UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

INGENIERÍA INDUSTRIAL

"DESARROLLO DE APLICACIONES Y PUESTA EN MARCHA DEL SISTEMA EASYPIC PARA MICROCONTROLADORES PIC"

PROYECTO FIN DE CARRERA

Enero – 2008

AUTOR: María de las Nieves Robles Botella

DIRECTOR: María Asunción Vicente Ripoll

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

DESARROLLO DE APLICACIONES Y PUESTA EN MARCHA DEL SISTEMA EASYPIC PARA MICROCONTROLADORES PIC

Proyectante: MARÍA DE LAS NIEVES ROBLES BOTELLA

Director: MARÍA ASUNCIÓN VICENTE RIPOLL

VºBº director/es del proyecto:

Fdo.: Fdo.:

Lugar y fecha: _____

CALIFICACIÓN NUMÉRICA ☐ MATRÍCULA DE HONOR ☐

Méritos justificativos en el caso de conceder Matrícula de Honor:

_____

_____

_____

_____

Conforme presidente:	Conforme secretario:	Conforme vocal:
Fdo.:	Fdo.:	Fdo.:

Lugar y fecha: _____

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS

El presente Proyecto Fin de Carrera se centra en el desarrollo de aplicaciones prácticas para la docencia en la programación de microcontroladores. En concreto, el trabajo ha sido realizado con microcontroladores PIC de gama media, en particular con los modelos PIC16F876A y PIC16F877A.

En los últimos años han tenido un gran auge los microcontroladores PIC fabricados por Microchip Technology Inc. Los PIC (Peripheral Interface Controller) son una familia de microcontroladores que ha tenido gran aceptación y desarrollo en los últimos años gracias a que sus buenas características, bajo precio, reducido consumo, pequeño tamaño, gran calidad, fiabilidad y abundancia de información, los convierten en muy fáciles, cómodos y rápidos de utilizar.

El trabajo realizado en este proyecto se puede dividir en 4 tareas básicas:

- Evaluación o puesta en marcha de un entrenador para PICs, en concreto, la tarjeta EasyPIC4.
- Desarrollo de programas fuente para el uso de los módulos de salida LCD y GLCD, presentes en dicho entrenador.
- Creación de un entorno de simulación o circuito virtual, que permite realizar de forma virtual las prácticas de laboratorio desde cualquier ordenador y en cualquier momento.
- Evaluación del microbot Moway y desarrollo de algoritmos de lucha entre robots.

A continuación se describe la estructura de la presente memoria, que consta de nueve capítulos:

En este primer capítulo se describen, como se ha visto, los objetivos del proyecto.

En el Capítulo 2 se realiza una introducción a la arquitectura de los microcontroladores en general, y se describe con detalle la familia de los microcontroladores PIC.

En el Capítulo 3 se analiza la tarjeta de evaluación EasyPIC4, se presenta su arquitectura física y el software necesario para hacer uso de ella.

En el Capítulo 4 se presentan los distintos programas que se han utilizado a lo largo del proyecto, tanto para la grabación de los microcontroladores, como para la depuración y simulación del código fuente.

En el Capítulo 5 se describe la metodología realizada durante la evaluación o puesta en marcha del sistema EasyPIC4.

En el Capítulo 6 se detallan las aplicaciones desarrolladas para los módulos de salida LCD y GLCD presentes en la tarjeta.

En el Capítulo 7 se presentan los circuitos virtuales representativos de la EasyPIC4 que permiten su simulación a través del software Proteus.

En el Capítulo 8 se muestra otra aplicación de los PICs, que puede ser más atractiva para los estudiantes, como es el control de un microbot basado en un PIC16F876A.

En el Capítulo 9 se hace una recopilación de conclusiones, así como los posibles trabajos futuros.

Por último, al final del documento se encuentran los Anexos A, B y C. El Anexo A, contiene los drivers desarrollados tanto para el módulo LCD como para el display GLCD. En el Anexo B, se muestra el estudio temporal realizado de la aplicación cronómetro.c, donde se explica, detalladamente, las funciones utilizadas y su duración. Mientras que el Anexo C contiene diferentes

aplicaciones desarrolladas con el módulo GLCD, las cuales han servido de base para el desarrollo de la aplicación “tension_RA2.c”.

CAPÍTULO 2. INTRODUCCIÓN A LOS MICROCONTROLADORES

2.1. INTRODUCCIÓN

Hace unos años, los sistemas de control se implementaban usando exclusivamente lógica de componentes, lo que hacía que fuesen dispositivos de gran tamaño y muy pesados. Para facilitar una velocidad más alta y mejorar la eficiencia de estos dispositivos de control, se trató de reducir su tamaño apareciendo así los microprocesadores. Siguiendo con el proceso de miniaturización, el siguiente paso consistió en la fabricación de un controlador que integrase todos sus componentes en un solo chip. A esto se le conoce con el nombre de microcontrolador.

2.1.1. DIFERENCIA ENTRE UN MICROCONTROLADOR Y UN MICROPROCESADOR

Un microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (CPU) de un computador, que consta de la Unidad de Control y del Camino de Datos. Mientras que un microcontrolador es un circuito integrado programable que contiene todos los bloques necesarios para controlar el funcionamiento de una tarea determinada:

- Procesador o CPU: que interpreta las instrucciones de programa.
- Memoria RAM para almacenar las variables necesarias.
- Memoria EPROM/PROM/ROM para el programa tipo.

- Puertos E/S para comunicarse con el exterior.
- Diversos módulos para el control de periféricos.
- Generador de impulsos de reloj que sincroniza el funcionamiento de todo el sistema.

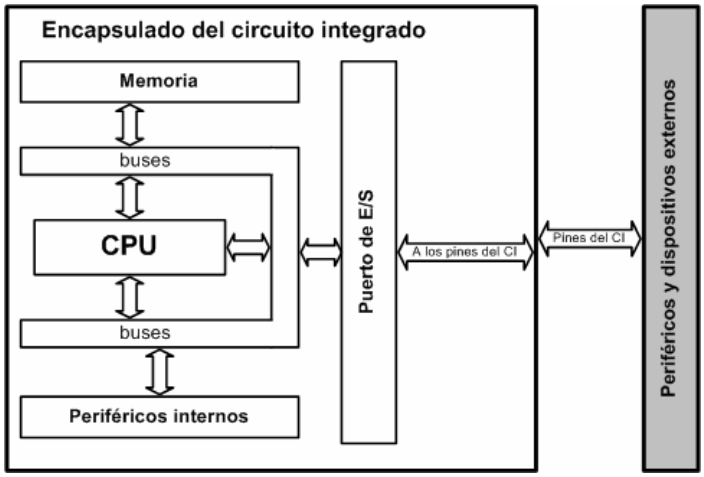


Figura 2.1: Estructura básica de un microcontrolador

En la Figura 2.1, se puede ver al microcontrolador embebido dentro de un encapsulado de circuito integrado, con su procesador, buses, memoria, periféricos y puertos de entrada/salida. Fuera del encapsulado se ubican otros circuitos para completar periféricos internos y dispositivos que pueden conectarse a los pines de entrada/salida.

Un microcontrolador es, por tanto, un circuito o chip que incluye en su interior las tres unidades funcionales de un ordenador: CPU, Memoria y Unidades de E/S, es decir, se trata de un computador completo en un solo circuito integrado.

