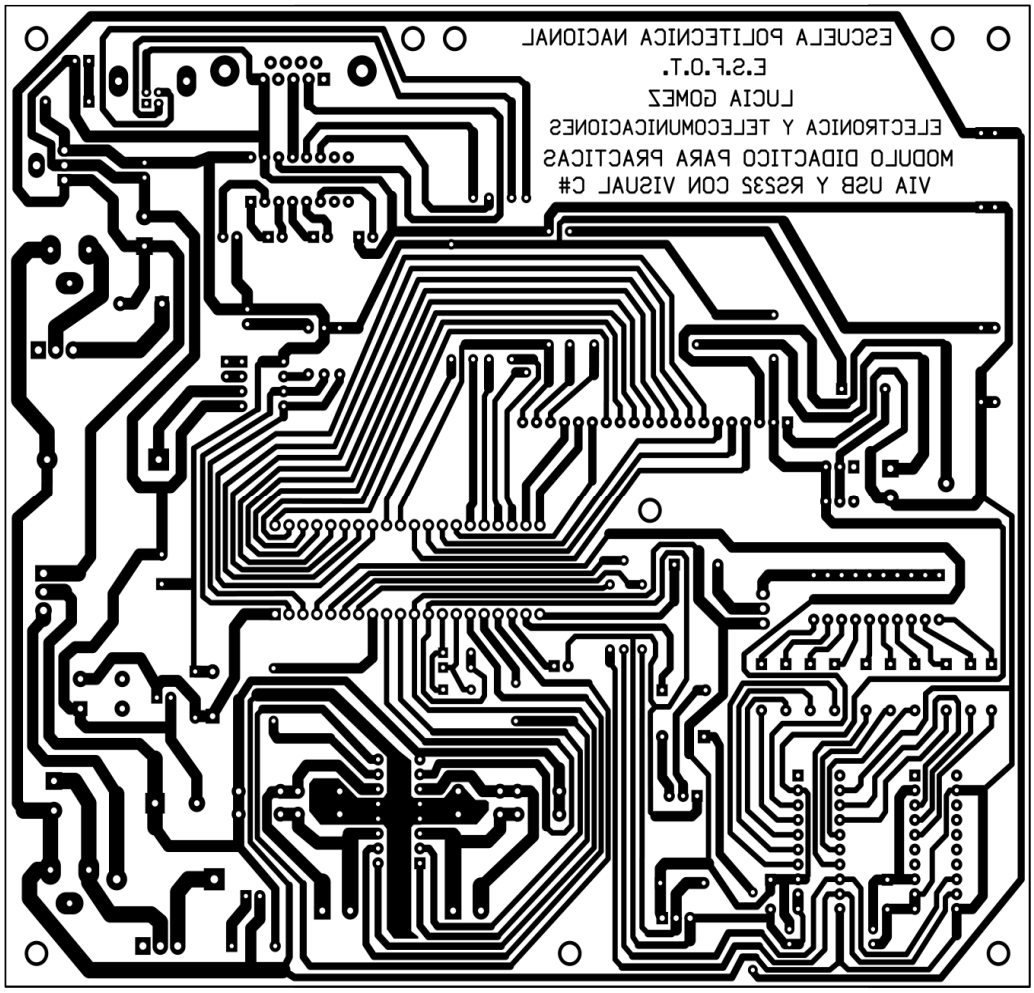


Figura 2.32. Pistas del circuito de la Libreta Electrónica de Prácticas.

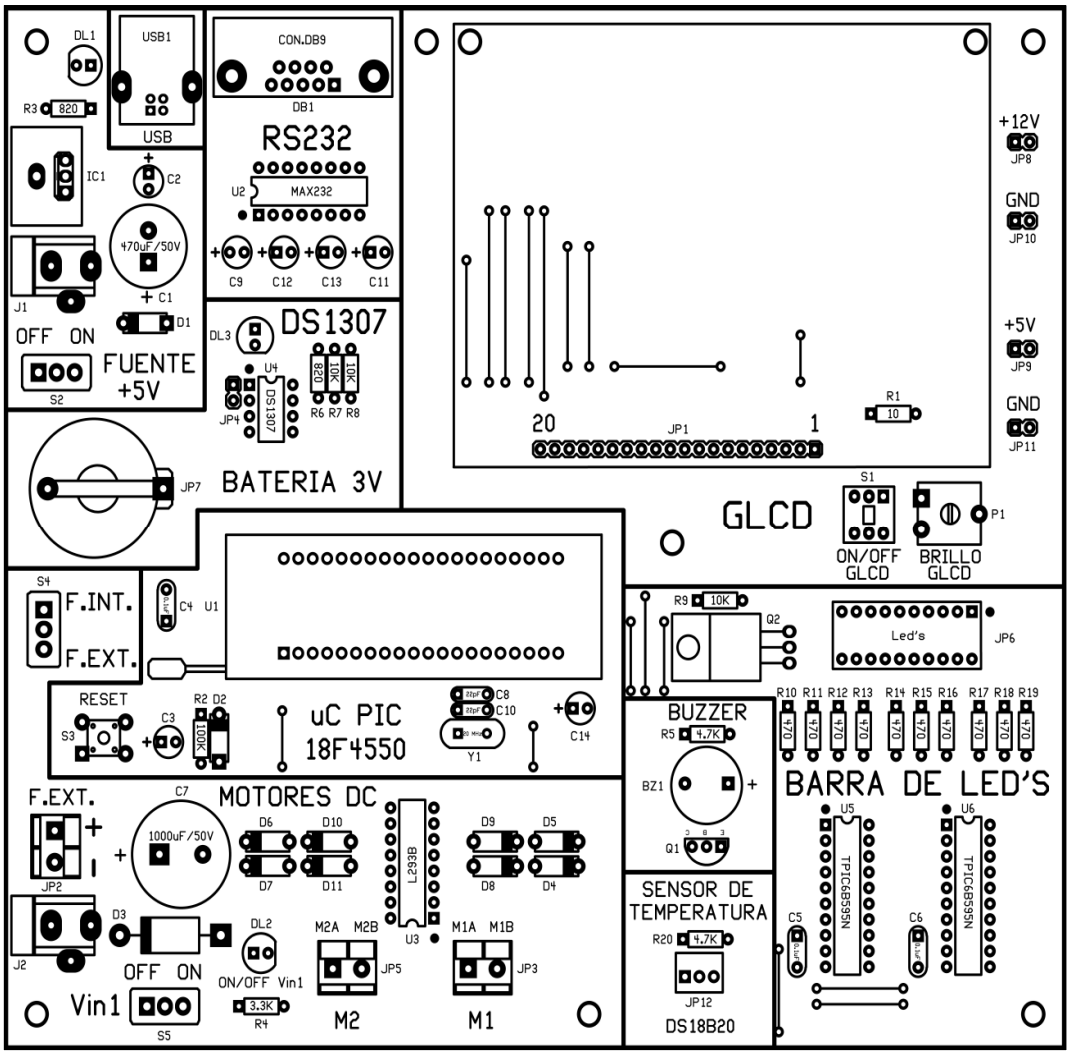


Figura 2.33. Distribución de los elementos en la Libreta Electrónica de Prácticas.

2.6 PRINCIPIO DE FUNCIONAMIENTO.

Una vez realizadas las conexiones de todo el circuito con mucha precaución procedemos a comprobar el funcionamiento de la Tarjeta Electrónica de Prácticas y observar cómo interactúa la interfaz gráfica, es decir verificar los datos enviados desde la PC hacia el PIC y luego con las diferentes aplicaciones.

2.6.1 PRESENTACIÓN DE LA LIBRETA ELECTRÓNICA DE PRÁCTICAS.

Figura 2.34. Libreta Electrónica de Prácticas

En la Libreta Electrónica de Practicas de la figura 2.34, podemos apreciar que consta de un integrado RTC que nos permite observar la hora del PC, en el GLCD podemos observar la lectura de la temperatura ambiente y las variaciones del ciclo de trabajo de los motores cuando giran, además posee un potenciómetro para disminuir o aumentar el brillo del GLCD y un interruptor para indicar si está encendida o apagada la pantalla. Funciona con la fuente interna de 5V y para motores más grades la fuente externa de 12V.

La Libreta Electrónica de Practicas propuesta consta de 5 aplicaciones que son: un GLCD, un sensor, un buzzer, una barra de LEDs y 2 motores DC y cada aplicación cumple una función específica. Los datos se envían desde la

PC mediante comandos por el puerto serial RS232 o el puerto USB para estos ser acondicionados es decir aquí se decodifican los comandos enviados desde la PC, para que en la etapa de procesamiento lleguen al PIC y puedan ser enviados a cada aplicación para finalmente ser observados.

GLCD: En el GLCD podremos apreciar los resultados de varias aplicaciones o los datos enviados desde la PC, es decir el texto que enviemos o leer la temperatura del sensor.

BUZZER: Es un dispositivo que emite sonidos en el cual podemos regular su volumen más intenso o menos intenso.

SENSOR: Mide la temperatura ambiente de -55℃ a 125℃ y de otros dispositivos, este valor lo podremos observar en el GLCD.

MOTORES DE CC: Realizan giros en el sentido horario y anti horario además podemos aumentar o disminuir la velocidad de los mismos modificando la programación.

BARRA DE LEDS: En esta barra de LEDs podemos observar cómo se desplaza un 1L o 0L en cada LED de izquierda a derecha y viceversa, también se puede seleccionar un número decimal en la interfaz gráfica y observar en el GLCD en binario.

2.6.2 DESCRIPCIÓN DE LA INTERFAZ.

Como se aprecia en la figura 2.35, la Libreta tiene varios botones los cuales son muy útiles para manejar la Interfaz Gráfica.
Primeramente debemos escoger la forma de comunicación que vamos a usar con solo presionar el botón RS232 ó USB y estaremos listos para continuar, en la parte derecha de la interfaz gráfica podemos observar los botones que son para ingresar texto pero con cierta restricción de 20 caracteres debido a que el GLCD solo puede mostrar esta cantidad de caracteres en una línea.

En los botones PWM1, PWM2 de los motores podemos cambiar los valores de mínimo a máximo o viceversa para que estos giren más rápido o lentamente con el sentido de giro que deseemos, al presionar el botón de buzzer escucharemos un sonido agudo inmediatamente, el botón ON es para encender y OFF para apagar el Buzzer, al hacer contacto con el sensor observaremos en el GLCD cómo se eleva la temperatura.

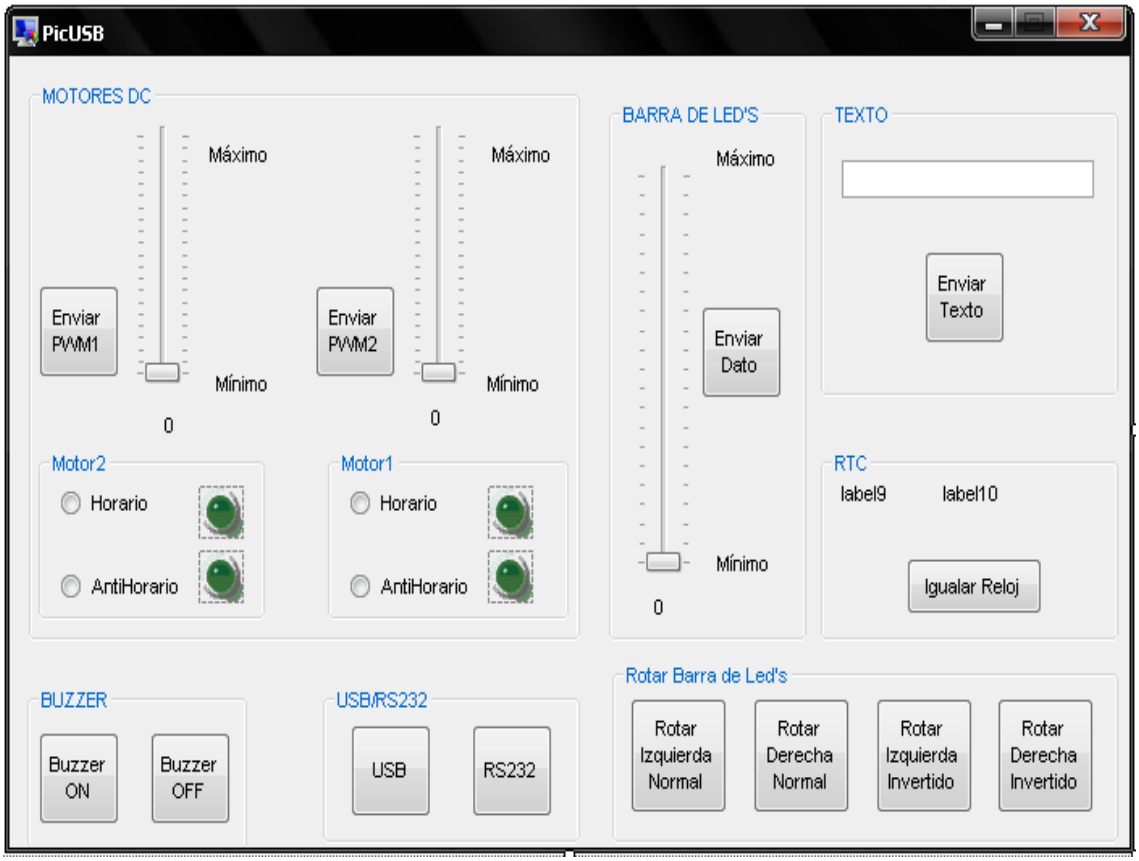


Fig. 2.35 Interfaz gráfica.

En la barra de LEDs tenemos varios botones uno que nos indica enviar dato con el cual la barra de LEDs se encuentra lista enviar el dato, luego podemos presionar cualquier botón para rotar la luz dependiendo del sentido que se desee. Finalmente tenemos un botón para ajustar el reloj en hora y fecha el cual es de mucha ayuda, su lectura se puede observar en el GLCD.

En este diagrama no está aún terminada la interfaz gráfica pero funciona correctamente, más adelante la presentaremos con todos los detalles y el entorno más agradable.

3. CAPÍTULO III: LENGUAJE DE PROGRAMACIÓN.

3.1 INTRODUCCIÓN.

En este capítulo se detallará paso a paso los programas autoguardados en el PIC mediante CCS y el programa general del Software realizado en Visual C #. El programa de cada aplicación desarrollado en CCS se encuentra grabado en el PIC 18F4550 y funciona de manera que desde la PC enviamos el dato que llega al PIC, y elegimos la aplicación que deseamos observar en el GLCD o cualquier otra aplicación.

El proyecto tiene 2 formas de trabajar la primera es usando el Puerto Serie RS232, aunque este se considera que es un puerto heredado y obsoleto lo hemos tomado muy en cuenta debido a que los puertos serie todavía se encuentran en uso; en sistemas de automatización industrial y algunos productos industriales y de consumo. Los dispositivos de redes, como los enrutadores y conmutadores, a menudo tienen puertos serie para modificar su configuración. Los puertos serie se usan frecuentemente en estas áreas porque son sencillos, baratos y permiten la interoperabilidad entre dispositivos y no incluyen el puerto USB; la otra forma de trabajar es mediante el puerto USB ya que este poco a poco ha reemplazado al puerto serie RS232 puesto que es más rápido. La mayor parte de las computadoras están conectadas a dispositivos externos a través de USB y, a menudo, ni siquiera traen un puerto serie RS232.

3.2. HERRAMIENTAS COMPUTACIONALES.

Existe diversidad de software dedicados a trabajar con microcontroladores en el mercado a los cuales podemos acceder fácilmente mediante internet con licencias gratis, debemos tomar en cuenta qué programas hemos aprendido y en qué forma son útiles para el diseño de nuestras aplicaciones. Para determinar el tipo de software que usamos nos basamos en las características de nuestro proyecto que usa comunicación USB y RS232.

Una vez analizados estos puntos hemos concluido trabajar con el siguiente software de Programación.

3.2.1 COMPILADOR CCS.

El compilador CCS (Custom Computer Services), se trata de un compilador de alto nivel, muy eficiente, el cual nos ayudará para depurar el código, ya que nos indica los diferentes errores que podamos haber cometido. Contiene bibliotecas muy completas, permite una combinación del lenguaje de alto nivel y Ensamblador, por si deseamos ajustar el código y muchas otras características. En la figura 3.1 se observa la ventana de CCS para crear un proyecto nuevo.

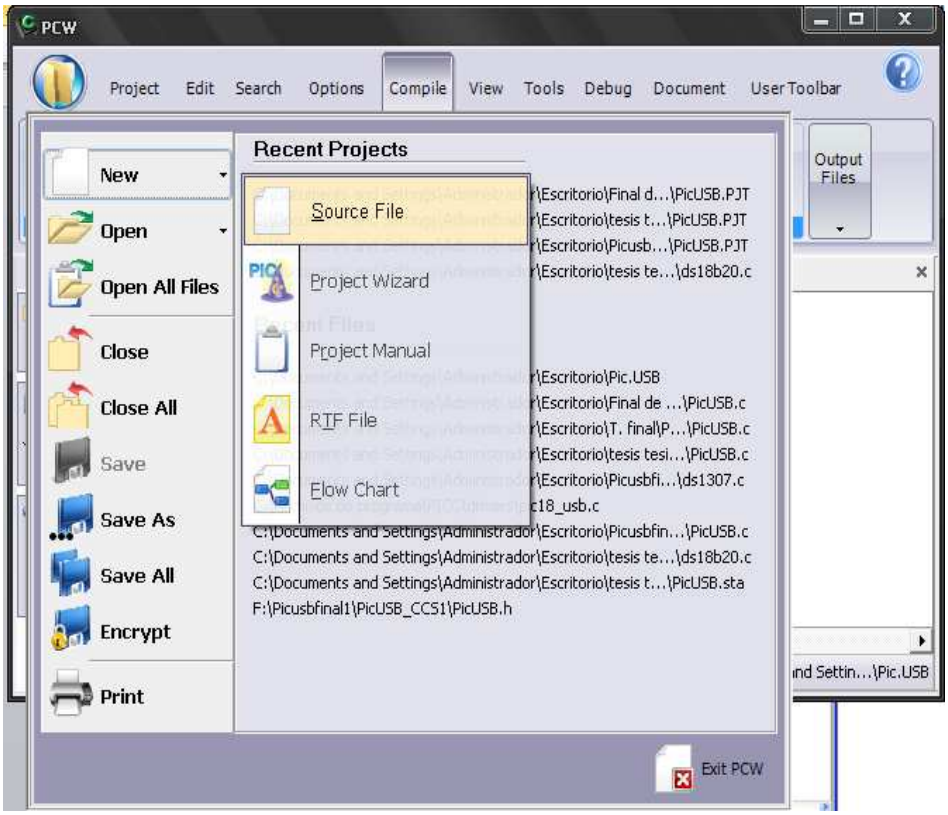


Figura 3.1 ventana de CCS para crear New Proyect.

En la figura 3.2 se muestra la ventana de PCW, el entorno en el que se va a desarrollar el programa.