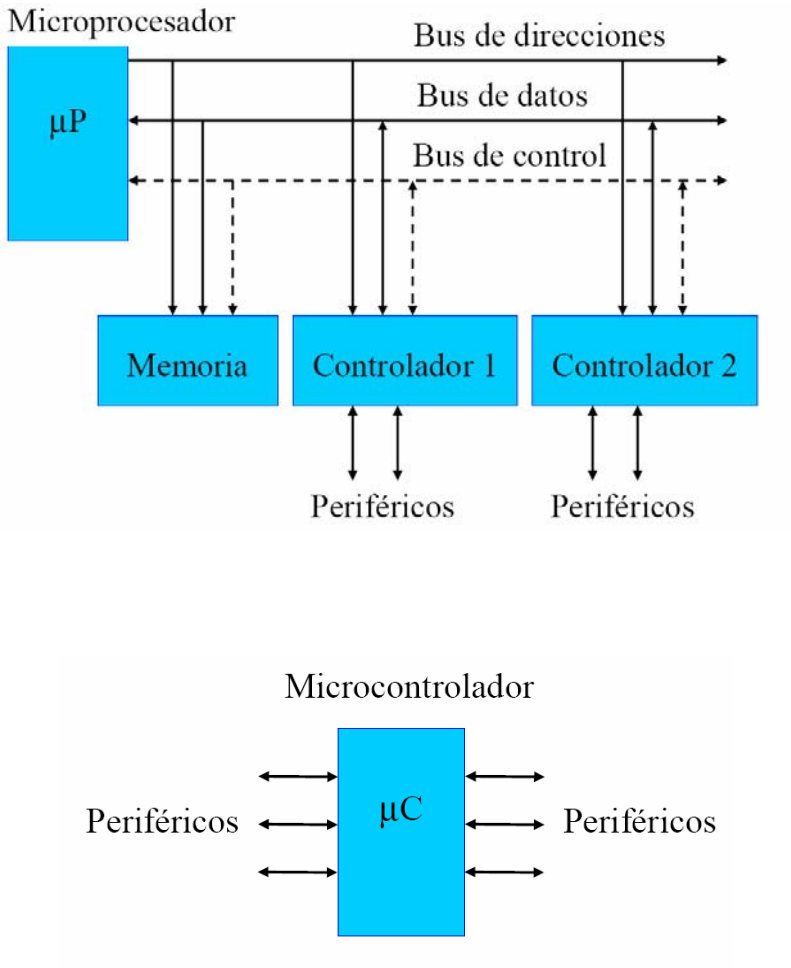


Figura 2.2: Microprocesador y Microcontrolador

Aparentemente el microcontrolador es un dispositivo más complejo que el microprocesador, pero la mayoría de las veces no suele ser así. El microprocesador con bastantes millones de transistores implementa una CPU muy potente, mientras que el microcontrolador con menor cantidad de transistores soporta a un computador completo, pero bastante sencillo. Para construir un computador alrededor de un microprocesador se precisan otros circuitos integrados que lo doten de memoria, de módulos E/S y de recursos complementarios. Todos estos componentes se adaptan al microprocesador a través de los buses de direcciones, de datos e instrucciones y de control, pudiendo ampliar y modificar fácilmente la capacidad y configuración del sistema de memoria, de módulos de E/S y de recursos auxiliares.

Por tanto, las principales características que diferencian a un microcontrolador de un microprocesador son:

- Son sistemas cerrados, pues contienen todos los elementos de un computador en un solo chip, frente a los microprocesadores que son sistemas abiertos, ya que sacan las líneas de los buses de datos, direcciones y control al exterior, para la conexión de memorias, interfaces de E/S, etc.
- Son de propósito específico, es decir, son programados para realizar una única tarea, mientras que los microprocesadores son de propósito general.

Figura 2.3: Microprocesador

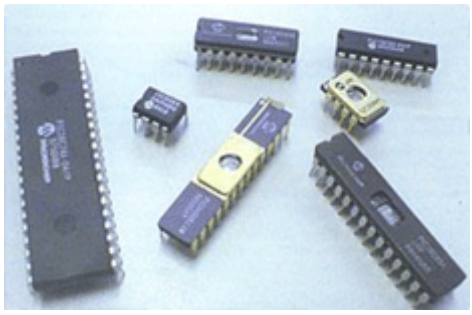


Figura 2.4: Microcontrolador

Debido a que los microcontroladores sólo incluyen las características específicas para una tarea, su coste es relativamente bajo. Un microcontrolador típico realiza funciones de manipulación de instrucciones, posee E/S de accesos fáciles y directos, y un proceso de interrupciones rápido y eficiente. Además también reducen de manera notable los costes de diseño.

2.1.2. APLICACIONES DE LOS MICROCONTROLADORES

El mercado de los microcontroladores está creciendo cada año y parece no tener barreras. 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 producidas de un modelo determinado en una semana. Este dato puede dar una idea de la masiva utilización de estos componentes.

Los microcontroladores a menudo se encuentran en aplicaciones domésticas y en equipos informáticos tales como: microondas, refrigeradores, lavadoras, televisión, equipos de música, ordenadores, impresoras, módems, lectores de discos, etc.

La industria de la automoción ha mostrado una avidez descomunal en el empleo de los microcontroladores, pudiendo haber del orden de 50 microcontroladores en un automóvil: frenos, seguridad, alarmas, climatización, computadores de viaje, etc. Las comunicaciones y sus sistemas de transferencia de información son unos de los campos que más productos fabrica con microcontroladores.

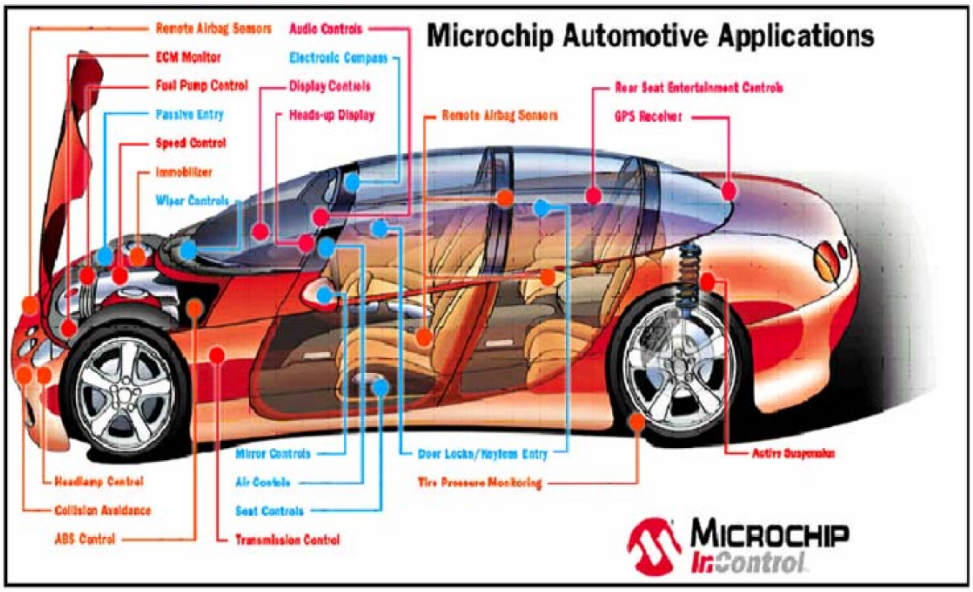


Figura 2.5: Microcontroladores presentes en un automóvil

Los microcontroladores también son muy utilizados en robótica, donde la comunicación entre controladores es muy importante. Esto hace posible muchas tareas específicas al distribuir un gran número de microcontroladores por todo el sistema. La comunicación entre cada microcontrolador y uno central, permite procesar la información por un ordenador central, o transmitirlo a otros microcontroladores del sistema.

Figura 2.6: Robots Moway

Otro ejemplo de aplicación de los microcontroladores, es la de la utilización para monitorizar y gravar parámetros medioambientales (temperatura, humedad, precipitaciones, etc.). El pequeño tamaño, el bajo consumo de potencia y su flexibilidad hacen de este dispositivo, una herramienta adecuada para este tipo de aplicaciones.

2.2. CARACTERÍSTICAS GENERALES DE LOS MICROCONTROLADORES.

Al estar todos los microcontroladores integrados en un chip, su estructura fundamental y sus características básicas son muy parecidas. Todos deben disponer de los bloques esenciales: procesador, memoria de datos y de instrucciones, líneas de E/S, oscilador de reloj y módulos controladores de periféricos. Sin embargo, cada fabricante intenta enfatizar los recursos más idóneos para las aplicaciones a las que se destinan preferentemente.

En este apartado se hace un recorrido de los recursos que se hallan en todos los microcontroladores, describiendo las diversas alternativas y opciones que pueden encontrarse según el modelo seleccionado.

2.2.1. ARQUITECTURA BÁSICA

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en la actualidad se impone la arquitectura Harvard.

La arquitectura de Von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan tanto datos como instrucciones. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control), simplificando así la lógica del microcontrolador.

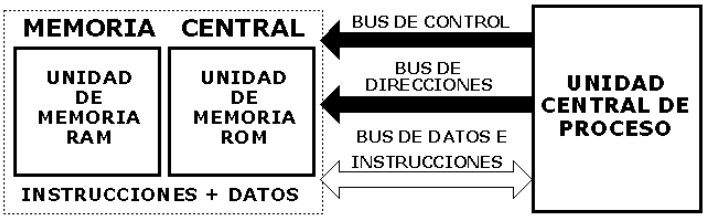


Figura 2.7: Arquitectura Von Neumann

La arquitectura Harvard dispone de dos memorias independientes, una que contiene sólo datos y otra, instrucciones. Ambas disponen de sus respectivos sistemas de buses y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias, consiguiendo que las instrucciones se ejecuten en menos ciclos de reloj.

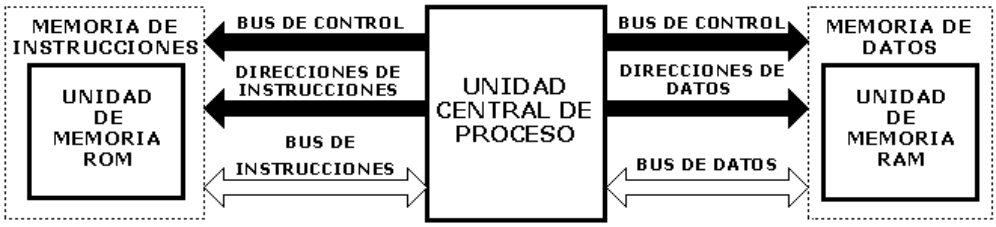


Figura 2.8: Arquitectura Harvard

Esta dualidad de la memoria de datos por un lado y la memoria de programa por otro, permite la adecuación del tamaño de las palabras y los buses a los requerimientos específicos de las instrucciones y los datos.

Se puede concluir que las principales ventajas de la arquitectura Harvard son:

- El tamaño de las instrucciones no está relacionado con el de los datos y, por tanto, puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando una mayor velocidad y una menor longitud de programa.
- El tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

2.2.2. EL PROCESADOR O CPU

El procesador es el elemento más importante del microcontrolador y determina sus principales características, tanto a nivel hardware como software. Se encarga de direccionar la memoria de instrucciones, recibir el código OP (código de operación) de la instrucción en curso, su decodificación y la ejecución de la operación que implica la instrucción, además de la búsqueda de los operandos y el almacenamiento del resultado. Existen tres orientaciones en cuanto a la arquitectura y funcionalidad de los procesadores actuales:

- CISC: Un gran número de procesadores usados en los microcontroladores están basados en la filosofía CISC. Son procesadores con un juego de instrucciones complejo. Su repertorio de instrucciones es elevado donde alguna de las instrucciones son muy sofisticadas y potentes. El problema es que requieren de muchos ciclos de reloj para ejecutar las instrucciones complejas.
- RISC: Tanto la industria de los computadores comerciales como la de los microcontroladores están decantándose hacia la filosofía RISC (Computadores 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

de reloj. La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

- SISC: Estos procesadores poseen un juego de instrucciones específico para cada aplicación. Están destinados a aplicaciones muy concretas, donde el juego de instrucciones, permite adaptarse a las necesidades previstas. Esta filosofía se conoce con el nombre de SISC (Computadores de Juego de Instrucciones Específico).

2.2.3. MEMORIA

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Mientras que otra parte de la memoria será tipo RAM, volátil, destinada a guardar las variables y los datos.

Hay dos peculiaridades que diferencian a los microcontroladores de los computadores personales:

- No existen sistemas de almacenamiento masivo como disco duro o disquetes.
- Como el microcontrolador únicamente se destina a una tarea, en la memoria ROM, sólo hay que almacenar un único programa de trabajo.

La memoria RAM en estos dispositivos es de poca capacidad ya que solamente debe contener las variables y los cambios de información que se produzcan en el transcurso del programa. Por otra parte, como sólo existe un programa activo, no se requiere guardar una copia del mismo en la RAM pues se ejecuta directamente desde la ROM.

Los usuarios de computadores personales están habituados a manejar Megabytes de memoria, pero, los diseñadores con microcontroladores trabajan con capacidades de ROM comprendidas entre 512 bytes y 8 Kbytes, y de RAM comprendidas entre 20 y 512 bytes.

En el caso de la memoria de programa se utilizan diferentes tecnologías, y el uso de una u otra depende de las características de la aplicación a desarrollar. A continuación se describen las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado.

- ROM con máscara: Es una memoria no volátil de sólo lectura cuyo contenido se graba durante la fabricación del chip. El elevado coste del diseño de la máscara sólo hace aconsejable el empleo de los microcontroladores con este tipo de memoria cuando se precisan cantidades superiores a varias miles de unidades.

- OTP: Este tipo de memoria, también es conocida como PROM o simplemente ROM. Los microcontroladores con memoria OTP se pueden programar una sola vez. Es el usuario quien escribe el programa en el chip mediante un sencillo grabador controlado por un programa desde un ordenador. Se utilizan en sistemas donde el programa no requiera futuras actualizaciones y para series relativamente pequeñas, donde la variante de máscara sea muy costosa; también resulta útil para sistemas que requieren serialización de datos, almacenados como constantes en la memoria de programas.

- EPROM: Los microcontroladores que disponen de memoria EPROM pueden borrarse y grabarse varias veces. La grabación se realiza, como en el caso de los OTP, con un grabador gobernado desde un ordenador. Si, posteriormente, se desea borrar el contenido, disponen, en su superficie, de una ventana de cristal por la que se somete a la EPROM a una fuente de luz ultravioleta durante varios minutos. Las cápsulas son de material cerámico con una ventana de vidrio desde la cual puede verse la oblea de silicio del microcontrolador. Estos microcontroladores son más caros que los microcontroladores con memoria OTP que están hechos con material plástico.

- EEPROM: Se trata de memorias de sólo lectura, programables y borrables eléctricamente. Tanto la programación como el borrado se realizan eléctricamente desde el propio grabador y bajo el control programado de un ordenador, por lo que la ventana de cristal de cuarzo

y los encapsulados cerámicos no son necesarios. La operación de grabado y borrado es muy cómoda y rápida.

Los microcontroladores dotados de memoria EEPROM, una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se desee sin ser retirados de dicho circuito. Para ello se utilizan “grabadores en circuito” que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo.

El número de veces que se puede grabar y borrar una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Son idóneos para la enseñanza y la Ingeniería de diseño. En los fabricantes, se va extendiendo la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables, para guardar y modificar cómodamente una serie de parámetros, que adecuan el dispositivo a las condiciones del entorno.