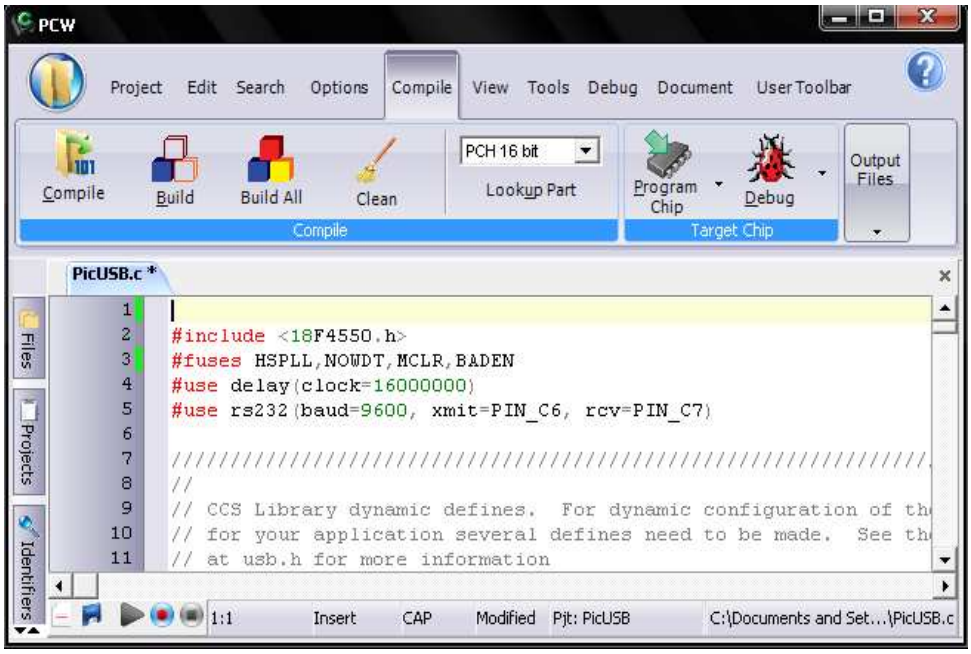


Figura 3.2 Ventana principal de CCS.

Para trabajar en CCS abrimos el programa damos clic en Project luego en new y clic en source file, y observamos la ventana de trabajo. En este programa podemos encontrar librerías con ejemplos los cuales son muy útiles, una vez realizado el programa lo compilamos y nos indica cuántos errores tenemos o si está correcto el programa desarrollado. El programa desarrollado se encuentra en el Anexo G.

También es posible verificar qué porcentaje del microcontrolador PIC que estemos usando, está utilizado por nuestro programa lo que nos ayuda a ver el espacio que nos sobra. CCS es un software que nos permite desarrollar los programas en lenguaje C y su librería incluye el PIC 18F4550 que es el que vamos a usar en el proyecto, como incluye la librería para PICs nos ayuda a conectarnos con el PC mediante USART y USB.

Además este software contiene un set de instrucciones que son compatibles con Visual C#. Si queremos realizar la programación de los microcontroladores PIC en un lenguaje como el C, es preciso utilizar un compilador de C.

Dicho compilador nos genera ficheros en formato Intel-hexadecimal, que es el necesario para programar (utilizando un programador de PIC) un microcontrolador de 6, 8, 18 ó 40 patillas. El compilador de C que vamos a utilizar es el PCW de la casa CCS Inc. A su vez, el compilador lo integraremos

en un entorno de desarrollo integrado (IDE) que nos va a permitir desarrollar todas y cada una de las fases que componen un proyecto, desde la edición hasta la compilación pasando por la depuración de errores. La última fase, a excepción de la depuración y retoques hardware finales, será programar el PIC. Al igual que el compilador de Turbo C, éste "traduce" el código C del archivo fuente (.C) a lenguaje de máquina para los microcontroladores PIC, generando así un archivo en formato hexadecimal (.HEX).

3.2.2 VISUAL C#.

Para trabajar en Visual C# abrimos el programa, damos clic en Archivo, clic en nuevo proyecto y nos aparece la ventana que se observa en la figura 3.3. Escogemos la plantilla con la que deseamos crear el proyecto la más usada es Aplicación de Windows Forms y damos clic en aceptar y estamos listos para trabajar. En la figura 3.4 se observa la ventana principal de Visual C#, en la parte izquierda de la ventana tenemos en cuadro de herramientas o formulario del cual podemos escoger cualquier botón según nuestra conveniencia y lo arrastramos al centro y enseguida en la parte derecha de la ventana que se despliega otra ventana llamada propiedades, esta ventana es muy importante ya que aquí escribimos lo que queremos hacer, ponemos nombre, color al texto, etc. Una vez listo el programa lo compilamos dando clic en depurar y listo eso es todo.

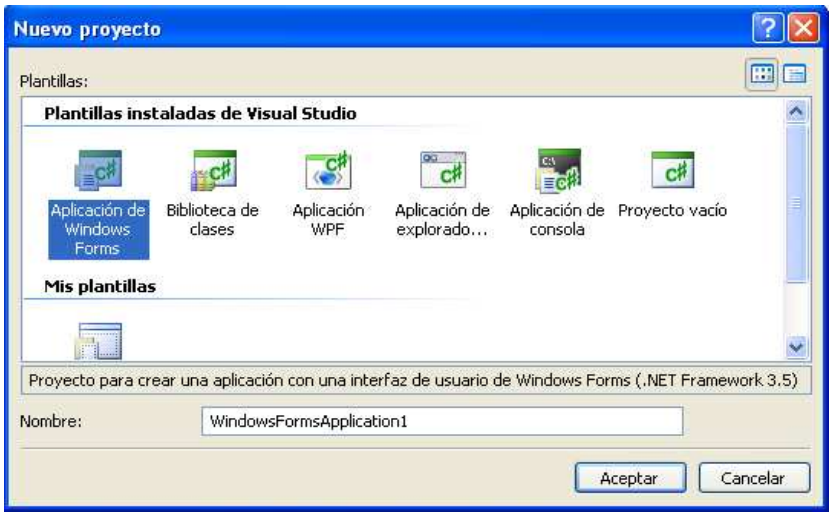


Figura 3.3 Ventana de Visual C# para crear New Proyect.

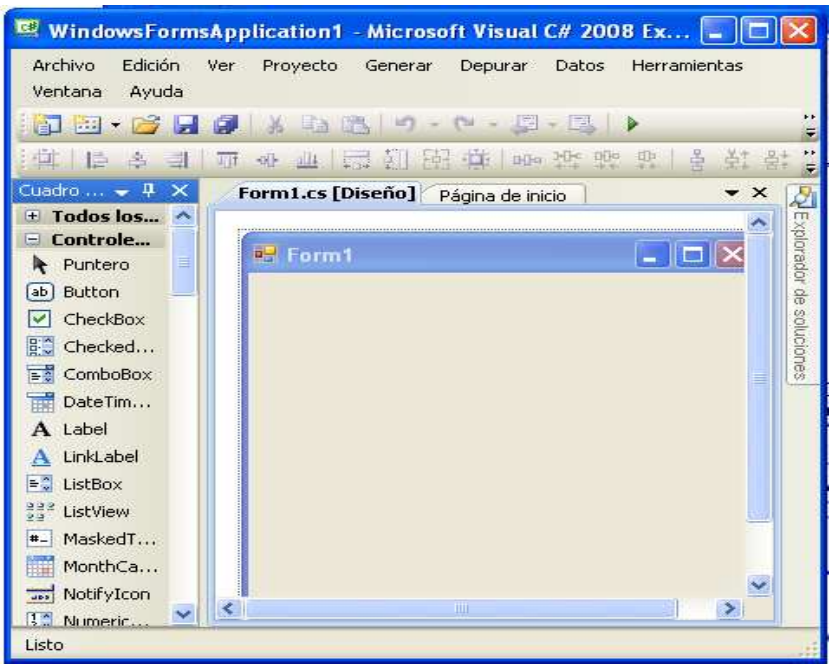


Figura 3.4 Ventana principal de Visual C#.

Microsoft Visual C# es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos. Con sus diversas innovaciones, C# desarrolla aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Visual Studio admite Visual C# con un editor de código completo, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz y fácil de usar, además de otras herramientas. La biblioteca de clases .NET Framework ofrece acceso a una amplia gama de servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa.

La creación del nombre del lenguaje, C#, proviene de dibujar dos signos positivos encima de los dos signos positivos de "C++", queriendo dar una imagen de salto evolutivo, del mismo modo que ocurrió con el paso de C a C++.

3.2.3 WIN PIC 800.

Winpic800 es un programa grabador de microcontroladores muy usado y fácil que puedes descargar libremente del Internet. Este software es útil para grabar el archivo .hex en la memoria del PIC18F4550. Su ventana principal se observa en la figura 3.5.

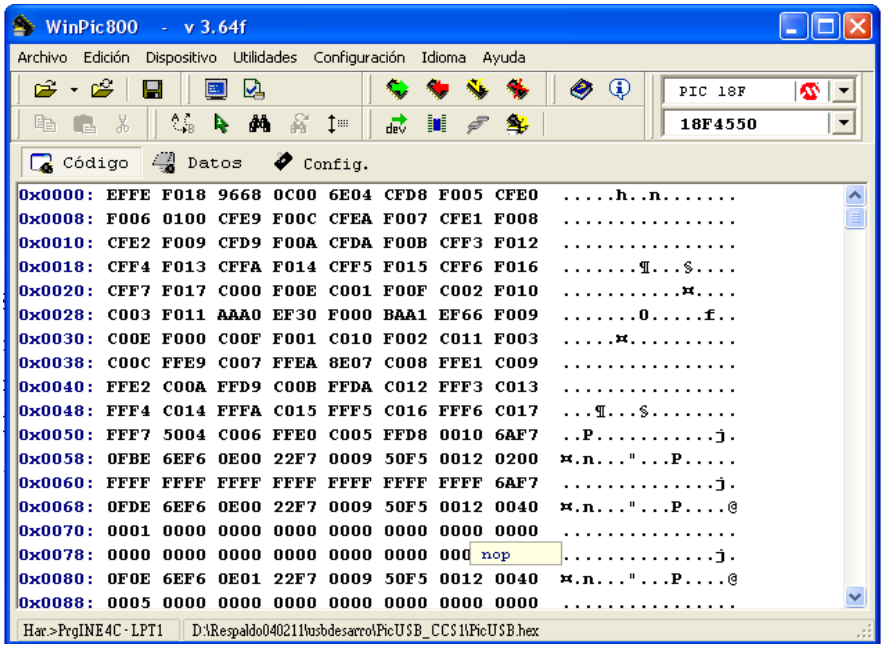


Figura 3.5 Ventana principal del programa WinPic800.

Este software genera automáticamente la configuración que se necesita, al escoger el PIC deseado, por lo que no debemos olvidar que el PIC utiliza el PLL interno y la opción USBPLL deberá estar activada. En la figura 3.6 se observa los campos que se tienen que activar para su funcionamiento.

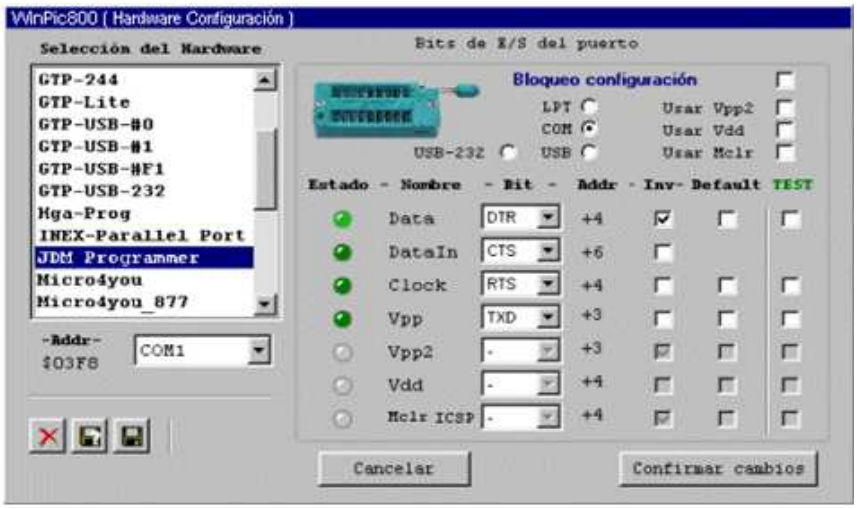


Figura 3.6 Configuración para grabar el PIC.

3.3 DESARROLLO DEL PROGRAMA PARA EL MICROCONTROLADOR 18F4550.

El código del programa está escrito en Lenguaje Visual C#, este código es compilado por el programa CCS el cual genera el archivo .hex necesario para poder grabar en el PIC.

3.3.1 COMUNICACIÓN USB PIC CON EL COMPUTADOR.

Es necesario generar un código dentro del programa para establecer la comunicación entre el PIC y el computador el código mencionado anteriormente es generado siguiendo los siguientes pasos:

Es importante aclarar que los diferentes software como Visual C# y CCS tienen comunicación RS-232 y comunicación USB.

- Para tener acceso a la comunicación USB en Visual C # usamos un programa llamado Protón el cual nos permite generar un código que es compatible con Visual C#.

Abrimos Protón IDE, ejecutamos el comando EasyHid USB Wizard, este nos permitirá generar el código requerido para establecer la comunicación USB entre la Tarjeta Electrónica y el PC. En la figura 3.7 se observa la ventana de Protón IDE.

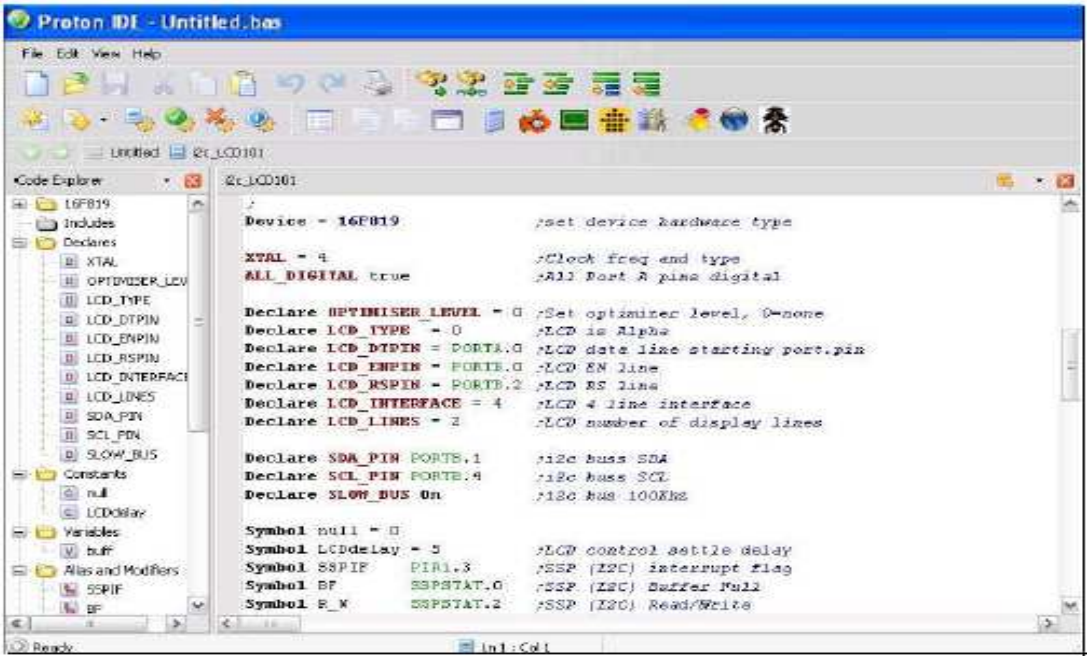


Fig.3.7 Comando utilizado para generar la comunicación USB.

Luego procedemos a dar un nombre con el que deseamos generar el archivo para establecer la comunicación, el que se observa en la figura 3.8.

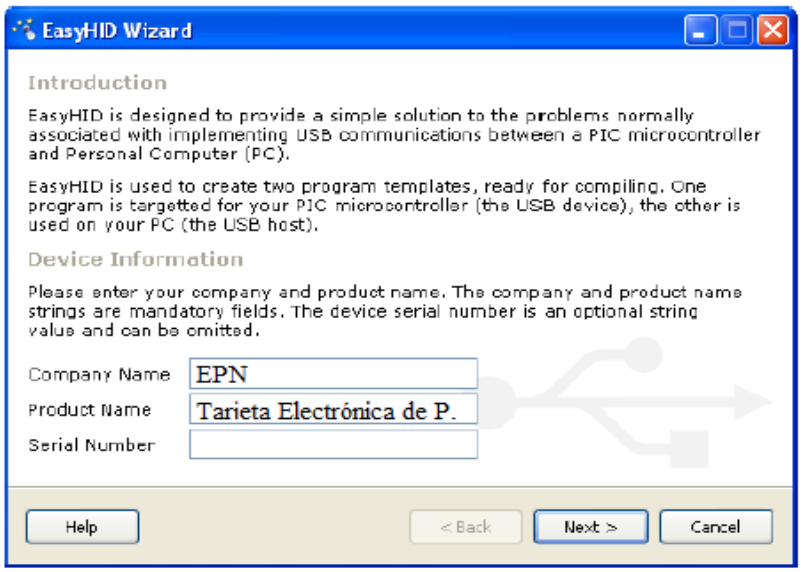


Fig.3.8 Nombre con el que se genera el archivo para establecer la comunicación USB.

Luego escogemos la identificación del producto y del vendedor, los valores tienen un rango, pero es preferible mantenerlos por default esto se muestra en la figura 3.9.

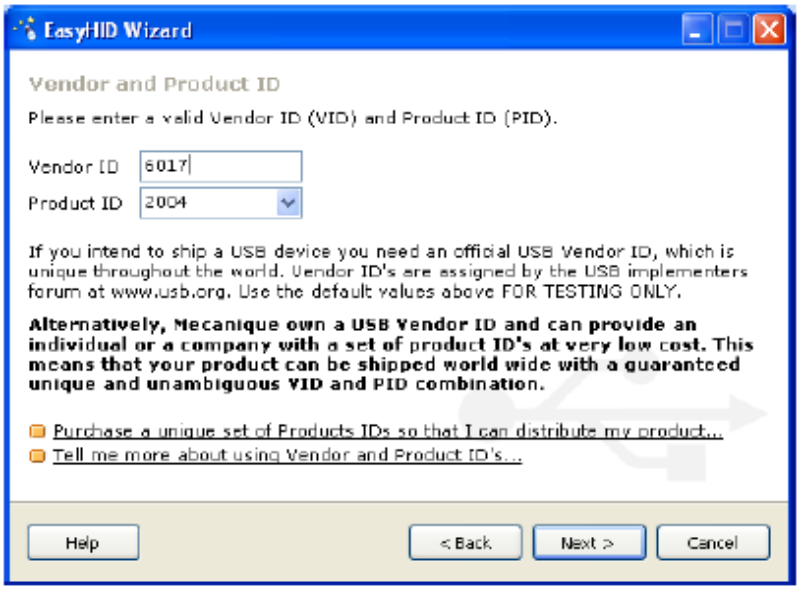


Fig.3.9 Identificación del producto y el vendedor.

Continuamos configurando los parámetros de tiempos requeridos y los buffer de entrada y salida necesarias para la transmisión y recepción de datos. Como se muestra en la figura 3.10. Es recomendable usar los valores por default.

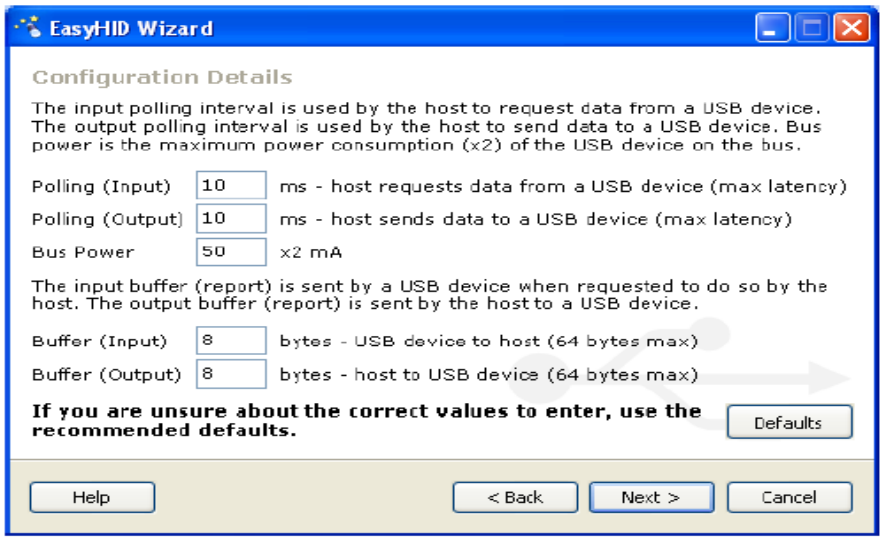


Fig.3.10 Configuración de buffers, para la TX y RX de datos.