- FLASH: En el campo de las memorias reprogramables para microcontroladores, son el último avance tecnológico en uso a gran escala, y han sustituido a los microcontroladores con memoria EEPROM. Se trata de una memoria no volátil, de bajo consumo, que puede ser escrita y borrada. Funciona como una ROM y una RAM pero consume menos y es más pequeña. A diferencia de la ROM, la memoria FLASH es programable en el circuito. Es más rápida y de mayor densidad que sus predecesoras, lo cual, permite incrementar la cantidad de memoria de programa a un coste bastante bajo.

  La alternativa FLASH se recomienda frente a la EEPROM cuando se precisa gran cantidad de memoria de programa no volátil, ya que es más veloz y tolera más ciclos de escritura/borrado.

  Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados “en circuito”, es decir, sin tener que sacar el circuito integrado de la tarjeta. Así, un dispositivo con este tipo de memoria incorporado al control del motor de un automóvil, permite que se pueda modificar el programa

durante la rutina de mantenimiento periódico, compensando los desgastes y otros factores tales como la compresión, la instalación de nuevas piezas, etc. La reprogramación del microcontrolador puede convertirse en una labor rutinaria dentro de la puesta a punto.

Lo más habitual es encontrar la memoria de programa y datos ubicada, toda, dentro del microcontrolador, de hecho, actualmente son pocos los microcontroladores que permiten conectar memoria de programas en el exterior del encapsulado. Las razones para estas “limitaciones” se deben a que el objetivo fundamental es obtener la mayor integración posible, y el conectar memorias externas consume líneas de E/S que son uno de los recursos más preciados de los microcontroladores.

A pesar de lo comentado anteriormente, existen familias como la INTEL 51 cuyos microcontroladores tienen la capacidad de ser expandidos en una variada gama de configuraciones para el uso de memoria de programas externa. En el caso de los microcontroladores PIC, estas posibilidades están limitadas sólo a algunos microcontroladores de la gama alta.

La Figura 2.9 muestra algunas de las configuraciones de la memoria de programa que se puede encontrar en los microcontroladores. La configuración (a) es la típica y se encuentra casi en el 100% de los microcontroladores. La configuración (b) es poco frecuente y generalmente se logra configurando al microcontrolador para sacrificar la memoria de programa interna. La configuración (c) es la que se encuentra habitualmente en los microcontroladores que tienen posibilidades de expandir su memoria de programa como algunos PIC de gama alta.

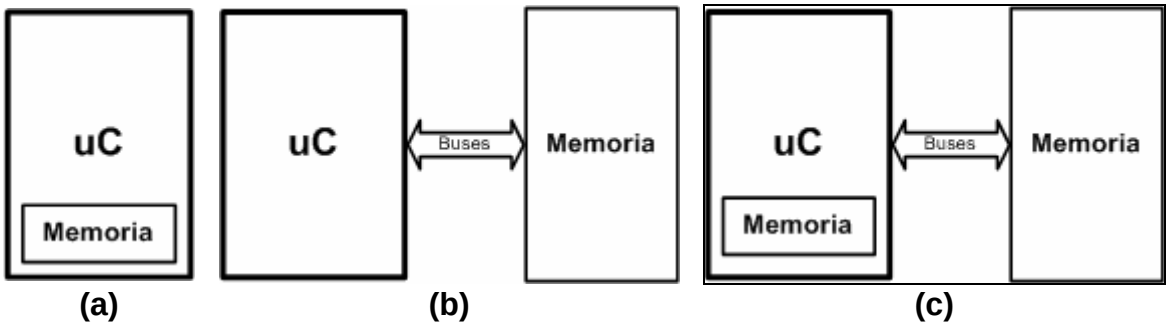


Figura 2.9: Algunas configuraciones de la memoria de programa en microcontroladores

Cuando se requiere aumentar la cantidad de memoria de datos, lo más frecuente es colocar dispositivos de memoria externa en forma de periféricos, de esta forma se pueden utilizar memorias RAM, FLASH o incluso discos duros como los de las computadoras. Mientras que para los cálculos y demás operaciones que requieran almacenamiento temporal de datos, se utiliza la memoria RAM interna del microcontrolador. Esta forma de expandir la memoria de datos viene determinada, en la mayoría de lo casos, por el tipo de repertorio de instrucciones del procesador y por el elevado número de configuraciones distintas, además del consiguiente ahorro de líneas de E/S logrado con el uso de memorias con buses de comunicación serie.

2.2.4. PUERTOS DE ENTRADA Y SALIDA

La principal utilidad de los pines que posee la cápsula que contiene un microcontrolador, es soportar las líneas de E/S que comunican al computador interno con los periféricos exteriores.

Según los controladores de periféricos que posea cada modelo de microcontrolador, las líneas de E/S se destinan a proporcionar el soporte a las señales de entrada, salida y control.

2.2.5. RELOJ PRINCIPAL

Todo microcontrolador requiere de un circuito que le indique la velocidad de trabajo, es el llamado oscilador o reloj. Éste genera una onda cuadrada de alta frecuencia que configura los impulsos de reloj usados como señal para sincronizar todas las operaciones del sistema. Este circuito es muy simple pero de vital importancia para el buen funcionamiento del sistema.

Generalmente, el circuito de reloj está incorporado en el propio microcontrolador y sólo se necesitan unos pocos componentes externos, como cristal de cuarzo junto a elementos pasivos o bien un resonador cerámico o una red RC (resistencia-condensador), para seleccionar y estabilizar la frecuencia de trabajo.

2.2.6. RECURSOS ESPECIALES

En este apartado se detallan los recursos especiales más comunes que pueden poseer los microcontroladores:

- Temporizadores y/o contadores

Se emplean para controlar periodos de tiempo actuando como temporizadores, o para llevar la cuenta de acontecimientos que suceden en el exterior funcionando como contadores.

Con el fin de medir tiempos, se carga el valor adecuado en el registro asociado al temporizador y a continuación, dicho valor se va incrementando o decrementando cada vez que el módulo recibe un pulso, señal de reloj, hasta que se desborde y llegue a 0, momento en el que se produce un aviso.

Cuando se desea contar eventos asociados a cambios de nivel o flancos en alguno de los pines del microcontrolador, el mencionado registro se va incrementando o decrementando al ritmo de dichos impulsos.

- Perro guardián o “Watchdog”

El perro guardián es diseñado para inicializar automáticamente el microcontrolador en el caso de que exista un mal funcionamiento del sistema.

Consiste en un temporizador cuyo objetivo es generar automáticamente un reset cuando se desborda y pasa por 0, a menos que se inhabilite en la palabra de configuración. Cuando el perro guardián está habilitado, se ha de diseñar el programa de manera que refresque o inicialice al perro guardián antes de que se provoque dicho reset. Si el programa falla o se bloquea, no se refrescará al perro guardián y cuando complete su temporización provocará un reset al sistema.

- Protección ante fallo de alimentación o “Brownout”

Se trata de un circuito que resetea al microcontrolador cuando la tensión de alimentación (VDD) es inferior de un determinado valor (*brownout*). El microcontrolador se mantiene reseteado mientras la tensión sea inferior a la de *brownout*, y comienza a funcionar normalmente al sobrepasar dicho valor.

- Estado de reposo o de bajo consumo

Existen diversas situaciones reales de trabajo en los cuales, el microcontrolador debe esperar durante muchos intervalos de tiempo, sin hacer nada, a que se produzca algún acontecimiento externo que le ponga de nuevo en funcionamiento. Durante estos periodos se podría “apagar” el microcontrolador para ahorrar energía. Para ello, los microcontroladores disponen de un modo de funcionamiento de ahorro de energía llamado modo de bajo consumo, reposo, *standby* o *sleep*, en el cual, los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y sus circuitos asociados, quedando el microcontrolador sumido en un profundo “sueño”. Al activarse una interrupción ocasionada por algún acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo.