En la figura 3.11 se muestra la ventana donde debemos escribir el nombre del archivo y el lugar donde se desea guardar. También debemos escoger el PIC, el compilador donde se generará la interfaz gráfica. Es importante habilitar la casilla para generar interrupciones USB ya que pueden presentarse errores de TX y RX si no se habilita esta casilla.

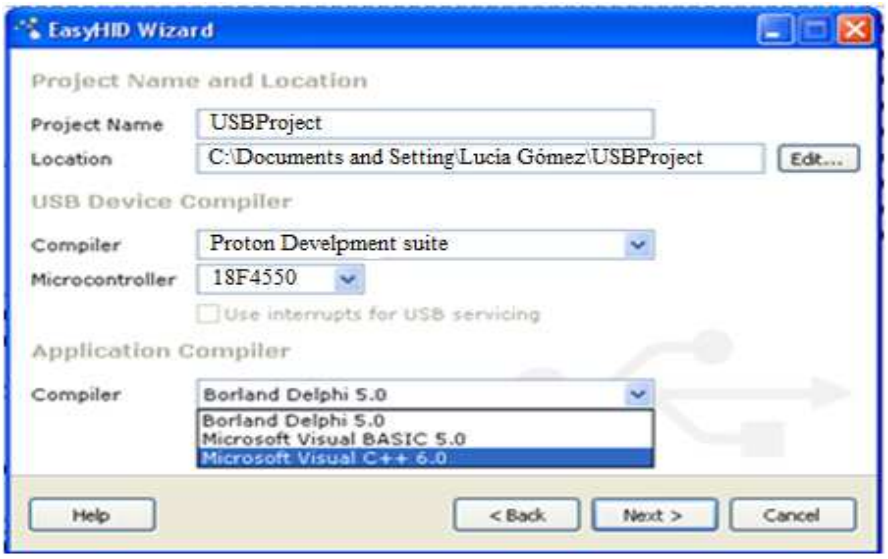


Fig.3.11 Nombre y dirección del archivo, PIC y compilador.

Si damos clic en el botón siguiente aparece una ventana, la misma se cerrará automáticamente si no se producen errores. Cuando abrimos la carpeta donde se guardó el archivo, encontraremos dos nuevas carpetas la una como PDS y la otra como Visual C, esto se muestra en la figura 3.12.

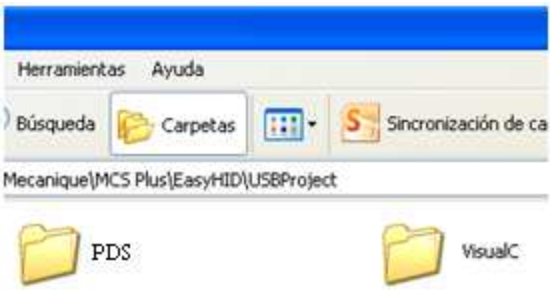


Fig.3.12 Carpetas generadas por el EasyHid Wizard USB.

En la figura 3.13 se muestra el contenido de la carpeta Visual C, encontramos un archivo con la terminación .dsp el cual genera un código para poder comunicarnos mediante el USB, este archivo es compatible con Visual C# el que nos permite realizar conversiones desde el archivo.dsp a Visual C#. Teniendo este código lo usamos para ajustarlo a nuestras necesidades y desarrollar nuestro programa.

Nombre	Tamaño	Tipo	Fecha de modific
Debug		Carpeta de archivos	4/4/2011 11:37
res		Carpeta de archivos	4/4/2011 11:37
mcHID.cpp	3 KB	C++ Source	4/4/2011 11:37
mcHID.h	3 KB	C Source File	4/4/2011 11:37
ReadMe.txt	4 KB	Documento de texto	4/4/2011 11:37
Resource.h	1 KB	C Source File	4/4/2011 11:37
StdAfx.cpp	1 KB	C++ Source	4/4/2011 11:37
StdAfx.h	2 KB	C Source File	4/4/2011 11:37
USBProject.clw	2 KB	Archivo CLW	4/4/2011 11:37
USBProject.cpp	3 KB	C++ Source	4/4/2011 11:37
USBProject.dsp	5 KB	VC++ 6 Project	4/4/2011 11:37
USBProject.dsw	1 KB	VC++ 6 Workspace	4/4/2011 11:37
USBProject.h	2 KB	C Source File	4/4/2011 11:37
USBProject.rc	6 KB	Resource Script	4/4/2011 11:37
USBProjectDlg.cpp	8 KB	C++ Source	4/4/2011 11:37
USBProject.h	2 KB	C Source File	4/4/2011 11:37

Fig.3.13 Archivos.dsp donde se almacenará el código para el programa en el software Protón.

Luego podemos apreciar en la figura 3.14, una nueva ventana que está lista para trabajar en Visual C# la cual nos indica que no contiene errores de compatibilidad. Una vez lista podemos diseñar la interfaz gráfica del computador.

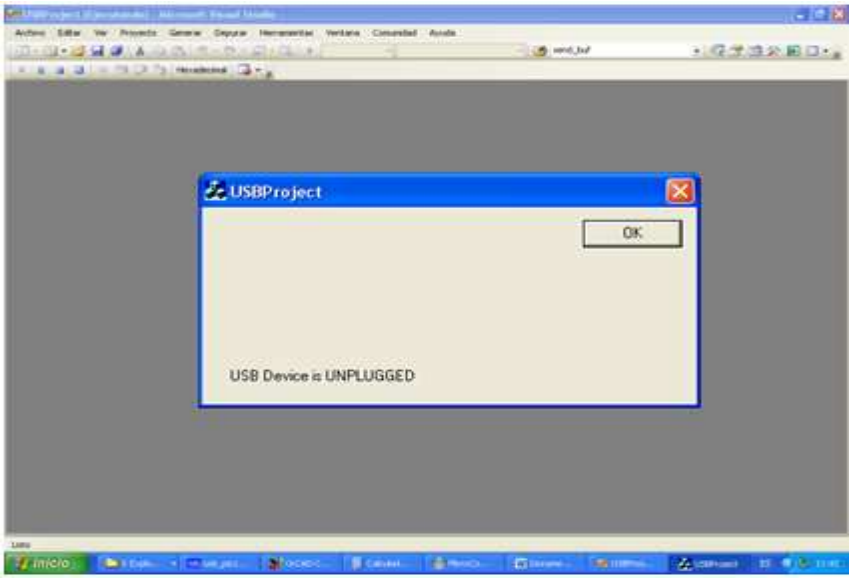


Fig.3.14 Ventana en Visual C# lista para trabajar.

- Y para tener acceso a la comunicación RS-232 en CCS usamos un ejemplo del mismo programa CCS y los vamos modificando a nuestras necesidades hasta obtener los resultados esperados.

3.3.2 DEFINICIÓN DE LOS PUERTOS.

Periféricos que se conectan al Microcontrolador.

Periféricos de entrada (Entra información al microcontrolador):

- Sensor

Periféricos de salida (Sale información del microcontrolador):

- GLCD
- Motor DC
- Buzzer
- Barra de LEDs.

Transmisión y Recepción de Datos:

- USB y Serie RS232.

El PIC 18F4550 tiene 5 Pórticos bidireccionales (A, B, C, D y E).

Pórtico A.- Está conformado de 6 líneas bidireccionales (A0, A1, A2, A3, A5) Para poder utilizarlas como líneas digitales de E/S hay que desactivar la

función analógica ya que por default viene definido como analógico. La línea A4 es digital y puede ser administrada por el usuario.

Pórtico B.- Está conformado de 8 líneas de entrada y salida, permite al usuario ejecutar interrupciones externas como es el GLCD, LCD, etc.

Pórtico C.- Está conformado de 7 líneas, es usado para las comunicaciones desde C0, C1, C2, C4, C5, C6 y C7, son entradas y salidas.

Pórtico D.- Este pórtico posee 8 líneas de entrada y salida.

Pórtico E.- Dispone de 3 líneas de entrada y salida (E0, E1, E2) y 1 línea E3 es solo de entrada.

Antes de realizar la programación debemos definir en el PIC qué puertos van a ser usados como entradas y como salidas. Como vamos a usar el PIC 18F4550 debemos saber que vamos a usar 14 pines para el GLCD, 1 pin para el buzzer, 6 pines para los motores DC , 1 pin para el sensor, 1 pin para la barra de LEDs, 2 pines para el Reloj, 2 pines para la conexión USB/ RS232 y 4 pines para los TPIC.

No debemos olvidar que si se asigna un cero (0) a un pin este quedará como salida y si se asigna un uno (1) a un pin este quedará como entrada.

EL pórtico A está configurado como salidas, a excepción del pin A4 que es entrada ya que se conecta el sensor de temperatura; por lo que podemos definir lo siguiente:

TRISA=%00010000

Podemos observar de derecha a izquierda que los Puertos A5, A3, A2, A1, A0 están definidos como salidas. A excepción del Puerto A.4 que es el único, que está definido como entrada, debido a que este pin está destinado para trabajar con el sensor.

En el pórtico B todos los pines se definen como salidas, por lo tanto son 8 0L. TRISB=%00000000.

En el pórtico C tenemos 6 salidas y una entrada, por lo tanto hay un 1L y 6 0L. Podemos observar de derecha a izquierda que existen siete 0L, desde el puerto C6, C5, C4,C2, C1, C0 que son configurados como salidas y un 1L que

es el puerto C7 que se encuentra configurado como entrada. Quedando el Registro TRISC así:
TRIS C=%10000000

3.3.2.1 Pines para el GLCD

Anteriormente definimos que el GLCD usa 14 pines para funcionar correctamente con el microcontrolador, de los cuales están configurados todo el pórtico B, que tiene 8 pines y el pórtico D, desde el pin D.2 hasta el pin D.7 que son 6 pines, en total los 14 pines que requiere el GLCD. Y los pórticos D.0, A.2, A.3, A5 para el TPIC6B595N, y el pórtico D.1 para la barra de LEDs.
Estos puertos se encuentran configurados en el programa de la siguiente manera:
TRISB=%00000000
TRISD=%00000000
Como sabemos que el GLCD es un dispositivo de salida entonces todos los 14 puertos están definidos como salidas con 0L.

3.3.2.2 Pines para la barra de LEDs.

Usamos solo el Puerto D.1 debido a que usamos un TPIC, el TPIC usa 4 pórticos del PIC y son D.0, A.2, A.3, A5; este circuito integrado puede reducir varias entradas a una sola entrada lo que nos permite ahorrar los puertos del PIC, este es uno de los pórticos que quedaron libres cuando usamos el GLCD y está definido como salida quedando configurado de esta manera.
TRISD=%00000000

3.3.2.3 Pines para los Motores.

Para los motores usamos los puertos E y C, de los cuales usamos para los motores 6 pines y estos son: Desde E0 hasta E2 y desde C0 hasta C2, anteriormente ya definimos qué pórticos son entradas y salidas.
TRISE=%00000000
TRISC=%10000000

3.3.2.4 Pin para el Sensor.

Una vez adquirido físicamente el sensor de temperatura y luego de revisar las características técnicas del mismo, se conoce que la temperatura se medirá en el rango de -55 ºC a 125 ºC. Y como sabemos que el sensor es digital entonces definimos el pórtico A4 como entrada y solo necesitamos un pin del PIC. TRISA=%00010000

3.3.2.5 Pin para el Buzzer.

Para el buzzer se usa el pórtico C, del cual solo un pin se conecta y es el pórtico C4 el cual se define como salida. TRISC=%1000000

3.3.3 DEFINICIONES PARA EL GLCD.

Aquí se puede comprender que los datos se van a enviar al GLCD, mediante los puertos B y D, como explicamos anteriormente, en pines para el GLCD.
Además se muestra las instrucciones de cómo se inicializa el GLCD, cómo seleccionar el color de fondo y como escribir los datos almacenados en la RAM para visualizarse en la pantalla. En la figura 3.15 se observa las definiciones del GLCD.

PicUSB.c | GRAPHICS.C | HDM64G512.c

```
25  Function Prototypes
26  void glcd_init(int1 mode);
27  void glcd_pixel(int8 x, int8 y, int1 color);
28  void glcd_fillScreen(int1 color);
29  void glcd_writeByte(int1 side, BYTE data);
30  BYTE glcd_readByte(int1 side);
31  void glcd_update();
32  // Definición de los pines del GLCD Gráfico y sus librerias
33  #define GLCD_CS1 PIN_D4
34  #define GLCD_CS2 PIN_D2
35  #define GLCD_DI PIN_D7
36  #define GLCD_RW PIN_D6
37  #define GLCD_E PIN_D5
38  #define GLCD_RST PIN_D3
```

Fig. 3.15 Definiciones para visualizar los datos en el GLCD.

No está por demás aclarar que de los 40 pines del PIC 32 son usados: en las aplicaciones, sobrando 8 que se usan para la polarización del PIC.

3.3.4 PRESENTACIÓN DE IMÁGENES EN EL GLCD.

Para visualizar una imagen en el GLCD, es necesario primero diseñar las imágenes en blanco y negro de 128 pixeles de ancho por 64 pixeles de alto, este sería el tamaño máximo de las imágenes que se desea presentar como se observa en la figura 3.16.

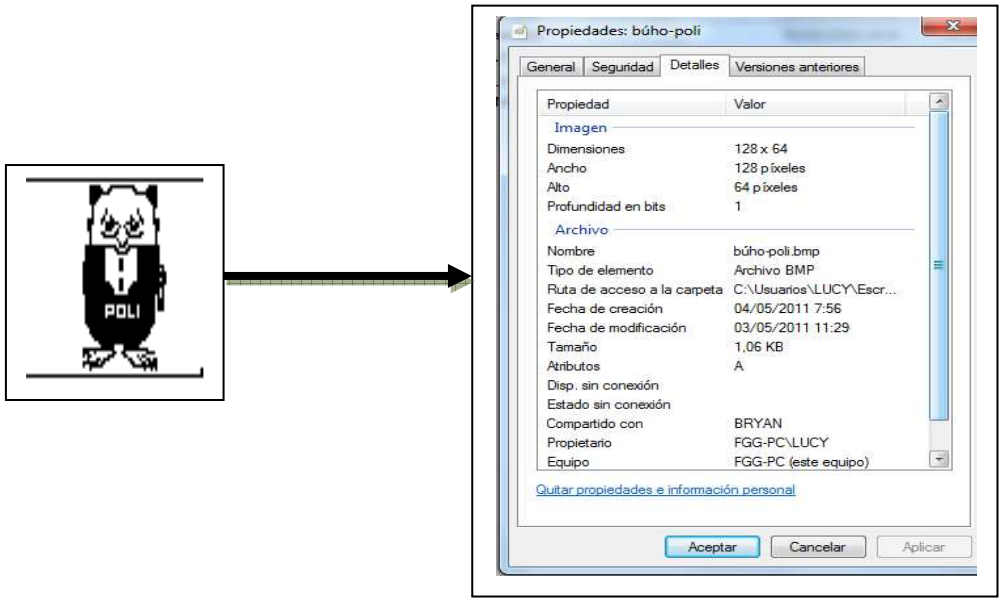


Fig. 3.16 Tamaño de imagen a grabar en el GLCD.

Usamos un software llamado GTP que es un programa para diseñar gráficos a blanco y negro con los pixeles deseados, en la figura 3.17 se observa la ventana principal del Software GTP. Este software crea un código el cual lo usamos para introducirlo en el programa CCS y obtenemos nuestra imagen y la visualizamos en el GLCD siguiendo estos pasos:

- Abrimos el software GTP
- Seleccionamos ANSI.C
- Damos clic en abrir, luego seleccionamos la imagen deseada, la que ya ha sido previamente guardada.
- Damos clic en CREAR y se genera el código.

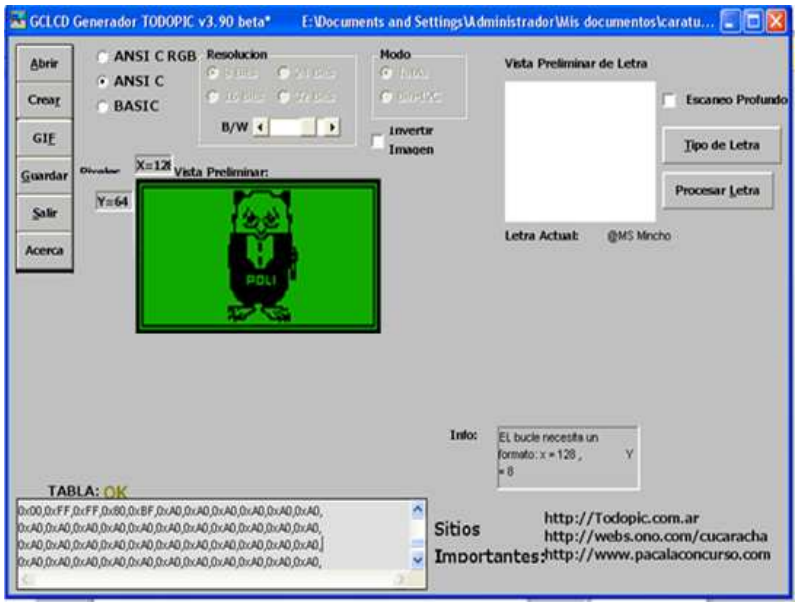


Fig. 3.17 Imagen que se visualizará en el GLCD.

Si la figura seleccionada no cumple con las características requeridas es decir la resolución de 128*64 pixeles el programa rechazará la imagen, por lo que no se podrá obtener el código necesario para nuestros fines.

Damos clic en el botón CODE para visualizar el código generado de acuerdo a la imagen del gráfico seleccionado.

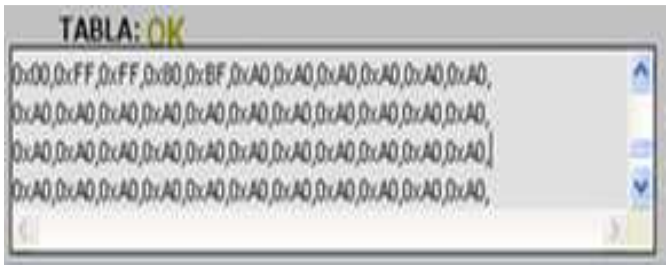


Fig. 3.18 Código generado de acuerdo a la imagen seleccionada.

En la figura 3.18 se muestra el código generado, este código se debe copiar en un archivo de texto (bloc de notas), y guardar en la carpeta PDS generada anteriormente en el subcapítulo 3.3.1 o simplemente se copia el código generado y se lo pega en el programa CCS y automáticamente se genera la imagen.

El código generado se debe incluir en el programa CCS de la siguiente manera: Unsigned char const SELLO y seguido el código.

A continuación detallamos las instrucciones para representar una imagen en el GLCD, esta será visualizada por un tiempo de 4 segundos para nuestro proyecto.

glcd_line(x1, y1, x2, y2, color)

glcd_rect(x1, y1, x2, y2, fill, color)

glcd_bar(x1, y1, x2, y2, width, color)

glcd_circle(x, y, radius, fill, color)

glcd_text57(x, y, textptr, size, color)

3.3.5 PRESENTACIÓN DE LOS DATOS EN EL GLCD.

Para visualizar datos provenientes de variables físicas como temperatura, resistencia, etc. Se debe seguir un proceso el cual se consigue con un sensor, luego el acoplamiento y finalmente este será el que envíe la información a los pórticos del PIC y aquí se realizarán las conversiones necesarias para luego distribuirse a sus respectivos periféricos. Para ver el programa completo ver en el Anexo G.

3.4 DESARROLLO DEL SOFTWARE PARA LA INTERFAZ GRÁFICA.

Otro mecanismo de salida de la Libreta Electrónica de Practicas es la interfaz gráfica, la misma que está desarrollada en el software Visual C#. En esta interfaz gráfica podemos observar los valores máximos y mínimos a los que giran los motores, además podemos observar el número de caracteres que podemos escribir, el valor de la lectura del sensor.

Cabe recalcar que el programa está diseñado para usar varias aplicaciones a la vez, según lo deseado por el interesado.

3.4.1 PROGRAMA PARA LA INTERFAZ GRÁFICA.

3.4.1.1 Código para la comunicación entre el PC y el PIC USB

En el capítulo 3.3 se generaron 2 carpetas una de nombre PDS y otra llamada Visual C.

Entonces abrimos la carpeta Visual C y encontramos un archivo llamado .dsp el cual nos genera un código para poder comunicarnos mediante el puerto USB, y este archivo es compatible con Visual C# el que nos permite realizar conversiones desde el archivo.dsp a Visual C#. De esta manera se logra la comunicación USB entre la Tarjeta Electrónica y la Interfaz Gráfica.

3.4.1.2 Código para la lectura desde el PIC.

Una vez abierto el archivo USB Project .dsp observamos 2 archivos el uno contiene el código de la interfaz gráfica y el otro contiene la presentación en la que podrán observar los usuarios

En el código están definidas las variables, los buffers de entrada y salida, además de la programación de los diferentes botones que se va a presentar en la interfaz gráfica y como se va a visualizar los datos en forma gráfica y numérica.

3.4.2 DISEÑO DE VISUALIZACIÓN DE LA INTERFAZ GRÁFICA.

En la figura 3.19 se observa la ventana donde se va a realizar la interfaz gráfica. Al dar clic en el botón archivo se abre una nueva ventana de trabajo, la cual posee un cuadro de herramientas, un explorador de soluciones, otra ventana de propiedades y en la parte inferior una pequeña ventana llamada lista de errores. En la ventana de trabajo podemos arrastrar: botones de comando, imágenes, cuadros de texto etc. desde el formulario a la ventana principal, luego damos clic en ver, clic en código y nos aparece unas líneas de la programación, a las cuales las podemos modificar de acuerdo a nuestras necesidades.

Una vez terminado el programa lo ejecutamos para ver si tenemos errores o ya está listo, por último observamos los resultados.

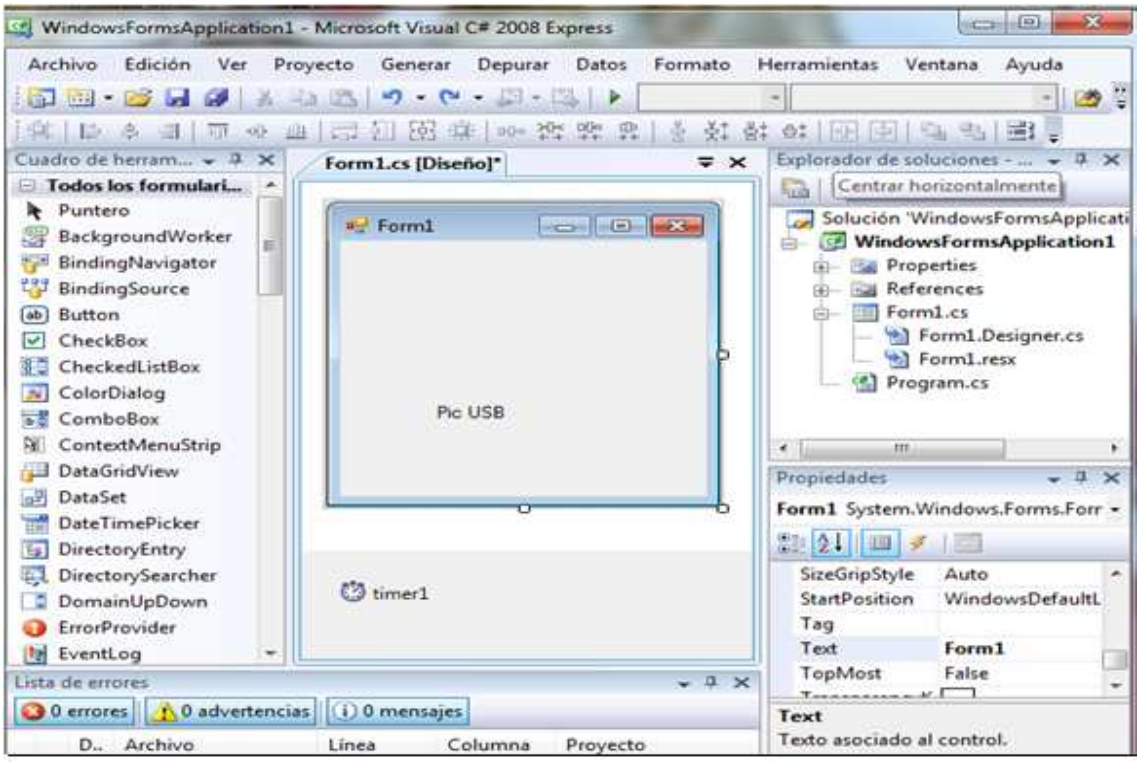


Fig. 3.19 Ventana donde se va generando la interfaz gráfica.

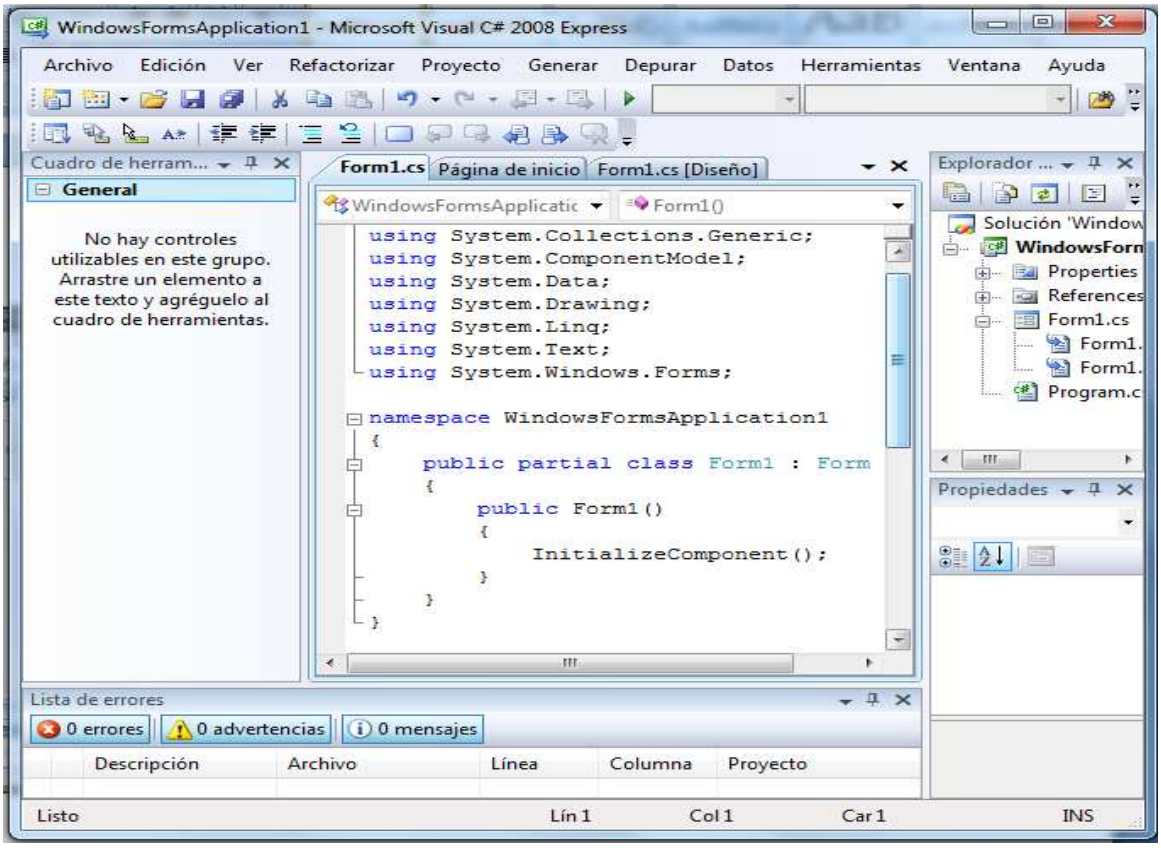


Fig. 3.20 Código generando del botón, Pic USB

Cada botón de comando como ya explicamos anteriormente puede ser modificado de manera que se desee con solo dar doble clic en el botón ver código y nos muestra el código sobre el que vamos a modificar como se observa en la figura 3.20. Para editar cada botón nos vamos a propiedades y observamos todo lo que podemos cambiar y dar forma. El programa completo se lo puede observar en el Anexo G.

3.4.3 GENERACIÓN DEL ARCHIVO EJECUTABLE.

Cuando se ha terminado de realizar el programa para la interfaz gráfica, damos clic en el botón de iniciar depuración, si todo el programa, tanto el código como el diseño están perfectamente funcionando no aparecerá ningún mensaje de error, de lo contrario se indicará con un mensaje qué errores se debe corregir. Cuando el programa esté listo sin errores procedemos a guardarlo y automáticamente se generan varios archivos y entre ellos un archivo ejecutable que lo obtuvimos de la siguiente manera:

- Buscamos la ubicación del proyecto y encontramos la carpeta PicUSB como se observa en la figura 3.21.

Fig. 3.21 Ubicación del proyecto.

- Abrimos la carpeta PicUSB y encontramos varias carpetas que se generan por default, de las que nos interesa la carpeta llamada bin que se observa en la figura 3.22.

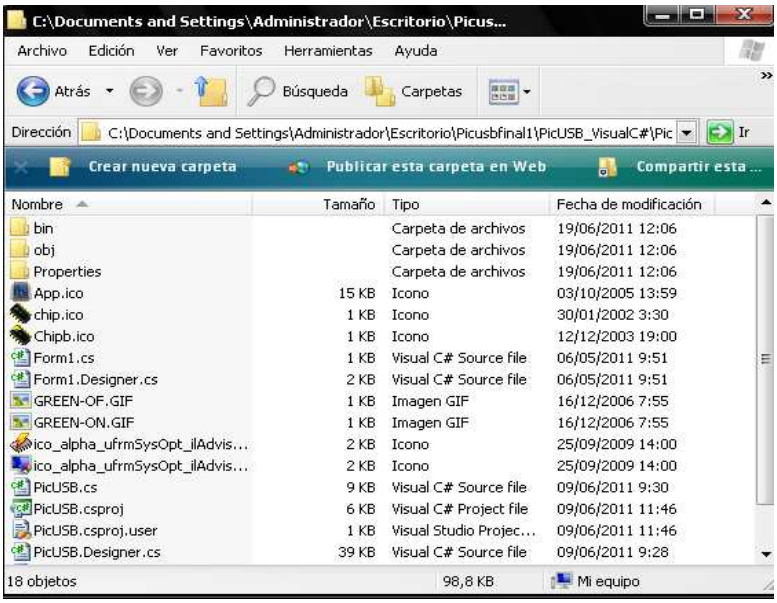


Fig. 3.22 Carpeta bin.

- Abrimos la carpeta bin y vemos que automáticamente se crearon unos archivos, el PicUSB.exe es el ejecutable, se muestra en la figura 3.23. al abrir este archivo se observa la Interfaz.

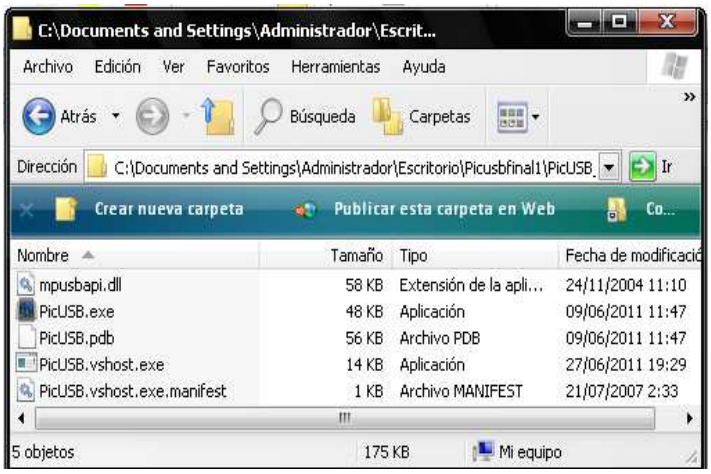


Fig. 3.23 Ventana del archivo ejecutable.

Es de suma importancia generar este archivo ejecutable, de lo contrario la interfaz gráfica solo se abrirá en PCs que tengan el programa Visual C#, la ventaja de este archivo es que se lo puede abrir en cualquier PC sin necesidad de que tenga instalado el programa Visual C#.

3.4.4 PRESENTACIÓN DE LA INTERFAZ GRÁFICA TERMINADA.

En el siguiente diagrama de la figura 3.24 podemos observar la interfaz gráfica novedosa y llamativa ya finalizada con todos los botones y lista para trabajar e interactuar con la PC.

En el diagrama podemos apreciar cómo se ha diseñado la Interfaz Gráfica y a la vez comprender su funcionamiento.

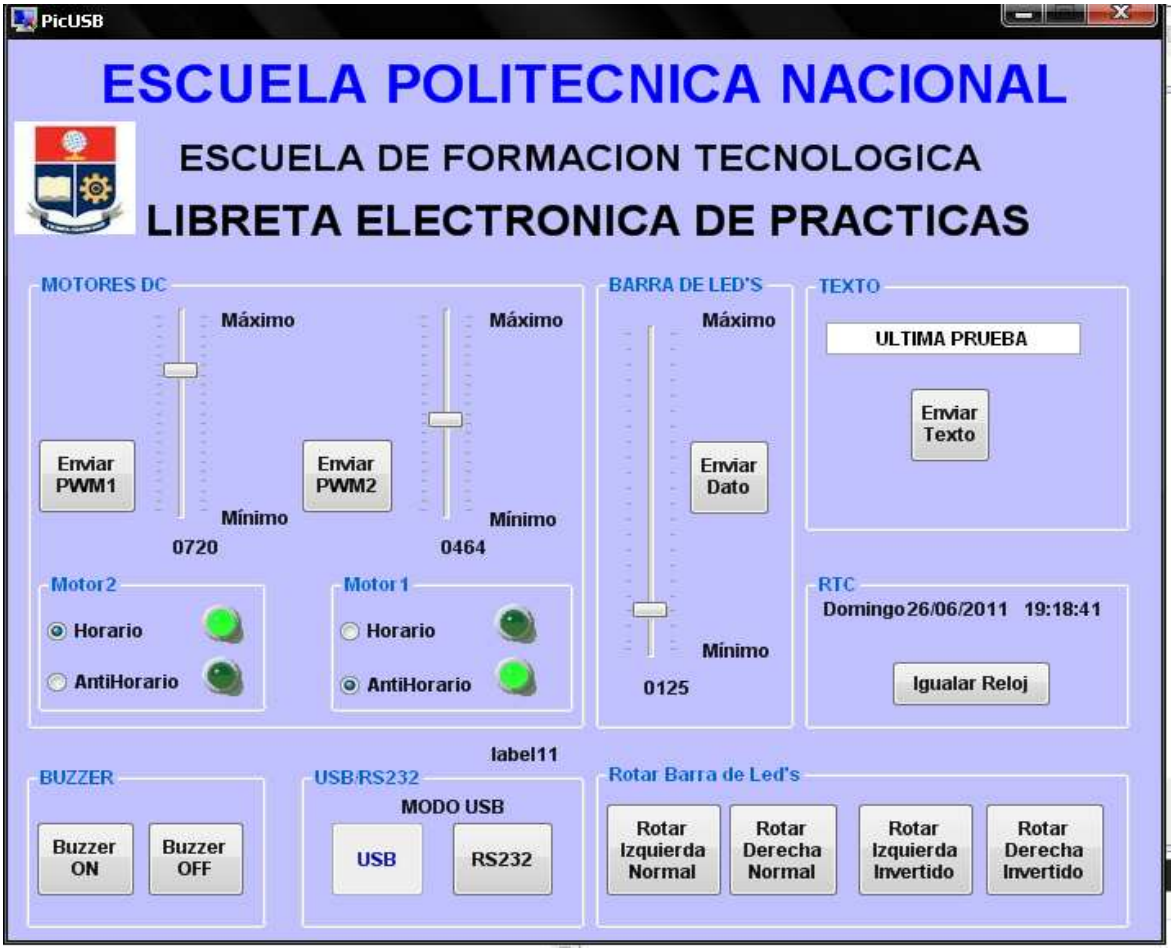


Fig. 3.24 Interfaz gráfica terminada.

3.4.5 DESCRIPCIÓN DE LOS BOTONES DE LA INTERFAZ GRÁFICA.

La Interfaz gráfica consta de cinco aplicaciones: para los motores DC se usan varios botones como se puede apreciar permitiéndonos escoger el sentido de giro y a la vez aumentar o disminuir la velocidad, para el buzzer 2 botones de encendido y apagado, para la comunicación USB y RS232 igual 2 botones, para la barra de LEDs se usan 5 botones, de los cuales 4 son para seleccionar la forma en que rota la luz en la barra de LEDs y un botón que envía dato mediante el cual seleccionamos un número y lo observamos en hexadecimal,

además posee un botón para igualar la fecha y hora; por último el botón que nos permite escribir y enviar texto de 20 caracteres.

3.4.6 LA LIBRETA DE PRÁCTICAS INTERACTUANDO CON LA PC.

Realizadas las conexiones del circuito comprobamos el funcionamiento de la Tarjeta Electrónica, observamos cómo interactúa la Interfaz Gráfica con la PC y los resultados de las diferentes aplicaciones. En la figura 3.25 podemos observar la Libreta y la Interfaz Gráfica funcionando.

Fig. 3.25 Conexión y funcionamiento completo de toda la Libreta Electrónica.

4. CAPÍTULO IV: PRUEBAS Y RESULTADOS.

El objetivo principal de este capítulo es comprobar el funcionamiento de cada una de las aplicaciones mencionadas anteriormente, visualizar cómo interactúa la interfaz gráfica con la PC y observar los resultados.

4.1 PRUEBAS CON LOS MOTORES DC.

En la figura 4.1 se muestra las pruebas realizadas en los motores, en esta prueba se puede observar la variación del ciclo de trabajo de cada motor en el GLCD ya que no se puede observar como giran los motores de acuerdo al sentido elegido.

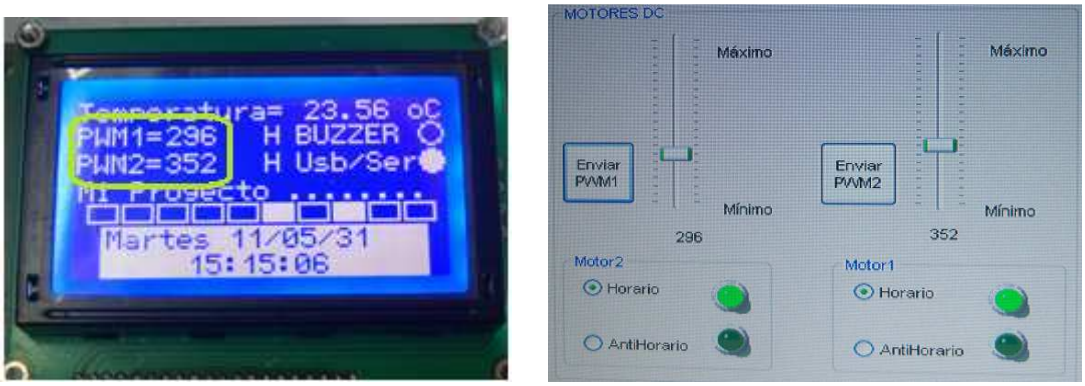


Fig. 4.1 Presentación de las lecturas a la giran los motores.

4.2 PRUEBAS CON LA BARRA DE LEDS.

En la figura 4.2 podemos observar que los LEDs forman el número 20 en binario entendiéndose que un led encendido es 1L y un led apagado en 0L, se lee de derecha a izquierda por lo tanto se lee 0000010100 como podemos apreciar en las imágenes.