- Conversor A/D

Como es muy frecuente el trabajo con señales analógicas, éstas deben ser convertidas a digital y por ello muchos microcontroladores incorporan un conversor A/D, el cual se utiliza para tomar datos de varias entradas diferentes que se seleccionan mediante un multiplexor.

Las resoluciones más frecuentes son 8 y 10 bits, aunque hay microcontroladores con conversores de 11 y 12 bits, para resoluciones mayores es preciso utilizar conversores A/D externos. Los conversores A/D son uno de los periféricos más codiciados en el mundo de los microcontroladores y es por ello que muchísimos microcontroladores los incorporan.

- Conversor D/A

Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analógica que saca al exterior por uno de los pines del microcontrolador.

- Comparador analógico

Algunos modelos de microcontroladores disponen internamente de un amplificador operacional que actúa como comparador entre una señal fija de referencia y otra variable que se aplica por uno de los pines de la cápsula. La

salida del comparador proporciona un nivel lógico 1 ó 0 según una señal sea mayor o menor que la otra.

También existen modelos de microcontroladores con un módulo de tensión de referencia que proporciona diversas tensiones de referencia que se pueden aplicar en los comparadores.

Este periférico muy útil para detectar cambios en señales de entrada de las que solamente nos interesa conocer cuando está en un rango determinado de valores.

- Modulador de anchura de impulsos o PWM

Son circuitos que proporcionan en su salida impulsos de anchura variable, que se ofrecen al exterior a través de los pines del encapsulado.

Los PWM son periféricos muy útiles sobre todo para el control de motores, sin embargo hay un grupo de aplicaciones que pueden realizarse con este periférico, dentro de las cuales se encuentra: la conversión digital analógica D/A y el control regulado de luz (*dimming*) entre otras.

- Puertas de E/S digitales

Todos los microcontroladores destinan algunos de sus pines a soportar líneas de E/S digitales. Generalmente, estas líneas se agrupan en puertos de 8 bits de longitud formando Puertas, permitiendo leer datos del exterior o escribir en ellos desde el interior del microcontrolador.

Las líneas digitales de las Puertas pueden configurarse como entrada o como salida cargando un 1 ó un 0 en el bit correspondiente de un registro destinado a su configuración.

- Puertos de comunicación

Los puertos de comunicación son herramientas que dotan al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos. Algunos modelos

disponen de recursos que permiten directamente esta tarea, entre los que destacan:

- UART, adaptador de comunicación serie asíncrona.
- USART, adaptador de comunicación serie síncrona y asíncrona.
- Puerta paralela esclava para poder conectarse con los buses de otros microprocesadores.
- USB, que es un moderno bus serie para los ordenadores.
- Bus I^2C, que es un interfaz serie de dos hilos desarrollado por Philips.
- CAN, para permitir la adaptación con redes de conexionado multiplexado desarrollado conjuntamente por Bosch e Intel para el cableado de dispositivos en automóviles. En EE.UU. se usa el J185O.

2.3. MODELOS DE MICROCONTROLADORES MÁS UTILIZADOS

Existen cientos de fabricantes de microcontroladores. Entre los diversos modelos de microcontroladores se destacan:

Fabricante	Modelos de microcontroladores	Longitud de instrucciones
Atmel [1]	AVR	
Freescale (Motorola) [2]	68HC05, 68HC08, 68HC11 68HC12, 68HC16 683xx	8 bits 16 bits 32 bits
Hitachi [3]	H8	8 bits
Intel [4]	MCS48, MCS51, 8xC251	8 bits

	MCS96, MXS296	16 bits
Microchip [5]	Gama baja (12Cxx) Gama media (12Fxx, 16Cxx y 16Fxx) Gama alta (18Cxx y 18Fxx) dsPIC	12 bits 14 bits 16 bits
National Semiconductor [6]	COP8	8 bits
Philips [7]	80C51	8 bits
Texas Instruments [8]	TMS370, MSP430	8 bits
Western Design Center [9]	W65C02-based W65816-based	8 bits 16 bits
Zilog [10]	Z8, Z86xx	8 bits

Tabla 2.1: Modelos de microcontroladores

2.4. LOS MICROCONTROLADORES PIC

En los últimos años han tenido un gran auge los microcontroladores PIC fabricados por Microchip Technology Inc. [5]. Los PIC son una familia de microcontroladores que ha tenido gran aceptación y desarrollo en los últimos año gracias a que sus buenas características, bajo precio, reducido consumo, pequeño tamaño, gran calidad, fiabilidad y abundancia de información, los convierten en fáciles, cómodos y rápidos de utilizar.

Los microcontroladores PIC fueron los primeros microcontroladores RISC, es decir, microcontroladores con un juego de instrucciones reducido. El hecho de ser procesadores de tipo RISC generalmente implica simplicidad en los diseños, permitiendo más características a bajo coste.

Los principales beneficios de esta simplicidad en el diseño son que los microcontroladores se implementan en chips muy pequeños, con pocos pines,

y tiene un consumo de potencia muy bajo. Una de las propiedades más característica de los microcontroladores PIC, es la utilización de la arquitectura Harvard, frente a la clásica arquitectura Von Neumann que dispone de una sola memoria principal donde se almacenan datos e instrucciones indistintamente, accediendo a ella a través de un sistema de buses único.

La arquitectura Harvard dispone de dos memorias independientes, una que contiene datos y otra instrucciones, con sus respectivos buses de acceso. Esto permite el acceso simultáneo al programa y a los datos, solapando algunas operaciones para mejorar el proceso.

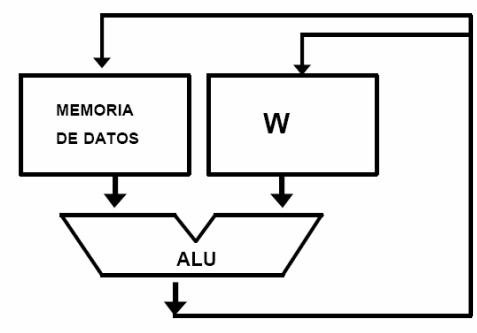
Las principales características derivadas de esta arquitectura son:

- Segmentación de instrucciones o Pipeline: Consiste en dividir la ejecución de las instrucciones en varias fases (en el caso concreto de los PIC, dos fases) de manera que se realizan simultáneamente distintas fases de distintas instrucciones. De esta forma cada ciclo de instrucción abarca tan sólo cuatro ciclos de reloj (un ciclo máquina), exceptuando las instrucciones de salto que ocupan tantos ciclos de instrucción como necesite para calcular la dirección de salto. Es fácil de comprender el hecho de que se inviertan dos ciclos de instrucción en ejecutar instrucciones de salto, ya que la instrucción que hay preparada (en la posición de memoria consecutiva a la actual) no es la correcta y se ha de buscar la correspondiente.

- Formato de instrucciones de longitud constante: Permite optimizar la memoria de instrucciones y facilita notablemente la implementación de ensambladores y compiladores.

- Juego de instrucciones reducido: La CPU que incorporan los PIC es de tipo RISC. Esta es una filosofía de diseño de CPU que da lugar a conjuntos de instrucciones pequeñas y simples que ocupan menor tiempo de CPU.

- Instrucciones ortogonales: Todas las instrucciones pueden manejar cualquier elemento de la arquitectura como fuente o destino de una operación.



- Arquitectura basada en bancos de registro: Todos los elementos que componen el sistema (puertos de E/S, temporizadores, posiciones de memoria, contador de programa, controles, etc.) están implementados físicamente como registros en bloques diferentes llamados bancos.