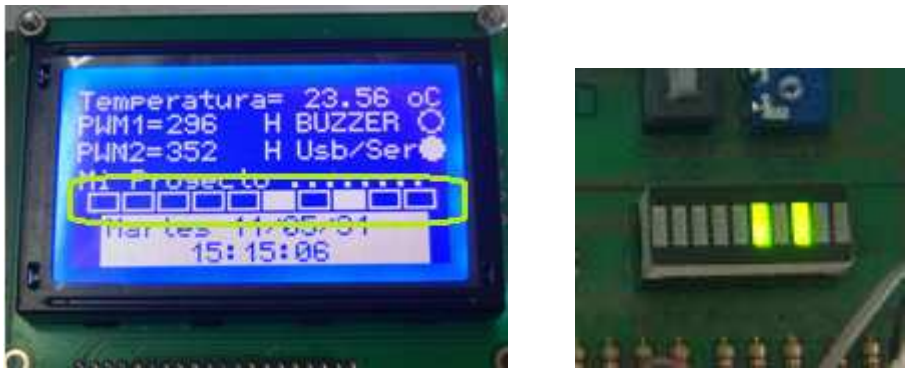


Fig. 4.2 Podemos observar el número 20 en binario en el GLCD y en la barra de LEDs.

4.3 PRUEBAS CON EL BUZZER.

El buzzer emite un sonido el cual no podemos indicar en esta prueba, pero en la figura 4.3 podemos observar que el círculo, que es símbolo de buzzer está activado tanto en la interfaz gráfica como en el GLCD.

Fig. 4.3 Círculo encendido en el GLCD y la interfaz gráfica.

4.4 PRUEBAS CON EL SENSOR.

En la figura 4.4 podemos observar al sensor midiendo la temperatura de una vela encendida y el valor de la temperatura está en 42℃ en el GLCD.

Fig. 4.4 Lectura de la Temperatura.

4.5 PRUEBAS CON EL GLCD.

En la figura 4.5 podemos visualizar los resultados que se realizan en cada aplicación, podemos observar el texto que se ingresa en la Interfaz Gráfica el cual debe ser de 20 caracteres o menos pero no más de eso, debido a que esa es cantidad de caracteres que se puede escribir en una línea del GLCD.

Fig. 4.5 Texto de 20 caracteres.

4.6 COSTOS DEL PROYECTO.

En la siguiente tabla de la figura 4.6 podemos observar los costos de cada elemento usado en nuestro proyecto.

El costo de cada elemento es en base a precios reales a los cuales los hemos adquirido.

El costo de cada elemento también puede variar dependiendo del lugar donde se adquiera los elementos.

Ítem	Elemento	Cantidad	P. Unitario	P. Total
1.	Buzzer	1	1.00	1
2.	Capacitor 1000µF	1	1.00	1
3.	Capacitor 10µF	6	0.10	0,60
4.	Capacitor 0.1µF	3	0.10	0,30
5.	Capacitor 470µF/100v	1	0.40	0,40
6.	Capacitor 22pF	2	0.10	0,20
7.	Capacitor 47µF/25v	1	0.10	0,10
8.	Conector DB9	1	1.00	1
9.	LED	2	0.10	0,20
10.	Diodo 1N4007	10	0.10	1
11.	Un diodo 1N5406	1	0.30	0,30
12.	Regulador 7805	1	2.00	2
13.	RCA JACK	2	0.30	0,60
14.	HEADER 20	1	1.00	1
15.	HEADER 2	7	0.30	2,10
16.	SENSOR DS18B20	1	1.00	1
17.	LEDs	1	1.50	1,50
18.	Batería	1	1.00	1
19.	HEADER 3	1	0.40	0,40

20.	Resistencia 50kΩ	1	1.00	1
21.	Resistencia 100kΩ	1	0.05	0,05
22.	Resistencia 2.7kΩ	1	0.05	0,05
23.	Resistencia 4.7kΩ	2	0.05	0,10
24.	Resistencia 10kΩ	3	0.05	0,15
25.	Resistencia 820Ω	1	0.05	0,05
26.	Resistencia 470Ω	10	0.05	0,50
27.	Resistencia 10Ω	1	0.05	0,05
28.	Transistor 2N3904	1	0.10	0,10
29.	TIP 127	1	0.50	0,50
30.	SW DPDT	1	0.40	0,40
31.	SW SDPDT	2	0.40	0,80
32.	PUSH BUTTON	1	0.50	0,50
33.	HEADER 4	1	1	1
34.	PIC 18F4550	1	15	15
35.	MAX 232	1	3	3
36.	DRIVER L293B	1	3	3
37.	RTC DS 1307	1	4	4
38.	TPIC 6B595N	2	5	10
39.	CRISTAL 20MHz	1	1	1
40.	CRISTAL 12MHz	1	0.60	0,60
41.	Zócalo 16 pines	2	0,3	0,60
42.	Zócalo 8 pines	1	0,2	0,20
43.	Zócalo 20 pines	2	0,4	0,80
44.	Zócalo Zip	1	10	10
45.	GLCD gráfico 128*64	1	40	40
46.	Cable USB	1	3	3
47.	Cable Serial 1.5 mts	1	3	3
48.	Construcción de la Placa	1	70	70
49.	Total			185,15

Fig. 4.6 Tabla de costos de elementos del proyecto.

5. CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES.

5.1 CONCLUSIONES.

- Es importante aclarar que los diferentes software como Visual C# y CCS poseen comunicación RS-232 y comunicación USB.
- Usamos el PIC 18F4550 ya que es un micro ampliamente utilizado por sus características y potencia, incluye una memoria Flash USB y control de flujo de datos complementándose con la comunicación I2C y SPI.
- Es substancial conocer que mediante la comunicación RS232 tenemos un alcance de 30 metros y con la comunicación USB el alcance es menor, de 5 metros.
- Esta Tarjeta Electrónica de Prácticas tiene la finalidad de ser usada por los estudiantes en el Laboratorio de Microprocesadores para la enseñanza práctica de la programación de los PICs de la familia 18FXX, usando Visual C# y los puertos USB y RS232.
- La fuente debe poseer una corriente de alimentación suficiente para los elementos que se van a usar, para nuestro caso como usamos motores la fuente debe ser mínimo de 1.2 Amperios.
- El pulsador de reset se lo debe usar para reiniciar la Tarjeta Electrónica cuando no funcione correctamente.
- Usamos el Software de Programación Visual C# que se deriva del Lenguaje C por poseer la sintaxis y estructura muy parecida a la de C++; además de ser un Software novedoso, moderno, orientado a objetos y en gran parte sencillo que facilita el aprendizaje a los usuarios.
- Generar un archivo ejecutable de la Interfaz Gráfica finalizada beneficia al usuario.

5.2 RECOMENDACIONES.

- Es de gran utilidad poner zócalos en los circuitos integrados más importantes para en un futuro poder reemplazarlos fácilmente sin que se produzca algún daño.
- Debemos tomar en cuenta que el Sistema Operativo Windows 7 x64 bits no es compatible con la mayoría de los Software de programación como ocurrió con nuestros Software Visual C# y Protón, pero sí es compatible el Sistema Operativo Windows 7 x32 bits con la mayoría de Software.
- Para facilitar el manejo de la Libreta Electrónica de Prácticas se aconseja leer el Manual del usuario, el cual está en el Anexo F.
- Comprobar que todos los elementos estén bien conectados antes de su uso.
- Queda como recomendación para otros usuarios probar la Tarjeta Electrónica con otros lenguajes de programación.
- Se recomienda tener precaución con el GLCD ya que este dispositivo es costoso.
- Instalar el driver de la Tarjeta Electrónica cada vez que se use en una computadora diferente.

REFERENCIAS BIBLIOGRÁFICAS.

No	Título Referencia tomada
[1]	Microcontroladores. http://es.scribd.com/doc/44197510/PIC-18F4550
[2]	Microcontroladores. http://perso.wanadoo.es/pictob/microprg.htm#microprocesadores_y_microcontroladores
[3]	Microcontroladores. http://perso.wanadoo.es/pictob/microcr.htm
[4]	Microcontroladores. http://www.bairesrobotics.com.ar/data/Manual_Compilador_CCS_PICC.pdf
[5]	Microcontroladores. http://www.electromicrodigital.com/site2/index.php?option=com_content&view=article&id=90&Itemid=29
[6]	RS232 www.codeproject.com/kb/system/PicRS232.aspx
[7]	RS232 http://focus.ti.com/lit/ds/symlink/max232.pdf
[8]	GLCD http://es.wikipedia.org/wiki/GLCD
[9]	GLCD http://www.micros-designs.com.ar/libreria-glcd-128x64-c/
[10]	Comunicación USB http://www.monografias.com/trabajos11/usbmem/usbmem.shtml
[11]	Comunicación USB http://www.i-micro.com/pdf/articulos/usb_imicro.pdf
[12]	Comunicación RS232 http://www.alegsa.com.ar/Dic/COM.php
[13]	Comunicación RS232 http://www.misrespuestas.com/que-es-un-puerto-serial.html
[14]	Comunicación RS232 http://www.lvr.com/serport.htm
[15]	Electrónica http://es.wikipedia.org/wiki/Zumbador
[16]	Electrónica (sensor) http://www.sparkfun.com/products/245
[17]	Electrónica(TPIC) http://focus.tij.co.jp/jp/lit/ds/slis032a/slis032a.pdf
[18]	Electrónica (RTC) http://www.hispavila.com/3ds/atmega/descargas/ds1307_rtc.pdf
[19]	Visual C# http://www.lvr.com/
[20]	Visual C# http://www.solocodigo.com/foros/viewtopic.php?f=169&t=37314
[21]	Visual C# Libro de Visual C # 2008 (Step by Step). Autor: John Sharp
[22]	Visual C# http://www.icomputo.com/content/creando-aplicaciones-graficas-usando-windowsforms-c-sharp

ANEXOS

ANEXO A:
MICROCONTROLADOR PIC18F4550.

MICROCHIP PIC18F2455/2550/4455/4550

28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology

Universal Serial Bus Features:

- USB V2.0 Compliant
- Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s)
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- Supports up to 32 Endpoints (16 bidirectional)
- 1-Kbyte Dual Access RAM for USB
- On-Chip USB Transceiver with On-Chip Voltage Regulator
- Interface for Off-Chip USB Transceiver
- Streaming Parallel Port (SPP) for USB streaming transfers (40/44-pin devices only)

Power-Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 5.8 µA typical
- Sleep mode currents down to 0.1 µA typical
- Timer1 Oscillator: 1.1 µA typical, 32 kHz, 2V
- Watchdog Timer: 2.1 µA typical
- Two-Speed Oscillator Start-up

Flexible Oscillator Structure:

- Four Crystal modes, including High Precision PLL for USB
- Two External Clock modes, up to 48 MHz
- Internal Oscillator Block:
  - 8 user-selectable frequencies, from 31 kHz to 8 MHz
  - User-tunable to compensate for frequency drift
- Secondary Oscillator using Timer1 @ 32 kHz
- Dual Oscillator options allow microcontroller and USB module to run at different clock speeds
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if any clock stops

Peripheral Highlights:

- High-Current Sink/Source: 25 mA/25 mA
- Three External Interrupts
- Four Timer modules (Timer0 to Timer3)
- Up to 2 Capture/Compare/PWM (CCP) modules:
  - Capture is 16-bit, max. resolution 5.2 ns (TCY/16)
  - Compare is 16-bit, max. resolution 83.3 ns (TCY)
  - PWM output: PWM resolution is 1 to 10-bit
- Enhanced Capture/Compare/PWM (ECCP) module:
  - Multiple output modes
  - Selectable polarity
  - Programmable dead time
  - Auto-shutdown and auto-restart
- Enhanced USART module:
  - LIN bus support
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I²C™ Master and Slave modes
- 10-bit, up to 13-channel Analog-to-Digital Converter module (A/D) with Programmable Acquisition Time
- Dual Analog Comparators with Input Multiplexing

Special Microcontroller Features:

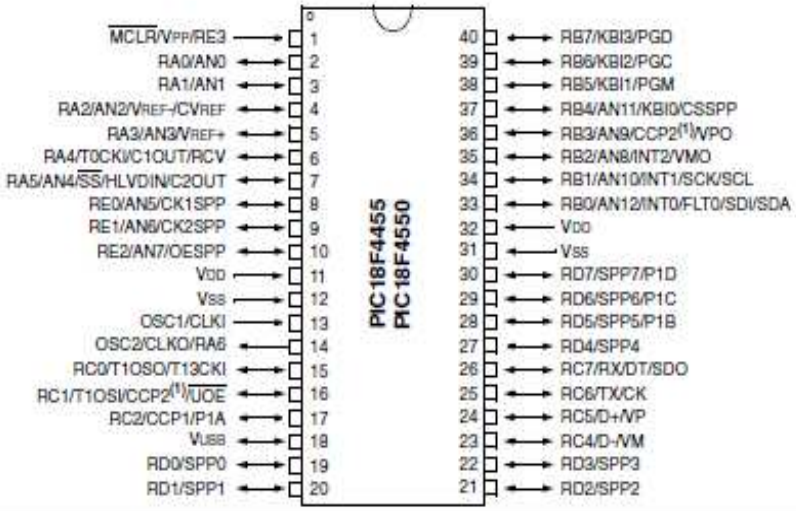
- C Compiler Optimized Architecture with optional Extended Instruction Set
- 100,000 Erase/Write Cycle Enhanced Flash Program Memory typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory typical
- Flash/Data EEPROM Retention: > 40 years
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 41 ms to 131s
- Programmable Code Protection
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Optional dedicated ICD/ICSP port (44-pin devices only)
- Wide Operating Voltage Range (2.0V to 5.5V)

Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EUSART	Comparators	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I²C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

PIC18F2455/2550/4455/4550

Pin Diagrams

40-Pin PDIP



PIC18F2455/2550/4455/4550

TABLE 1-2: PIC18F2455/2550 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number: PDIP, SOIC	Pin Type	Buffer Type	Description
MCLR/VPP/RE3	1			Master Clear (input) or programming voltage (input).
MCLR		I	ST	Master Clear (Reset) input. This pin is an active-low Reset to the device.
VPP		P		Programming voltage input.
RE3		I	ST	Digital input.
OSC1/CLKI	9			Oscillator crystal or external clock input.
OSC1		I	Analog	Oscillator crystal input or external clock source input.
CLKI		I	Analog	External clock source input. Always associated with pin function OSC1. (See OSC2/CLKO pin.)
OSC2/CLKO/RA6	10			Oscillator crystal or clock output.
OSC2		O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO		O	—	In select modes, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
RA6		I/O	TTL	General purpose I/O pin.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX Configuration bit is cleared.
2: Default assignment for CCP2 when CCP2MX Configuration bit is set.

ANEXO B

SENSOR DE TEMPERATURA DS18B20.

DS18B20
Programmable Resolution 1-Wire Digital Thermometer

DESCRIPTION

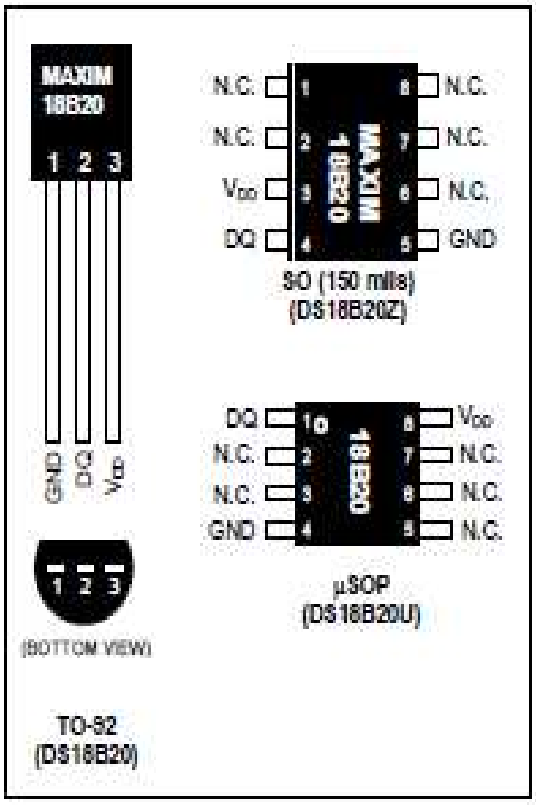
The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to +125°C and is accurate to ±0.5°C over the range of -10°C to +85°C. In addition, the DS18B20 can derive power directly from the data line ("parasite power"), eliminating the need for an external power supply.

Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment, or machinery, and process monitoring and control systems.

FEATURES

- Unique 1-Wire® Interface Requires Only One Port Pin for Communication
- Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- Requires No External Components
- Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V
- Measures Temperatures from -55°C to +125°C (-67°F to +257°F)
- ±0.5°C Accuracy from -10°C to +85°C
- Thermometer Resolution is User Selectable from 9 to 12 Bits
- User-Definable Nonvolatile (NV) Alarm Settings
- Alarm Search Command Identifies and Addresses Devices Whose Temperature is Outside Programmed Limits (Temperature Alarm Condition)
- Available in 8-Pin SO (150 mils), 8-Pin µSOP, and 3-Pin TO-92 Packages
- Software Compatible with the DS1822
- Applications Include Thermostatic Controls, Industrial Systems, Consumer Products, Thermometers, or Any Thermally Sensitive System

PIN CONFIGURATIONS



PIN DESCRIPTION

PIN			NAME	FUNCTION
SO	µSOP	TO-92		
1, 2, 6, 7, 8	2, 3, 5, 6, 7	—	N.C.	No Connection
3	8	3	V_{DD}	Optional V_{DD}. V_{DD} must be grounded for operation in parasite power mode.
4	1	2	DQ	Data Input/Output. Open-drain 1-Wire interface pin. Also provides power to the device when used in parasite power mode (see the *Powering the DS18B20* section.)
5	4	1	GND	Ground

OVERVIEW

Figure 1 shows a block diagram of the DS18B20, and pin descriptions are given in the *Pin Description* table. The 64-bit ROM stores the device's unique serial code. The scratchpad memory contains the 2-byte temperature register that stores the digital output from the temperature sensor. In addition, the scratchpad provides access to the 1-byte upper and lower alarm trigger registers (T_H and T_L) and the 1-byte configuration register. The configuration register allows the user to set the resolution of the temperature-to-digital conversion to 9, 10, 11, or 12 bits. The T_H, T_L, and configuration registers are nonvolatile (EEPROM), so they will retain data when the device is powered down.

The DS18B20 uses Maxim's exclusive 1-Wire bus protocol that implements bus communication using one control signal. The control line requires a weak pullup resistor since all devices are linked to the bus via a 3-state or open-drain port (the DQ pin in the case of the DS18B20). In this bus system, the microprocessor (the master device) identifies and addresses devices on the bus using each device's unique 64-bit code. Because each device has a unique code, the number of devices that can be addressed on one

1-WIRE BUS SYSTEM

The 1-Wire bus system uses a single bus master to control one or more slave devices. The DS18B20 is always a slave. When there is only one slave on the bus, the system is referred to as a "single-drop" system; the system is "multidrop" if there are multiple slaves on the bus.

All data and commands are transmitted least significant bit first over the 1-Wire bus.