Dicha arquitectura permite liberar área de silicio para implementar características que mejoren las prestaciones, aportando sencillez y rapidez a los dispositivos. Por lo tanto, los chips resultan más baratos, de menor consumo y con menor número de pines.

La forma de trabajar con los PIC es por medio de ciertos registros llamados Registros de Función Especial (SFR) que contienen tanto la configuración, como el estado de todos los elementos que componen el dispositivo.

A consecuencia de tener una CPU de tipo RISC, la introducción de datos a los registros debe hacerse a través de un solo registro llamado “de trabajo” (registro W). Así mismo todas las operaciones binarias que impliquen la intervención de la ALU tienen como uno de los operandos a W.

No obstante, en las gamas superiores esta dependencia del registro W es menor, ya que se dispone de juegos de instrucciones más amplios. Por ejemplo, se permite la asignación directa entre dos registros, sin tener que pasar el dato desde el origen a W y de éste al destino, como sucede en las gamas inferiores.

Los PIC, según la familia, disponen de varios puertos de entrada/salida digitales, conversores analógico digitales, interfaces compatibles con RS232, I^2C, SPI, CAN y USB.

La forma de designación de los PIC en general obedece a la siguiente estructura:

PIC nn LLL xxx

Siendo:

nn – un número propio de la gama del PIC.

LLL - código de letras donde la primera indica la tensión de alimentación y las otras dos el tipo de memoria que utiliza. En la tabla x.x se puede ver las distintas opciones que se pueden dar.

xxx – número que indica el modelo.

LETRAS	ALIMENTACIÓN	MEMORIA
C	Standard (4.5-6.0V)	EPROM
CR	Standard (4.5-6.0V)	ROM
F	Standard (4.5-6.0V)	FLASH
LC	Extendida (2.5-6.0V)	EPROM
LCR	Extendida (2.5-6.0V)	ROM
LF	Extendida (2.0-6.0V)	FLASH

Tabla 2.2: Nomenclatura de los PIC

2.5. MICROCONTROLADOR PIC16F877A

2.5.1. ELECCIÓN DEL MICROCONTROLADOR

Existe una gran diversidad de microcontroladores, como se ha podido comprobar con anterioridad. Dependiendo de la potencia y características que se necesiten, se pueden elegir microcontroladores de 4, 8, 16 ó 32 bits. Aunque las prestaciones de los microcontroladores de 16 y 32 bits son superiores a los de 4 y 8 bits, la realidad es que los microcontroladores de 8 bits dominan el

mercado y los de 4 bits se resisten a desaparecer. Eso es debido a que los microcontroladores de 4 y 8 bits son apropiados para la gran mayoría de las aplicaciones, por lo que no es necesario emplear microcontroladores más potentes y en consecuencia más caros.

También los modernos microcontroladores de 32 bits se van afianzando en el mercado, siendo las áreas de más interés el procesamiento de imágenes, las comunicaciones, las aplicaciones militares, los procesos industriales y el control de los dispositivos de almacenamiento masivo de datos.

A la hora de seleccionar el microcontrolador a utilizar en un diseño concreto se ha de tener en cuenta multitud de factores, como la documentación, herramientas de desarrollo disponibles y su precio, la cantidad de fabricantes que lo producen y por supuesto las características del microcontrolador (tipo de memoria de programa, número de temporizadores, interrupciones, etc.).

En el caso particular de este proyecto, la elección del microcontrolador vino influenciada por el uso del entorno de desarrollo EasyPIC4 que se suministra con el microcontrolador PIC16F877A. Por lo que el proceso de elección tuvo su curso en sentido inverso, es decir, en lugar de especificar las características que debía cumplir el microprocesador y en base a ello seleccionar el más adecuado, se tomó el PIC16F877A y se verificó si dicho microcontrolador cumplía los requisitos necesarios.

2.5.2. CARACTERÍSTICAS DEL MICROCONTROLADOR PIC16F877A

El PIC16F877A es un microcontrolador que pertenece a la familia de los PIC16F87XA de la casa Microchip. El elevado rendimiento de este microprocesador de diseño avanzado permite realizar una gran cantidad de funciones y prestaciones.

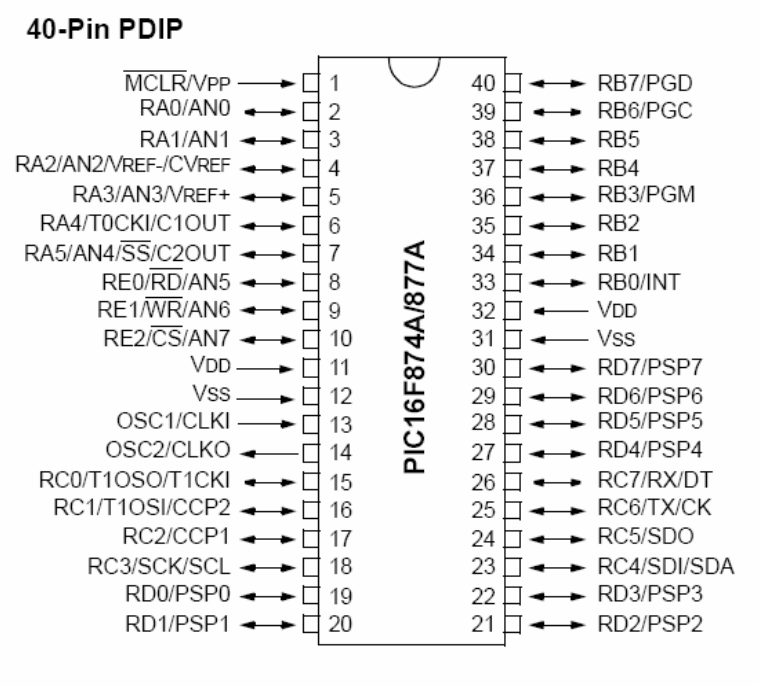


Figura 2.10: PIC16F874A/877A

A continuación se muestra las características generales del PIC16F877A:

Procesador de arquitectura RISC avanzada.

- Juego de sólo 35 instrucciones con 14 bits de longitud. Todas ellas se ejecutan en un ciclo de instrucción, menos las de salto que tardan dos.
- Hasta 8K palabras de 14 bits de memoria de programa tipo FLASH.
- Hasta 368 Bytes de memoria de datos RAM.
- Hasta 256 Bytes de memoria de datos EEPROM.
- Pines de salida compatibles con los microcontroladores PIC16CXXX y PIC16FXXX.

Recursos analógicos.

- Conversor Analógico/Digital de 10 bits.
- Reset de Brown-Out (BOR).

- Módulo de comparador analógico.

Recursos especiales.

- Código de protección programable.
- Modo SLEEP de bajo consumo.
- Perro Guardián (WDT).
- Programación serie en circuito con dos pines. Sólo necesita 5V para programarlo en este modo.

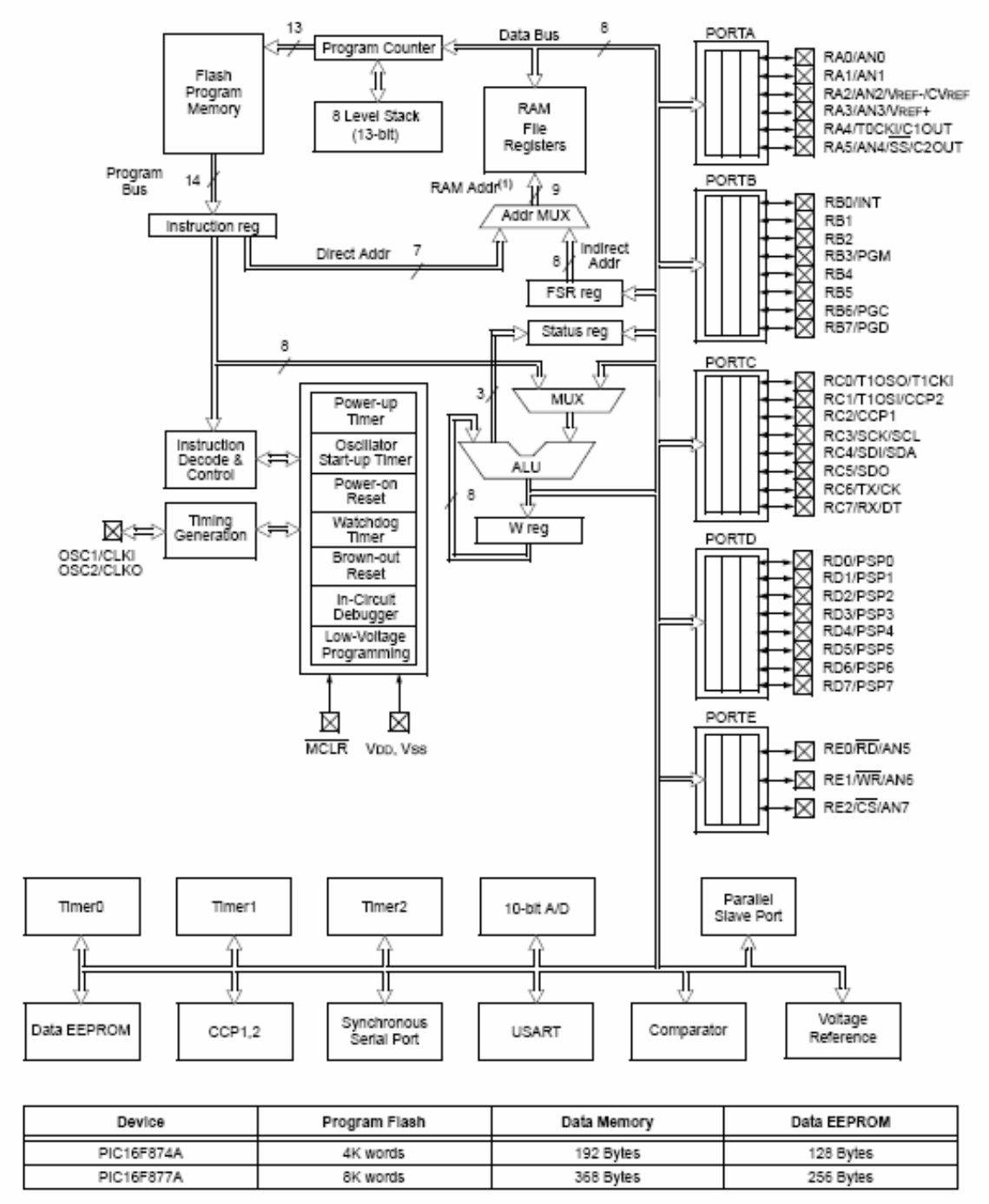
Recursos periféricos.

- Timer0: Temporizador-contador de 8 bits con prescaler de 8 bits.
- Timer1: Temporizador-contador de 16 bits con prescaler, puede incrementarse en modo sleep de forma externa por un cristal/clock.
- Timer2: Temporizador-contador de 8 bits con registro de periodo, prescaler y postescaler.
- Dos módulos de Captura, Comparación, PWM.
- Puerto Serie Síncrono (SSP) con SPI (Modo maestro) e I^2C (Master/Slave).
- USART/SCI (Universal Synchronous Asynchronous Receiver Transmitter) con 9 bits.
- Puerta Paralela Esclava (PSP) con control externo RD, WR y CS (sólo en encapsulados con 40 pines).

Tecnología CMOS.

- Voltaje de alimentación comprendido entre 2,0V y 5,5V.
- Bajo consumo.

Como se ha comentado anteriormente, el PIC16F877A se enmarca dentro de la familia de PIC16F87XA, cuya arquitectura se muestra en el siguiente diagrama:



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Figura 2.11: Arquitectura de la familia PIC16F87XA

Para más información sobre el microcontrolador se recomienda ver la hoja de características del componente disponible en el CD-ROM adjunto a la memoria

del presente Proyecto Fin de Carrera o bien en la dirección web del fabricante Microchip [5].

CAPÍTULO 3. ENTORNO DE DESARROLLO EASYPIC4

3.1. HARDWARE

3.1.1. INTRODUCCIÓN

El sistema de desarrollo EasyPIC4 de Mikroelektronika [11], consiste en un entrenador o placa didáctica de evaluación para aplicaciones basadas en los microcontroladores PIC de Microchip. Se ha diseñado para permitir a estudiantes e ingenieros explorar y trabajar con las capacidades de los microcontroladores PIC. Permite además, concentrarse principalmente en el desarrollo del software puesto que las conexiones entre microcontroladores PIC y circuitos externos son muy sencillas de realizar.

Dispone de una serie de periféricos básicos de E/S con los que se puede verificar el funcionamiento de una aplicación, así como los circuitos necesarios para la grabación de diversos modelos de microcontroladores PIC, en concreto, el sistema EasyPIC4 admite microcontroladores de 8, 14, 18, 28 y 40 pines.