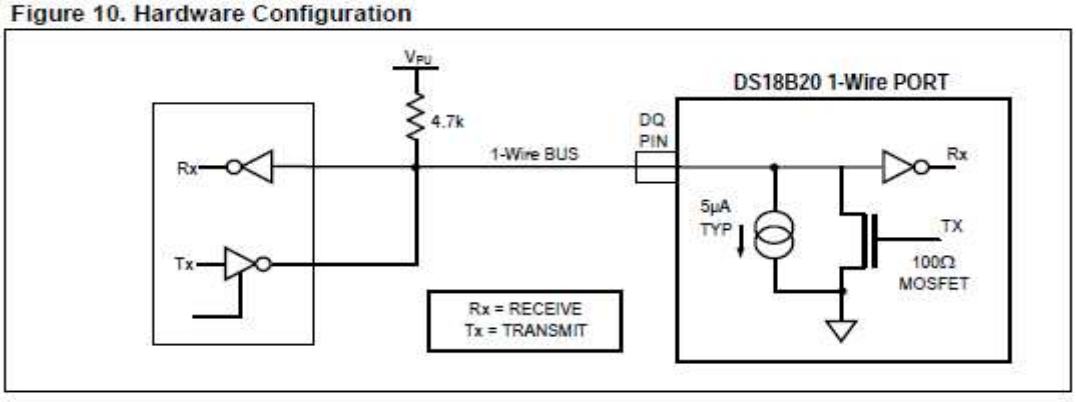
The following discussion of the 1-Wire bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-Wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-Wire bus has by definition only a single data line. Each device (master or slave) interfaces to the data line via an open-drain or 3-state port. This allows each device to "release" the data line when the device is not transmitting data so the bus is available for use by another device. The 1-Wire port of the DS18B20 (the DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 10.

The 1-Wire bus requires an external pullup resistor of approximately 5kΩ; thus, the idle state for the 1-Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-Wire bus is in the inactive (high) state during the recovery period. If the bus is held low for more than 480µs, all components on the bus will be reset.

Figure 10. Hardware Configuration



ANEXO C

RTC DS1307.

DALLAS SEMICONDUCTOR

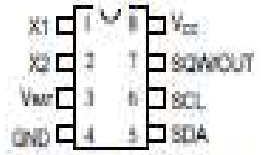
DS1307
64 x 8 Serial Real-Time Clock

www.maxim-ic.com

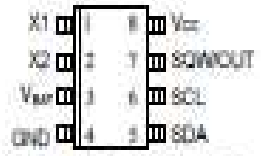
FEATURES

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap-year compensation valid up to 2100
- 56-byte, battery-backed, nonvolatile (NV) RAM for data storage
- Two-wire serial interface
- Programmable squarewave output signal
- Automatic power-fail detect and switch circuitry
- Consumes less than 500nA in battery backup mode with oscillator running
- Optional industrial temperature range: -40°C to +85°C
- Available in 8-pin DIP or SOIC
- Underwriters Laboratory (UL) recognized

PIN ASSIGNMENT



DS1307 8-Pin DIP (300-mil)



DS1307 8-Pin SOIC (150-mil)

PIN DESCRIPTION

V_{CC}	- Primary Power Supply
X1, X2	- 32.768kHz Crystal Connection
V_{BAT}	- +3V Battery Input
GND	- Ground
SDA	- Serial Data
SCL	- Serial Clock
SQW/OUT	- Square Wave/Output Driver

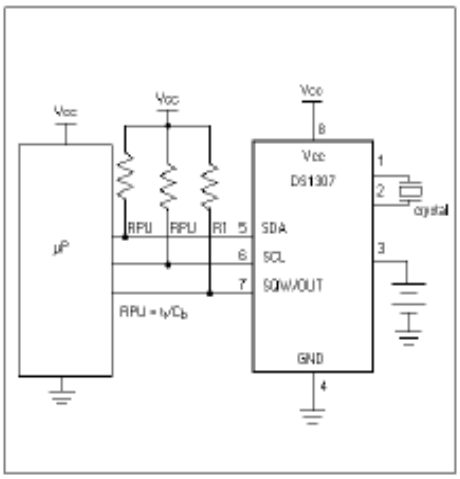
ORDERING INFORMATION

DS1307	8-Pin DIP (300-mil)
DS1307Z	8-Pin SOIC (150-mil)
DS1307N	8-Pin DIP (Industrial)
DS1307ZN	8-Pin SOIC (Industrial)

DESCRIPTION

The DS1307 Serial Real-Time Clock is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit that detects power failures and automatically switches to the battery supply.

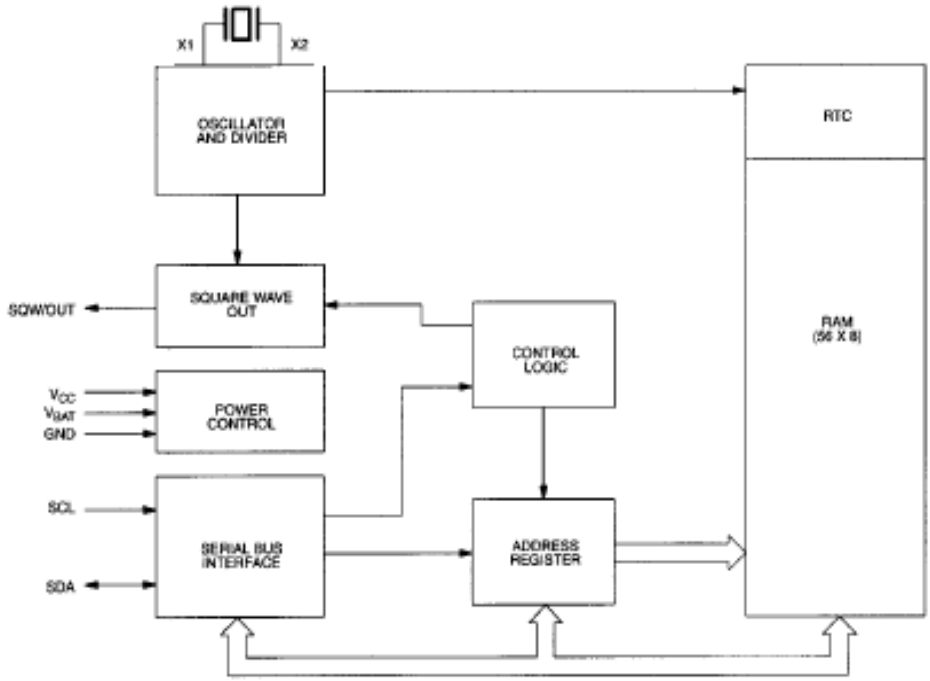
TYPICAL OPERATING CIRCUIT



OPERATION

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT} the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When V_{CC} falls below V_{BAT} the device switches into a low-current battery backup mode. Upon power-up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT} + 0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the serial RTC.

DS1307 BLOCK DIAGRAM Figure 1



ANEXO D

DRIVER L293B.

L293B

PUSH-PULL FOUR CHANNEL DRIVERS

- OUTPUT CURRENT 1A PER CHANNEL
- PEAK OUTPUT CURRENT 2A PER CHANNEL (non repetitive)
- INHIBIT FACILITY
- HIGH NOISE IMMUNITY
- SEPARATE LOGIC SUPPLY
- OVERTEMPERATURE PROTECTION

DESCRIPTION

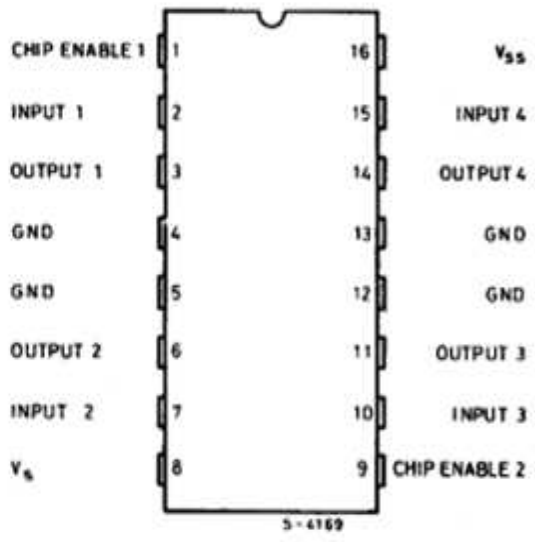
The L293B and L293E are quad push-pull drivers capable of delivering output currents to 1A per channel. Each channel is controlled by a TTL-compatible logic input and each pair of drivers (a full bridge) is equipped with an inhibit input which turns off all four transistors. A separate supply input is provided for the logic so that it may be run off a lower voltage to reduce dissipation.

Additionally, the L293E has external connection of sensing resistors, for switchmode control.

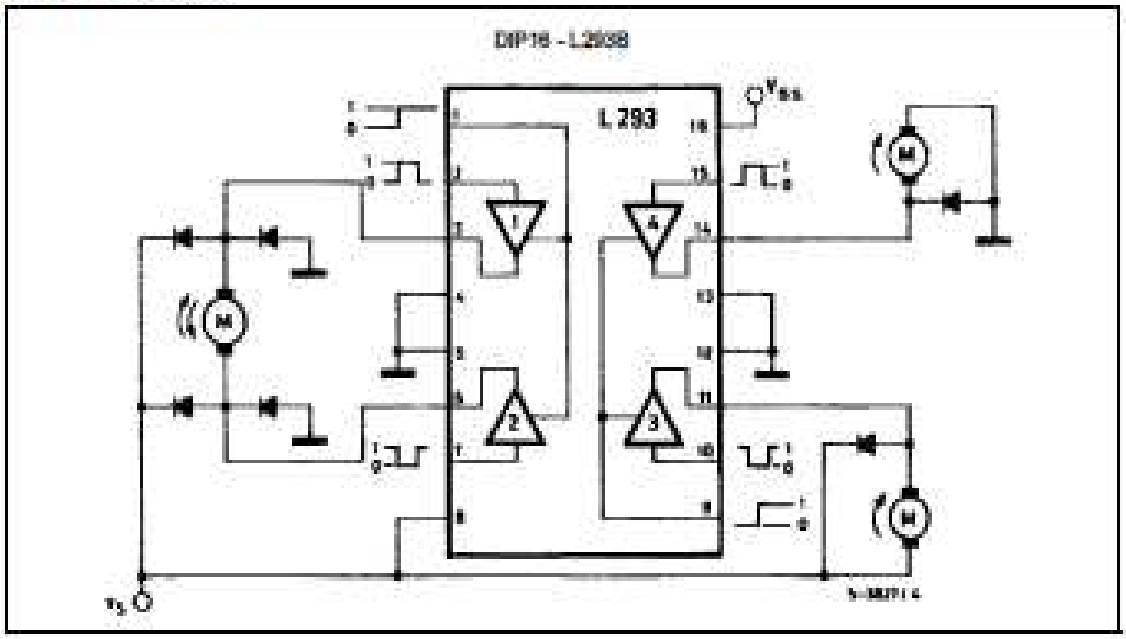
The L293B and L293E are package in 16 and 20-pin plastic DIPs respectively ; both use the four center pins to conduct heat to the printed circuit board.

PIN CONNECTIONS

DIP16 - L293B



BLOCK DIAGRAMS



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_s	Supply Voltage	36	V
V_{ss}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{inh}	Inhibit Voltage	7	V
I_{out}	Peak Output Current (non repetitive t = 5ms)	2	A
P_{tot}	Total Power Dissipation at $T_{ground-pins}$ = 80°C	5	W
T_{stg}, T_j	Storage and Junction Temperature	–40 to +150	°C

THERMAL DATA

Symbol	Parameter		Value	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	80	°C/W

ELECTRICAL CHARACTERISTICS

For each channel, V_S = 24V, V_{SS} = 5V, T_{amb} = 25°C, unless otherwise specified

Symbol	Parameter	Test Conditions	Min.	TYp.	Max.	Unit
V_s	Supply Voltage		V_{ss}		36	V
V_{ss}	Logic Supply Voltage		4.5		36	V
I_s	Total Quiescent Supply Current	V_i = L I_o = 0 V_{inh} = H V_i = H I_o = 0 V_{inh} = H V_{inh} = L		2 16	6 24 4	mA
I_{ss}	Total Quiescent Logic Supply Current	V_i = L I_o = 0 V_{inh} = H V_i = H I_o = 0 V_{inh} = H V_{inh} = L		44 16 16	60 22 24	mA
V_{IL}	Input Low Voltage		-03.		1.5	V
V_{IH}	Input High Voltage	$V_{SS} \leq 7V$ $V_{ss} > 7V$	2.3 2.3		V_{ss} 7	V
I_{IL}	Low Voltage Input Current	V_{il} = 1.5V			-10	µA
I_{IH}	High Voltage Input Current	$2.3V \leq V_{IH} \leq V_{ss} - 0.6V$		30	100	µA
V_{InhL}	Inhibit Low Voltage		-0.3		1.5	V
V_{InhH}	Inhibit High Voltage	$V_{SS} \leq 7V$ $V_{ss} > 7V$	2.3 2.3		V_{ss} 7	V
I_{InhL}	Low Voltage Inhibit Current	V_{InhL} = 1.5V		-30	-100	µA
I_{InhH}	High Voltage Inhibit Current	$2.3V \leq V_{InhH} \leq V_{ss} - 0.6V$			±10	µA
V_{CEsatH}	Source Output Saturation Voltage	I_o = -1A		1.4	1.8	V
V_{CEsatL}	Sink Output Saturation Voltage	I_o = 1A		1.2	1.8	V
V_{SENS}	Sensing Voltage (pins 4, 7, 14, 17) (**)				2	V
t_r	Rise Time	0.1 to 0.9 V_o (*)		250		ns
t_f	Fall Time	0.9 to 0.1 V_o (*)		250		ns
t_{on}	Turn-on Delay	0.5 V_i to 0.5 V_o (*)		750		ns
t_{off}	Turn-off Delay	0.5 V_i to 0.5 V_o (*)		200		ns

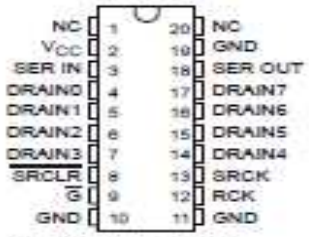
ANEXO E

TPIC 6B595N.

TPIC6B595
POWER LOGIC 8-BIT SHIFT REGISTER

- Low $r_{DS(on)}$ . . . 5 Ω Typical
- Avalanche Energy . . . 30 mJ
- Eight Power DMOS-Transistor Outputs of 150-mA Continuous Current
- 500-mA Typical Current-Limiting Capability
- Output Clamp Voltage . . . 50 V
- Devices Are Cascadable
- Low Power Consumption

DW OR N PACKAGE
(TOP VIEW)



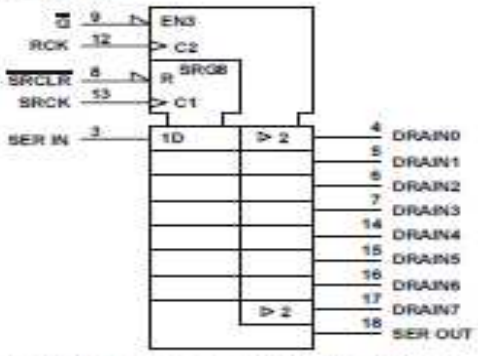
NC – No internal connection

description

The TPIC6B595 is a monolithic, high-voltage, medium-current power 8-bit shift register designed for use in systems that require relatively high load power. The device contains a built-in voltage clamp on the outputs for inductive transient protection. Power driver applications include relays, solenoids, and other medium-current or high-voltage loads.

This device contains an 8-bit serial-in, parallel-out shift register that feeds an 8-bit D-type storage register. Data transfers through both the shift and storage registers on the rising edge of the shift-register clock (SRCK) and the register clock (RCK), respectively. The storage register transfers data to the output buffer when shift-register clear ($\overline{SRCLR}$) is high. When $\overline{SRCLR}$ is low, the input shift register is cleared. When output enable ($\overline{G}$) is held high, all data in the output buffers is held low and all drain outputs are off. When $\overline{G}$ is held low, data from the storage register is transparent to the output buffers. When data in the output buffers is low, the DMOS-transistor outputs are off. When data is high, the DMOS-transistor outputs have sink-current capability. The serial output (SER OUT) allows for cascading of the data from the shift register to additional devices.

logic symbol†



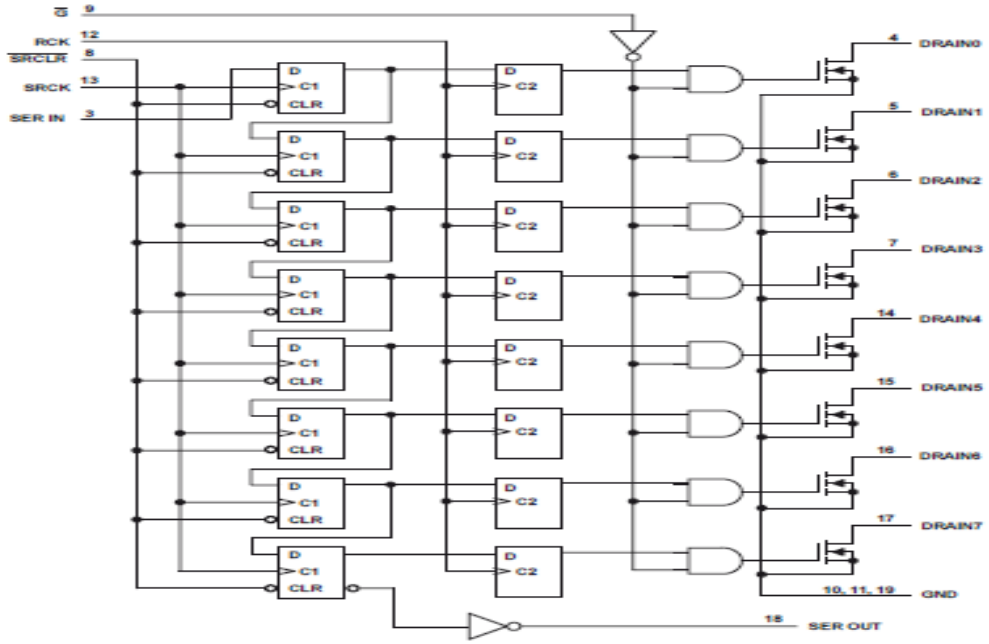
† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

Outputs are low-side, open-drain DMOS transistors with output ratings of 50 V and 150-mA continuous sink-current capability. Each output provides a 500-mA typical current limit at T_C = 25°C. The current limit decreases as the junction temperature increases for additional device protection.

The TPIC6B595 is characterized for operation over the operating case temperature range of −40°C to 125°C.

TPIC6B595
POWER LOGIC 8-BIT SHIFT REGISTER

logic diagram (positive logic)



ANEXO F

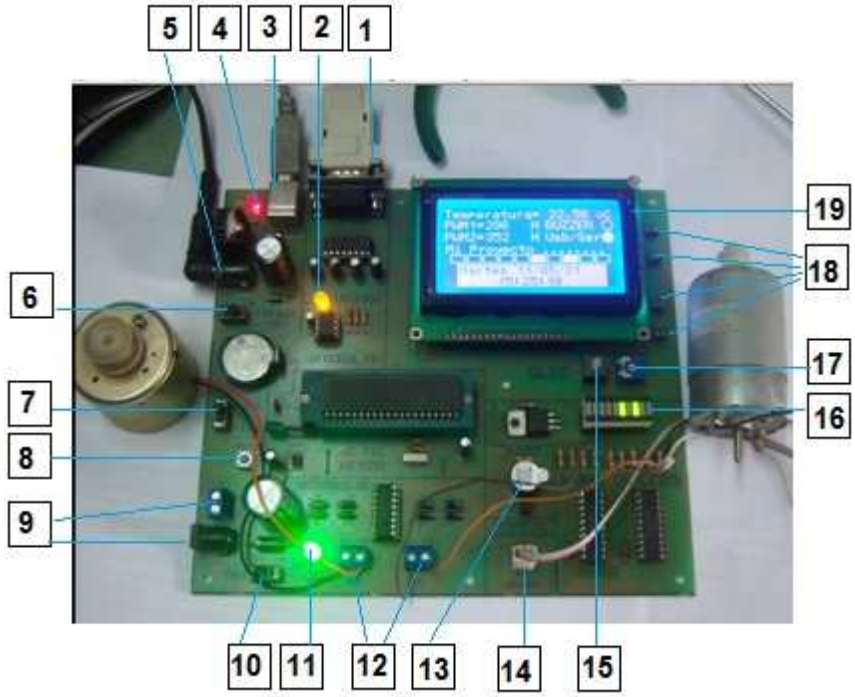
MANUAL DE USUARIO DE LA TARJETA ELECTRÓNICA DE PRÁCTICAS.

Esta Tarjeta Electrónica de Prácticas fue construida para probar la programación de los PICs de la Familia 18FXXXX con Visual C#.

CARACTERÍSTICAS DE LA TARJETA ELECTRÓNICA.

- Verificación de los resultados en la tarjeta de cada aplicación y en la interfaz gráfica.
- La Tarjeta electrónica se alimenta con 5V de fuente interna y posee otro terminal de fuente externa para una fuente de 12V para los motores.
- Posee un puerto de comunicación USB y RS232 para enviar datos.
- Posee un switch para la fuente de alimentación, que se puede escoger la fuente interna o externa.
- Posee un switch para controlar la luz de fondo.
- Posee un potenciómetro para regular el contraste del GLCD.
- Posee un botón de reset de toda la tarjeta.
- Posee 4 terminales negros para medir los voltajes de fuente.
- Posee 2 terminales azules para conectar los motores.
- Posee 1 terminal azul para conectar la fuente externa para usar motores de hasta 24 voltios.
- Led verde me indica que existe voltaje en la fuente interna o externa, es decir que si hay suficiente potencia para los motores.
- Led rojo me indica que hay 5V de la fuente interna.
- Led naranja me indica que está funcionando el reloj en tiempo real a 1Hz/seg.
- Cada circuito integrado posee un zócalo para reemplazar fácilmente al elemento dañado.

NUMERACIÓN DE LOS ELEMENTOS DE LA TARJETA.



1. Puerto de comunicación RS232.
2. Led naranja indica que trabaja el reloj.
3. Puerto de comunicación USB.
4. Led rojo me indica que hay 5V.
5. Terminal para conectar la fuente interna.
6. Switch ON/OFF para Fuente interna.
7. Switch ON/OFF para escoger el tipo de fuente.
8. Pulsador de reset.
9. Terminales para conectar la fuente externa.
10. Switch ON/OFF para Fuentes externa.
11. Led verde indica que hay voltajes en las fuentes.
12. Terminales para los motores.
13. Buzzer.
14. Sensor de temperatura.
15. Switch para controlar luz de fondo del GLCD.
16. Barra de LEDs.
17. Potenciómetro del contraste del GLCD.
18. Terminales para medir el voltaje.
19. Pantalla GLCD.

FUNCIONAMIENTO DE LA TARJETA ELECTRÓNICA.

Una vez encendida la PC, es muy importante antes de conectar la tarjeta verificar que el puerto de comunicación USB esté conectado a la PC ya que el programa empieza con este puerto en la primera línea de programación de lo contrario se queda en un lazo infinito y puede que no funcione correctamente cada aplicación.

También debo verificar la polaridad para conectar la fuente y luego encender el switch tomando en cuenta estos parámetros, observo que esté titilando el led naranja, que me indica que sí está trabajando el reloj de lo contrario debo igualar el reloj desde la PC en la Interfaz Gráfica dando un clic en igualar reloj y enseguida se observará la fecha y hora en el GLCD.

Una vez lista la Tarjeta Electrónica, la PC y los demás parámetros indicados anteriormente, podemos proceder a ejecutar la Interfaz Gráfica y luego seleccionar cualquiera de las aplicaciones para observarlas en la Interfaz gráfica y en el GLCD.

FUNCIONAMIENTO DE LA INTERFAZ GRÁFICA.

La Interfaz Gráfica es muy fácil de manejar, ejecutamos el programa en el Software Visual C # y enseguida observamos la Interfaz gráfica lista para funcionar la cual posee varios botones los cuales funcionan con solo dar un clic, en algunos botones se debe arrastrar el cursor para variar los valores, seleccionamos la aplicación que deseemos observar, pueden funcionar todas las aplicaciones juntas.

Tenemos la opción de escoger el tipo de comunicación sea USB o RS232, además los valores de cada aplicación que se observan en la Interfaz gráfica también se observan en el GLCD comprobando que funciona correctamente, es importante presionar los botones cuidadosamente de lo contrario podría no responder el programa y habría que ejecutarlo nuevamente.

ANEXO G
PROGRAMAS DE LA TARJETA ELECTRÓNICA.

PROGRAMA DESARROLLADO EN VISUAL C#

PicUSB.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;    //Clase para abrir página web
using System.IO.Ports;

namespace PicUSB
{ public partial class PicUSB : Form
{ PicUSBAPI usbapi = new PicUSBAPI();
  bool flag1;
  string data_recibida;
  public PicUSB()
{ InitializeComponent();
  serialPort1.DataReceived += new
SerialDataReceivedEventHandler(Recibir1);  }
private void Recibir1(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
{ data_recibida = serialPort1.ReadExisting();
  this.Invoke(new EventHandler(Actualizar1)); }
private void Actualizar1(object a, EventArgs e)
{ label11.Text = label11.Text + data_recibida;}
private void tkBarPWM1_Scroll(object sender, EventArgs e)
{ lblPWM1.Text = (tkBarPWM1.Value).ToString("0000"); }
private void tkBarPWM2_Scroll(object sender, EventArgs e)
{ lblPWM2.Text = (tkBarPWM2.Value).ToString("0000");}
private void rdBtHorario1_CheckedChanged(object sender, EventArgs e)
{ string data_temp;
  byte[] send_buf1 = new byte[8];
  byte Sentido1a;
  if (rdBtHorario1.Checked == true)
{ pictVerdeON1.Visible=true;
  pictVerdeOFF1.Visible = false;
  pictVerdeON2.Visible = false;
  pictVerdeOFF2.Visible = true;
  Sentido1a = 1;
  if (flag1)
  usbapi.Sentido1(Sentido1a);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
{ send_buf1[0] = 0x04;
  send_buf1[1] = 0x01;
  data_temp = System.Convert.ToString(send_buf1[0]);
  serialPort1.Write(data_temp);
  usbapi.Retardo();
  data_temp = System.Convert.ToString(send_buf1[1]);
  serialPort1.Write(data_temp);
  usbapi.Retardo(); }}}}
  private void button1_Click(object sender, EventArgs e)
{ string data_temp;
  byte[] send_buf1 = new byte[8];
  if (flag1)
  usbapi.BuzzerON(6);
```

```
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
{ send_buf1[0] = 0x06;
  send_buf1[1] = 0x06;
  data_temp = System.Convert.ToString(send_buf1[0]);
  serialPort1.Write(data_temp);
  usbapi.Retardo();
  data_temp = System.Convert.ToString(send_buf1[1]);
  serialPort1.Write(data_temp);
  usbapi.Retardo();}}}
  private void button2_Click(object sender, EventArgs e)
{ string data_temp;
  byte[] send_buf1 = new byte[8];
  if (flag1)
  usbapi.BuzzerOFF(7);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
{ send_buf1[0] = 0x07;
  send_buf1[1] = 0x07;
  data_temp = System.Convert.ToString(send_buf1[0]);
  serialPort1.Write(data_temp);
  usbapi.Retardo();
  data_temp = System.Convert.ToString(send_buf1[1]);
  serialPort1.Write(data_temp);
  usbapi.Retardo();}}}
  private void button4_Click(object sender, EventArgs e)
{ serialPort1.Close();
  button3.Enabled = true;
  button4.Enabled = false;
  label1.Text = "MODO USB";
  usbapi.Usb(8);
  flag1 = true;}
  private void button3_Click(object sender, EventArgs e)
{ try
{ serialPort1.Open(); //this.Text = "Puerto Serial : Puerto abierto";
  button3.Enabled = false;
  button4.Enabled = true;
  label1.Text = "MODO RS232";
  usbapi.Serial(9);
  flag1 = false;}
  catch (Exception)
{ //this.Text = "Puerto Serial : No se pudo abrir el puerto";
  button3.Enabled = true;
  button4.Enabled = false;
  label1.Text = "Puerto Incorrecto";}}
  private void button5_Click(object sender, EventArgs e)
{ byte x1;
  String Cadena;
  string datoserial;
  int Longitud;
  Cadena = sumando1.Text;
  if (string.IsNullOrEmpty(Cadena))
  Longitud=0;
  else
  Longitud=Cadena.Length;
  if (Longitud < 20)
  Cadena = Cadena.PadRight(20);
  sumando2.Text = "";
  sumando2.Text = Cadena;
```

```
  if (flag1)
  usbapi.EnviarTexto(sumando2.Text);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
  serialPort1.Write("0");
  usbapi.Retardo();
  for (x1 = 0; x1 < 20; x1++)
{ datoserial = (Cadena.Substring(x1, 1));
  serialPort1.Write(datoserial);
  usbapi.Retardo();}}}
  private void DatoBarraLeds_Scroll_1(object sender, EventArgs e)
{ label8.Text = (DatoBarraLeds.Value).ToString("0000");}
  private void button6_Click(object sender, EventArgs e)
{ if (flag1)
  usbapi.DatoLeds(DatoBarraLeds.Value);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
{ serialPort1.Write("1");
  usbapi.Retardo();
  string miles = (label8.Text.Substring(0, 1));
  serialPort1.Write(miles);
  usbapi.Retardo();
  string cente = (label8.Text.Substring(1, 1));
  serialPort1.Write(cente);
  usbapi.Retardo();
  string dece = (label8.Text.Substring(2, 1));
  serialPort1.Write(dece);
  usbapi.Retardo();
  string unid = (label8.Text.Substring(3, 1));
  serialPort1.Write(unid);
  usbapi.Retardo();}}}
  private void button7_Click(object sender, EventArgs e)
{ if (flag1)
  usbapi.PwmDuty1(tkBarPWM1.Value);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
{ serialPort1.Write("2");
  usbapi.Retardo();
  string miles = (lblPWM1.Text.Substring(0, 1));
  serialPort1.Write(miles);
  usbapi.Retardo();
  string cente = (lblPWM1.Text.Substring(1, 1));
  serialPort1.Write(cente);
  usbapi.Retardo();
  string dece = (lblPWM1.Text.Substring(2, 1));
  serialPort1.Write(dece);
  usbapi.Retardo();
  string unid = (lblPWM1.Text.Substring(3, 1));
  serialPort1.Write(unid);
  usbapi.Retardo();}}}
  private void button8_Click_1(object sender, EventArgs e)
{ if (flag1)
  usbapi.PwmDuty2(tkBarPWM2.Value);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
{ serialPort1.Write("3");
  usbapi.Retardo();
```

```
  if (flag1)
  usbapi.Igualar(label10.Text);
  else
{ label11.Text = "";                    usbapi.Retardo();
  if (serialPort1.IsOpen)               serialPort1.Write(dias);
{ serialPort1.Write("A");               usbapi.Retardo();
  usbapi.Retardo();                     serialPort1.Write(diamd);
  serialPort1.Write(segsd);             usbapi.Retardo();
  usbapi.Retardo();                     serialPort1.Write(diamu);
  serialPort1.Write(segsu);             usbapi.Retardo();
  usbapi.Retardo();                     serialPort1.Write(mesd);
  serialPort1.Write(minud);             usbapi.Retardo();
  usbapi.Retardo();                     serialPort1.Write(mesu);
  serialPort1.Write(minuu);             usbapi.Retardo();
  usbapi.Retardo();                     serialPort1.Write(aniod);
  serialPort1.Write(horad);             usbapi.Retardo();
  usbapi.Retardo();                     serialPort1.Write(aniou);
  serialPort1.Write(horau);             usbapi.Retardo();}}}
  private void button10_Click(object sender, EventArgs e)
{ if (flag1)
  usbapi.RotarIN(11);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
  serialPort1.Write("B");
  usbapi.Retardo();}}
  private void button11_Click(object sender, EventArgs e)
{ if (flag1)
  usbapi.RotarDN(12);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
  serialPort1.Write("C");
  usbapi.Retardo();}}
  private void button12_Click(object sender, EventArgs e)
{ if (flag1)
  usbapi.RotarII(13);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
  serialPort1.Write("D");
  usbapi.Retardo();}}
  private void button13_Click(object sender, EventArgs e)
{ if (flag1)
  usbapi.RotarDI(14);
  else
{ label11.Text = "";
  if (serialPort1.IsOpen)
  serialPort1.Write("E");
  usbapi.Retardo();}}
  private void PicUSB_Load(object sender, EventArgs e)
{ button4.Enabled = false;
  label1.Text = "MODO USB";
  flag1 = true;
  lblPWM1.Text = (tkBarPWM1.Value).ToString("0000");
  lblPWM2.Text = (tkBarPWM2.Value).ToString("0000");
  label8.Text = (DatoBarraLeds.Value).ToString("0000");}
  private void label15_Click(object sender, EventArgs e){}
  private void label14_Click(object sender, EventArgs e){}
  private void label15_Click_1(object sender, EventArgs e {}
  private void label5_Click(object sender, EventArgs e){} }}
```

PicUSBAPI.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Runtime.InteropServices; // Clase para importar DLL
using PVOID = System.IntPtr;
using DWORD = System.UInt32;
namespace PicUSB{
unsafe public class PicUSBAPI {
        #region Definición de los Strings: EndPoint y VID_PID
        string vid_pid_norm = "vid_04d8&pid_0011";
        string out_pipe = "\\MCHP_EP1";
        string in_pipe = "\\MCHP_EP1";
        #endregion
        #region Funciones importadas de la DLL: mpusbapi.dll
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDLLVersion();
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDeviceCount(string
pVID_PID);
        [DllImport("mpusbapi.dll")]
        private static extern void* _MPUSBOpen(DWORD instance, string
pVID_PID, string pEP, DWORD dwDir, DWORD dwReserved);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBRead(void* handle, void*
pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBWrite(void* handle, void*
pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBReadInt(void* handle, DWORD*
pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern bool _MPUSBClose(void* handle);
        #endregion

        void* myOutPipe;
        void* myInPipe;
        static void Main()
        { Application.EnableVisualStyles();
          Application.Run(new PicUSB());}
        public void OpenPipes()
        { DWORD selection = 0;
       myOutPipe = _MPUSBOpen(selection, vid_pid_norm, out_pipe, 0, 0)
       myInPipe = _MPUSBOpen(selection, vid_pid_norm, in_pipe, 1, 0);}
        public void ClosePipes()
        { _MPUSBClose(myOutPipe);
          _MPUSBClose(myInPipe);}
        private void SendPacket(byte* SendData, DWORD SendLength)
        { uint SendDelay = 1000;
          DWORD SentDataLength;
          OpenPipes();
          _MPUSBWrite(myOutPipe, (void*)SendData, SendLength,
&SentDataLength, SendDelay);
          ClosePipes();}
private void ReceivePacket(byte* ReceiveData, DWORD *ReceiveLength)
        { int ReceiveDelay=1000;
          DWORD ExpectedReceiveLength = *ReceiveLength;
          OpenPipes();
```

```
          _MPUSBRead(myInPipe, (void*)ReceiveData,
ExpectedReceiveLength, ReceiveLength, ReceiveDelay);
          ClosePipes();}
          public void Retardo()
        { DWORD x111;
          DWORD x222;
          for (x111 = 1; x111 < 6000; x111++)
        { for (x222 = 1; x222 < 6000; x222++){}}}
        public void Retardo1()
        { DWORD x11;
          DWORD x22;
          for (x11 = 1; x11 < 100000; x11++)
        { for (x22 = 1; x22 < 1000; x22++){}}}
          public void EnviarTexto(string sumando2)
        { byte* send_buf = stackalloc byte[8];
          System.Text.Encoding enc = System.Text.Encoding.ASCII;
          string datostring;
          datostring = sumando2;
          byte[] myByteArray = enc.GetBytes(datostring);
          send_buf[0] = 0x00;      // Código de Entrada Enviar TEXTO
            send_buf[1] = myByteArray[0];
            send_buf[2] = myByteArray[1];
            send_buf[3] = myByteArray[2];
            send_buf[4] = myByteArray[3];
            send_buf[5] = myByteArray[4];
            send_buf[6] = myByteArray[5];
            send_buf[7] = myByteArray[6];
            SendPacket(send_buf, 8);
            Retardo1();
            send_buf[0] = 0x00;    // Código de Entrada Enviar TEXTO
            send_buf[1] = myByteArray[7];
            send_buf[2] = myByteArray[8];
            send_buf[3] = myByteArray[9];
            send_buf[4] = myByteArray[10];
            send_buf[5] = myByteArray[11];
            send_buf[6] = myByteArray[12];
            send_buf[7] = myByteArray[13];
            SendPacket(send_buf, 8);
            Retardo1();
            send_buf[0] = 0x00;    // Código de Entrada Enviar TEXTO
            send_buf[1] = myByteArray[14];
            send_buf[2] = myByteArray[15];
            send_buf[3] = myByteArray[16];
            send_buf[4] = myByteArray[17];
            send_buf[5] = myByteArray[18];
            send_buf[6] = myByteArray[19];
            send_buf[7] = myByteArray[19];
            SendPacket(send_buf, 8); } } }
```

PROGRAMA DESARROLLADO EN EL COMPILADOR CCS PARA EL PIC 18F4550.

```
#include <18F4550.h>
#fuses
HSPLL,NOWDT,MCLR,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV4,VREGEN,NO
PBADEN
#use delay(clock=16000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
```

```
#define USB_HID_DEVICE     FALSE             //deshabilitamos el uso
de las directivas HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1)
for IN bulk/interrupt transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1)
for OUT bulk/interrupt transfers
#define USB_EP1_TX_SIZE    1                 //size to allocate for
the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE    8                 //size to allocate for
the rx endpoint 1 buffer

#include <pic18_usb.h>        //Microchip PIC18Fxx5x Hardware layer
for CCS's PIC USB driver
#include <PicUSB.h>           //Configuración del USB y los
descriptores para este dispositivo
#include <usb.c>              //handles usb setup tokens and get
descriptor reports

// Definición del pin del DS18B20 (Sensor de Temperatura)
#define ONE_WIRE_PIN PIN_A4
#include "ds18b20.c"

// Definición de los pines del GLCD Gráfico y sus librerias
#define GLCD_CS1 PIN_D4
#define GLCD_CS2 PIN_D2
#define GLCD_DI PIN_D7
#define GLCD_RW PIN_D6
#define GLCD_E PIN_D5
#define GLCD_RST PIN_D3

#include "HDM64GS12.c"
#include "graphics.c"

unsigned char const
SELLO[8][128] ={
0xFF,0x01,0xFD,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0xF5,0xF5,0xF5,0xF5,
0xF5,0x65,0x45,0x45,0x65,0x35,0x
15,0x15,0x15,0x15,0x15,
0x15,0x15,0x15,0x15,0x35,0x65,0x
45,0x45,0x45,0x45,0xE5,
0xF5,0xF5,0xF5,0xF5,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0x05,0x05,0x
05,0x05,0x05,0x05,0x05,
0x05,0x05,0x05,0x05,0xFD,0x01,0x
FF,0xFF,0x00,0xFF,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0xC0,0xC3,0xFF,0x03,0x
01,0x00,0xC0,0x60,0x98,
0x80,0xE0,0xC0,0x80,0x80,0x00,0x
00,0x00,0x80,0x80,0xE0,
0x40,0x80,0x98,0x60,0xC0,0x00,0x
00,0x01,0x03,0xFF,0xC7,
0xC0,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0xFF,0x00
0xBF,0xFF,0x4F,0x0F,0x0F,0x4F,0x
FF,0xBF,0x3F,0x3F,0x3F,
0x3F,0xBF,0x2F,0xA7,0xA7,0x63,0x
41,0x40,0x40,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0xFF,
0x00,0xFF,0xFF,0x80,0xBF,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A6,0xA2,0xA3,0xAE,0xA6,
0xA6,0xA7,0xA5,0xA6,0xA3,0xA1,0x
A1,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA1,0xA3,0x
A6,0xA4,0xA4,0xA5,0xA6,
0xAE,0xA5,0xA7,0xAE,0xA7,0xAF,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0,0xA0,0x
A0,0xA0,0xA0,0xA0,0xA0,
0xA0,0xA0,0xA0,0xA0,0xA0
0xA0,0xBF,0x80,0xFF
};

// Definición del L293B
#define M1a  PIN_E2
#define M1b  PIN_E1
#define M2a  PIN_E0
#define M2b  PIN_C0

// Definición de los pines del TPIC695
#define SERIN1  PIN_A2
#define SRCLK1  PIN_A3
#define RCLK1   PIN_A5
#define TIP127  PIN_D1

// Definición de los pines del
RTC DS1307
#define RTC_SDA  PIN_A1
#define RTC_SCL  PIN_A0
#include "ds1307.c"
byte sec;
byte min;
byte hrs;
byte day;
byte month;
byte yr;
byte dow;
char sdow[11];
int1 flag1=0;
int1 flag2=0;
int1 flag3=0;
int1 flag4=0;
int1 flag5=1;
int16 barraleds;

#bit dato0 = barraleds.0
#bit dato1 = barraleds.1
#bit dato2 = barraleds.2
#bit dato3 = barraleds.3
#bit dato4 = barraleds.4
#bit dato5 = barraleds.5
#bit dato6 = barraleds.6
#bit dato7 = barraleds.7
#bit dato8 = barraleds.8
#bit dato9 = barraleds.9

#define Buzzer PIN_D0

#define modo      recibe[0]
#define param1    recibe[1]
#define param2    recibe[2]
#define param3    recibe[3]
#define param4    recibe[4]
#define param5    recibe[5]
#define param6    recibe[6]
#define param7    recibe[7]

#define resultado envia[0]

#define dat1    recibe1[0]
#define dat2    recibe1[1]
#define dat3    recibe1[2]
#define dat4    recibe1[3]
#define dat5    recibe1[4]
#define dat6    recibe1[5]
#define dat7    recibe1[6]
#define dat8    recibe1[7]

#define dat9     recibe2[0]
#define dat10    recibe2[1]
#define dat11    recibe2[2]
#define dat12    recibe2[3]
#define dat13    recibe2[4]
#define dat14    recibe2[5]
#define dat15    recibe2[6]
#define dat16    recibe2[7]

#define dat17    recibe3[0]
#define dat18    recibe3[1]
```

```
#define dat19    recibe3[2]
#define dat20    recibe3[3]
#define dat21    recibe3[4]
#define dat22    recibe3[5]
#define dat23    recibe3[6]
#define dat24    recibe3[7]

// Definición de las SUBRUTINAS utilizadas
void DatoTpicNormal(int16 valor1)
{  int16 i;
   output_low(Rclk1);
   output_low(Srclk1);
   for(i=0;i<=9;i++)
{  if( bit_test(valor1,i))
   output_high(Serin1);
   else
   output_low(Serin1);
   output_high(Srclk1);
   output_low(Srclk1); }
   output_high(Rclk1);
   output_low(Rclk1);
   output_low(TIP127);
   return; }
void DatoTpicInvertido(int16 valor1)
{  int16 i;
   output_low(Rclk1);
   output_low(Srclk1);
   for(i=0;i<=9;i++)
{  if( bit_test(valor1,i))
   output_low(Serin1);
   else
   output_high(Serin1);
   output_high(Srclk1);
   output_low(Srclk1); }
   output_high(Rclk1);
   output_low(Rclk1);
   output_low(TIP127);
   return; }

void displayTemperatura(float
temp1)
{  char Temperatura[9];
   sprintf(Temperatura,
"%f",temp1 );   // Convierte
Temperatura a Texto
   Temperatura[5] = '\0';
// Limita la muestra a 4 Digitos
   CHAR text[ ]="Temperatura= ";
// comentario asignado a la
variable "text"
   glcd_text57(1,1,text,1,1);
// muestra el contenido de text
   glcd_rect(80, 1, 110, 8, YES,
OFF);       // Clear anterior
Dato de Temperatura

glcd_text57(80,1,temperatura,1,1
);          // Nuevo dato de
temperartura
   CHAR text2[ ]=" oC ";
// comentario asignado a la
variable "text2"
   glcd_text57(110,1,text2,1,1);
// muestra el contenido de text2
   return; }
void
lee_y_transmite_date_and_time(vo
id)
{  CHAR textTEMP[30];

ds1307_get_day_of_week((char*)
sdow);   // Lee dia de la semana

ds1307_get_date(day,month,yr,dow
);    // Lee la fecha
   ds1307_get_time(hrs,min,sec);
// Lee el Tiempo
   sprintf(textTEMP,"%s
%02u/%02u/%02u",sdow,day,month,y
r); // Da formato al texto de la
Fecha
   glcd_rect(10, 46, 120, 63,
YES, ON);       // Clear
anterior Dato de Fecha y Hora
   glcd_text57(12, 47, textTEMP,
1, OFF);        // Escribe nuevo
Dato de Fecha

sprintf(textTEMP,"%02u:%02u:%02u
",hrs,min,sec); // Da formato al
texto de la Hora
   glcd_text57(40, 56, textTEMP,
1, OFF);          // Escribe
nuevo Dato de la Hora }

void DatoBarraLeds(int16 Dato)
{  int8 i;
```

```
   barraleds=dato;
   glcd_rect(5, 38, 127, 44,
yes, Off);
   glcd_rect(5, 38, 127, 44, no,
Off);
   for( i = 0 ; i < 10 ; i ++ )
{ switch(i)
{ case 9:
  if (dato9)
  glcd_rect(5, 38, 15, 44, yes,
On);
  else
  glcd_rect(5, 38, 15, 44, no,
On);
  break;
  case 8:
  if (dato8)
  glcd_rect(17, 38, 27, 44, yes,
On);
  else
  glcd_rect(17, 38, 27, 44, no,
On);
  break;
  case 7:
  if (dato7)
  glcd_rect(29, 38, 39, 44, yes,
On);
  else
  glcd_rect(29, 38, 39, 44, no,
On);
  break;
  case 6:
  if (dato6)
  glcd_rect(41, 38, 51, 44, yes,
On);
  else
  glcd_rect(41, 38, 51, 44, no,
On);
  break;
  case 5:
  if (dato5)
  glcd_rect(53, 38, 63, 44, yes,
On);
  else
  glcd_rect(53, 38, 63, 44, no,
On);
  break;
  case 4:
  if (dato4)
  glcd_rect(65, 38, 75, 44, yes,
On);
  else
  glcd_rect(65, 38, 75, 44, no,
On);
  break;
  case 3:
  if (dato3)
  glcd_rect(77, 38, 87, 44, yes,
On);
  else
  glcd_rect(77, 38,87, 44, no,
On);
  break;
  case 2:
  if (dato2)
  glcd_rect(89, 38, 99, 44, yes,
On);
  else
  glcd_rect(89, 38, 99, 44, no,
On);
  break;
  case 1:
  if (dato1)
  glcd_rect(101, 38, 111, 44,
yes, On);
  else
  glcd_rect(101, 38, 111, 44,
no, On);
  break;
  case 0:
  if (dato0)
  glcd_rect(113, 38, 123, 44,
yes, On);
  else
  glcd_rect(113, 38, 123, 44,
no, On);
  break; }}}
```

```
char Keypress=' ';
char dat_serial=' ';
int8 mode;
int8 parametro;
int16 parametro1;
int16 parametro2;
int8 p1;
int8 p2;
int8 p3;
int8 p4;
int8 p5;
int8 p6;
int8 p7;
int8 p8;
CHAR text10a[30];
// Interrupcion Serial (RS232)
```

```
#int_rda
void serial_isr()
{ Keypress=0x00;
  Keypress=getc();
  if(Keypress!=0x00)
{ putchar(keypress);
  switch (keypress)
{ case '0':
  text10a="";
  for(p8=0;p8<=19;p8++)
{ dat_serial=getc();
  putchar(dat_serial);
  text10a[p8]=dat_serial; }
  text10a[20]='\0';
  glcd_rect(1, 29, 127, 37, yes,
off);
```

```
  glcd_text57(2,30,text10a,1,1);    // muestra el contenido de text10
  break;                              dat_serial=getc();
  case '1':                           putchar(dat_serial);
  mode=1;                             parametro2=dat_serial-48;
  parametro1=0;
  dat_serial=getc();                parametro1=parametro1+(parametro
  putchar(dat_serial);              2*100);
  parametro2=dat_serial-48;           dat_serial=getc();
  parametro1=parametro2*1000;         putchar(dat_serial);
  dat_serial=getc();                  parametro2=dat_serial-48;
  putchar(dat_serial);
  parametro2=dat_serial-48;         parametro1=parametro1+(parametro
                                    2*10);
parametro1=parametro1+(parametro      dat_serial=getc();
2*100);                               putchar(dat_serial);
  dat_serial=getc();                  parametro2=dat_serial-48;
  putchar(dat_serial);
  parametro2=dat_serial-48;         parametro1=parametro1+parametro2
                                    ;
parametro1=parametro1+(parametro      break;
2*10);                                case '4':
  dat_serial=getc();                  mode=4;
  putchar(dat_serial);                dat_serial=getc();
  parametro2=dat_serial-48;           putchar(dat_serial);
                                      parametro=dat_serial-48;
parametro1=parametro1+parametro2      break;
;                                     case '5':
  break;                              mode=5;
  case '2':                           dat_serial=getc();
  mode=2;                             putchar(dat_serial);
  parametro1=0;                       parametro=dat_serial-48;
  dat_serial=getc();                  break;
  putchar(dat_serial);                case '6':
  parametro2=dat_serial-48;           mode=6;
  parametro1=parametro2*1000;         dat_serial=getc();
  dat_serial=getc();                  putchar(dat_serial);
  putchar(dat_serial);                parametro=6;
  parametro2=dat_serial-48;           break;
                                      case '7':
parametro1=parametro1+(parametro      mode=7;
2*100);                               dat_serial=getc();
  dat_serial=getc();                  putchar(dat_serial);
  putchar(dat_serial);                parametro=7;
  parametro2=dat_serial-48;           break;
                                      case '8':
parametro1=parametro1+(parametro      mode=8;
2*10);                                dat_serial=getc();
  dat_serial=getc();                  putchar(dat_serial);
  putchar(dat_serial);                parametro=8;
  parametro2=dat_serial-48;           break;
                                      case 'A':
parametro1=parametro1+parametro2      mode=10;
;                                     parametro1=0;
  break;                              dat_serial=getc();
  case '3':                           putchar(dat_serial);
  mode=3;                             parametro2=dat_serial-48;
  parametro1=0;                       parametro1=parametro2*10;
  dat_serial=getc();                  dat_serial=getc();
  putchar(dat_serial);                putchar(dat_serial);
  parametro2=dat_serial-48;           parametro2=dat_serial-48;
  parametro1=parametro2*1000;
```

```
parametro1=parametro1+parametro2
;
  p1=parametro1;

//////////////////////////////
  parametro1=0;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;

parametro1=parametro1+(parametro
2*10);
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;

parametro1=parametro1+parametro2
;
  p2=parametro1;

//////////////////////////////
  parametro1=0;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;
  parametro1=parametro2*10;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;

parametro1=parametro1+parametro2
;
  p3=parametro1;

//////////////////////////////
  parametro1=0;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;
  parametro1=parametro2;
  p4=parametro1;

//////////////////////////////
  parametro1=0;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;
  parametro1=parametro2*10;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;


//// Programa principal ////
void main(void) {
   SET_TRIS_A( 0b00010000 );
   SET_TRIS_B( 0b00000000 );
   SET_TRIS_C( 0b10000000 );
   SET_TRIS_D( 0b00000000 );
   SET_TRIS_E( 0b00000000 );
```

```
parametro1=parametro1+parametro2
;
  p5=parametro1;

//////////////////////////////
  parametro1=0;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;

parametro1=parametro1+(parametro
2*10);
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;

parametro1=parametro1+parametro2
;
  p6=parametro1;

//////////////////////////////
  parametro1=0;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;
  parametro1=parametro2*10;
  dat_serial=getc();
  putchar(dat_serial);
  parametro2=dat_serial-48;

parametro1=parametro1+parametro2
;
  p7=parametro1;

//////////////////////////////
  break;
  case 'B':
  mode=11;
  putchar('B');
  break;
  case 'C':
  mode=12;
  putchar('C');
  break;
  case 'D':
  mode=13;
  putchar('D');
  break;
  case 'E':
  mode=14;
  putchar('E');
  break; }}}
```

```
   disable_interrupts(global);
   disable_interrupts(int_rda);
   setup_adc_ports(NO_ANALOGS);                  // Deshabilitado ADC
   setup_adc(ADC_OFF);                           // ADC off
   setup_spi(FALSE);                             // Deshabilitado SPI
   setup_psp(PSP_DISABLED);                      // Deshabilitado PSP
   setup_ccp1(ccp_pwm);                          // Habilita PWM1
   setup_ccp2(ccp_pwm);                          // Habilita PWM2
   setup_comparator(NC_NC_NC_NC);                // Deshabilitado
Comparadores
   setup_vref(FALSE);                            // Deshabilitado
voltaje de referencia
   setup_timer_2(T2_DIV_BY_16,248,1);            // Setea Periodo de
1KHz al timer2
   output_low(M1a);                              // Motor 1a apagado
   output_low(M1b);                              // Motor 1b apagado
   output_low(M2a);                              // Motor 2a apagado
   output_low(M2b);                              // Motor 2b apagado
   output_high(TIP127);                          // Barra de Led's
apagada
   output_low(Buzzer);                           // Buzzer apagado
   float temp;                                   // Variable para leer
la Temperatura

    glcd_text57(7,5,text,1,1);                // muestra el contenido
de text
    glcd_text57(40,17,text2,1,1);             // muestra el contenido
de text2
    glcd_text57(10,29,text3,1,1);             // muestra el contenido
de text3
    glcd_text57(30,41,text4,1,1);             // muestra el contenido
de text4
    glcd_text57(50,53,text5,1,1);             // muestra el contenido
de text5

    delay_ms(2000);                           // Espera de dos segundos
    glcd_fillScreen(OFF);                     // Se limpia GLCD
    onewire_reset();                          // Reset Bus ONEWIRE
    ds1307_init(DS1307_OUT_ON_DISABLED_HIHG | DS1307_OUT_ENABLED |
DS1307_OUT_1_HZ);
    DatoBarraLeds(0);
    GLCD_circle(123,13,4,yes,off);
    GLCD_circle(123,13,4,no,on);
    text1="BUZZER";                           // comentario asignado a
la variable "text"
    glcd_text57(77,10,text1,1,1);             // Nuevo dato de Buzzer
    GLCD_circle(123,23,4,yes,on);
    text1="Usb    ";                          // comentario asignado a
la variable "text"
    glcd_text57(77,20,text1,1,1);             // Nuevo dato de USB

////////////////////////////////////////////////////////////////////
 while (TRUE)
{ if (x<10)                                 DatoBarraLeds(~y);
 { if (i2<1)                                DatoTpicInvertido(y); }}
 { //delay_ms(1);                           else
   i2=i2+1; }                              { x=0;
   else                                     y=512;
 { y=y>>1;                                  DatoBarraLeds(~y);
   x=x+1;                                   DatoTpicInvertido(y); }}
   i2=0;
```

```
////////////////////////////////
////////////////////////////////
//////
   if(usb_enumerated()) // si el
PicUSB está configurado
 { if (usb_kbhit(1))//si el
endpoint de salida contiene
datos del host
 { usb_get_packet(1,recibe,8);
// cojemos el paquete de tamaño
8bytes del EP1 y almacenamos en
recibe
   if (modo == 0)       //
Modo_Recibir Texto
 { switch(contador)
 { case 0:
   dat1=param1;
   dat2=param2;
   dat3=param3;
   dat4=param4;
   dat5=param5;
   dat6=param6;
   dat7=param7;
   break;
   case 1:
   dat9=param1;
   dat10=param2;
   dat11=param3;
   dat12=param4;
   dat13=param5;
   dat14=param6;
   dat15=param7;
   break;
   case 2:
   dat17=param1;
   dat18=param2;
   dat19=param3;
   dat20=param4;
   dat21=param5;
   dat22=param6;
   dat23=param7;
   break; }
   contador=contador+1;
   if (contador==3)
  { contador=0;
   glcd_rect(1, 29, 127, 37,
yes, off);
   text10="";

sprintf(text10,"%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",dat1,dat2,da
t3,dat4,dat5,dat6,dat7,dat9,dat10,dat11,dat12,dat13,dat14,dat15,dat17,
dat18,dat19,dat20,dat21,dat22);

glcd_text57(2,30,text10,1,1);   // muestra el contenido de text10}}
   if (modo == 1) // Modo_Barra_Leds
{  flag1=0;
   flag2=0;
   flag3=0;
   flag4=0;
   x=0;
   y=0;
   i2=0;
   sprintf(text1,"%lu",valor);
   CHAR text[ ]="PWM1= ";  // comentario asignado a la variable "text"
   glcd_text57(1,10,text,1,1);    // muestra el contenido de text
   glcd_rect(32, 10, 55, 18, YES, OFF);   // Clear anterior Dato pwm1
   glcd_text57(32,10,text1,1,1); // Nuevo dato de pwm1}
   if (modo == 3)                    // pwm2
{  valor = param1 + (param2*256);
   sprintf(text1,"%lu",valor);
   CHAR text[ ]="PWM2= "; // comentario asignado a la variable "text"
   glcd_text57(1,20,text,1,1); // muestra el contenido de text
   glcd_rect(32, 20, 55, 28, YES, OFF); // Clear anterior Dato pwm2
   glcd_text57(32,20,text1,1,1);  // Nuevo dato de pwm2}
{ switch(param1)
 { case 1:
   output_high(M1a);
   output_low(M1b);
 glcd_rect(60, 20, 72, 28, YES, OFF);//Clear anterior Dato de Sentido2
   text1=" H";   // comentario asignado a la variable "text"
   glcd_text57(60,20,text1,1,1); // Nuevo dato de Sentido2
   break;
   case 2:
   output_low(M1a);
   output_high(M1b);
```

```
    output_low(M2b);
    break; }}
    if (modo == 6) // Buzzer ON
   { switch(param1)
   { case 6:
    output_high(Buzzer);
    GLCD_circle(123,13,4,yes,on);
    text1="BUZZER"; // comentario asignado a la variable "text"

    glcd_text57(77,10,text1,1,1); // Nuevo dato de Buzzer
   { switch(param1)
   { case 7:
  output_low(Buzzer);
  GLCD_circle(123,13,4,yes,off);
  GLCD_circle(123,13,4,no,on);


  text1="BUZZER"; // comentario asignado a la variable "text"
  glcd_text57(77,10,text1,1,1); // Nuevo dato de Buzzer break; }}
  if (modo == 8)  // USB ON
{ switch(param1)
{ case 8:
| DS1307_OUT_ENABLED |DS1307_OUT_1_HZ);
  ds1307_set_date_time(p5,p6,p7,p4,p3,p2,p1);}
  if (modo == 11)  // ROTAR BARRA DE LED'S IZQUIERDA NORMAL
{ flag1=1;
  flag2=0;
  flag3=0;
  flag4=0;
  y=1;
  x=0;
  i2=0;
  DatoBarraLeds(y);
  DatoTpicNormal(y); }
  if (modo == 12) // ROTAR BARRA DE LED'S DERECHA NORMAL
  flag1=0;
  flag2=1;
  flag3=0;
  flag4=0;
  i2=0;
  DatoBarraLeds(y);
  DatoTpicNormal(y); }
  if (modo == 13) // ROTAR BARRA DE LED'S IZQUIERDA INVERTIDO
{ flag1=0;
  flag2=0;
  flag3=1;
  flag4=0;
  y=1;
  x=0;
  i2=0;
  DatoBarraLeds(~y);
  DatoTpicInvertido(y); }
  if (modo == 14) // ROTAR BARRA DE LED'S DERECHA INVERTIDO
  flag4=1;
  y=512;
  x=0;
  i2=0;
  DatoBarraLeds(~y);
  DatoTpicInvertido(y); }
  mode=255; } } } } }
```