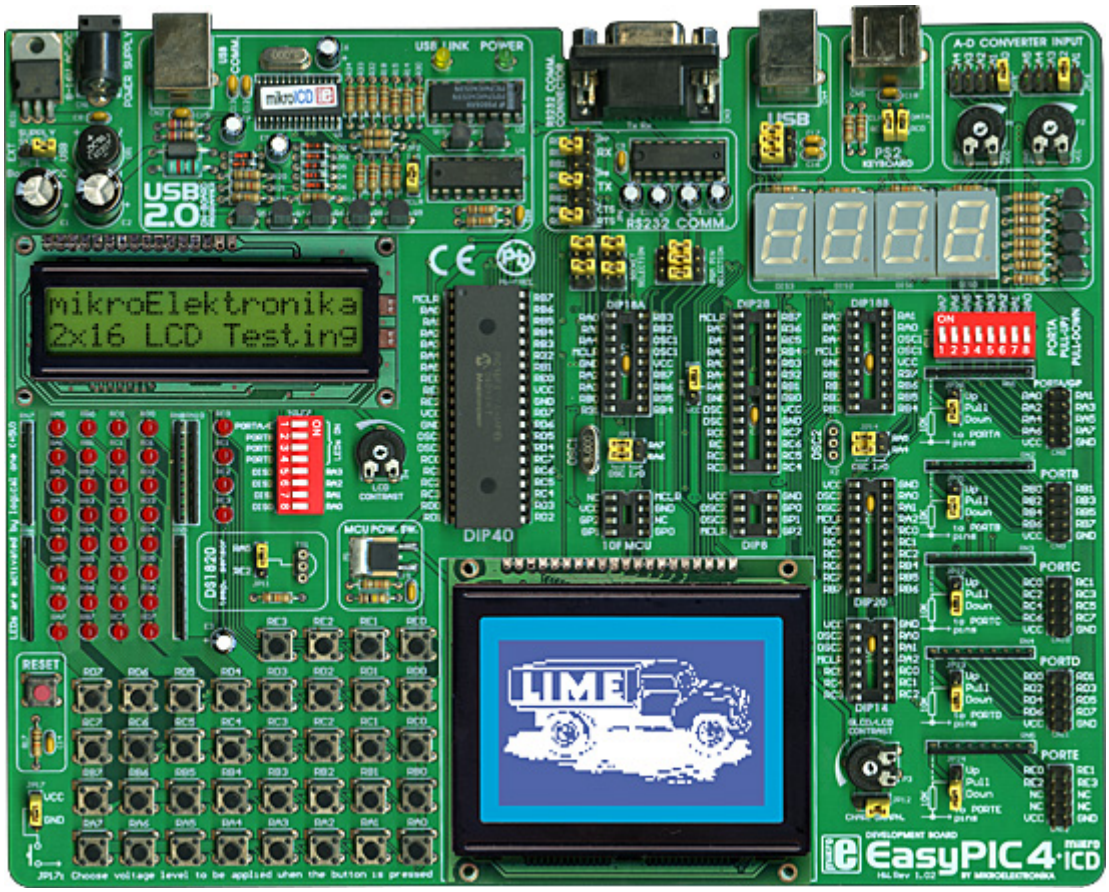


Figura 3.1: Sistema de desarrollo EasyPIC4

3.1.2. CARACTERÍSTICAS

El sistema de desarrollo EasyPIC4 se presenta totalmente montado, a excepción del LCD, GLCD y el sensor de temperatura, con un manual donde se incluye un tutorial con diversos ejemplos de demostración. También contiene un CD-ROM con las diferentes herramientas de diseño, así como los programas fuentes de los ejemplos propuestos en el manual.

A continuación se enumeran las principales características del sistema EasyPIC4:

1. Alimentación mediante fuente de alimentación externa AC/DC de 8V a 16V.
2. Selector de fuente de alimentación externa o vía USB.
3. Programador USB integrado con mikroICD (rápido y flexible).

4. Sensor de temperatura DS1820, permite medir temperatura con una precisión de 0.5 ºC.
5. Comunicación RS232 con selección TX y RX para microcontroladores pequeños.
6. Potenciómetros P1 y P2. Algunos pines están conectados a dichos potenciómetros pudiendo utilizarse para medir tensiones.
7. Mediante los interruptores SW1 se conecta el PORTA a una red de resistencias. Si un interruptor se encuentra en la posición OFF, el pin asociado no estará conectado con la resistencia pull-up o pull-down. Esto es importante ya que permite que el PORTA pueda ser utilizado en modo analógico como un conversor A/D o como un puerto digital de E/S.
8. Jumpers. Al situar el jumper en la posición de arriba (pull-up) aparece un uno digital en el puerto correspondiente. Si el jumper se encuentra en la posición de abajo (pull-down), los pines reciben un cero lógico (pull-down).
9. Zócalo para módulo LCD en modo 4 bits.
10. Zócalo para módulo LCD gráfico o LCD en modo de 8 bits.
11. Zócalos para situar los microcontroladores en DIP8, DIP14, DIP18, DIP20, DIP28 y DIP40.
12. 36 botones para controlar todos los pines del microcontrolador.
13. Jumper para seleccionar el tipo de pulsación (activo a nivel bajo o a nivel alto).
14. LEDs, cada pin del microcontrolador tiene asociado un LED.
15. Displays de 7 segmentos en modo multiplexor.
16. Interruptores que encienden o apagan los LEDs de los puertos PORTA, PORTB, PORTC, PORTD y PORTE.
17. Para seleccionar el contraste del LCD.
18. Control de fuente de alimentación.

19. Módulo de comunicación USB.

20. Conector para el teclado.

21. Circuito de reset.

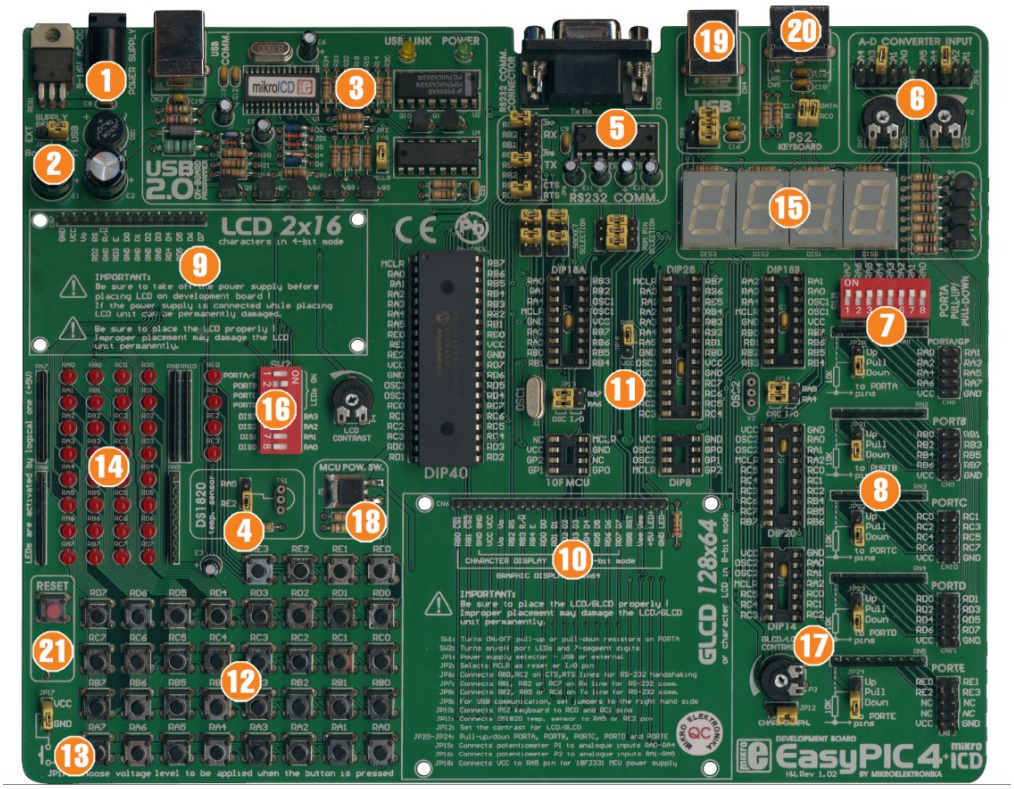


Figura 3.2: Componentes del sistema de desarrollo EasyPIC4

3.1.3. ARQUITECTURA DEL ENTRENADOR EASYPIC4

En este apartado se realiza una explicación exhaustiva del sistema de desarrollo EasyPIC4.

- Interruptores

Los interruptores son dispositivos que tienen dos posiciones, ON y OFF, los cuales tienen la función de habilitar o deshabilitar la conexión entre dos contactos. La placa EasyPIC4 contiene dos grupos de interruptores:

El primer grupo, SW1, realiza la conexión entre el puerto del microcontrolador con capacidad analógica (PORTA) y las resistencias externas Pull-Up/Down. Las resistencias Pull-Up/Down deben estar desconectadas al utilizar los pines como entradas/salidas analógicas, ya que de no ser así, afectarán al nivel de tensión de entrada. Cuando los pines de PORTA se utilizan como entradas/salidas digitales, las resistencias Pull-Up/Down deben ser habilitadas.

Los cuatro interruptores de SW2 superiores son utilizados para habilitar los LEDs conectados a PORTA/E, PORTB, PORTC y PORTD. Por ejemplo, si el interruptor de PORTB está desconectado, todos los LEDs de PORTB estarán apagados.

Mientras que los cuatro interruptores inferiores de SW2 se utilizan para habilitar los displays de 7 segmentos.

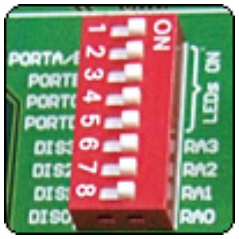


Figura 3.3: Grupo de interruptores SW2

- Jumpers

Los jumpers, al igual que los interruptores, realizan la conexión entre dos puntos. Debajo de la cubierta de plástico del jumper existe un contacto metálico que establece la conexión al situar el jumper entre dos pines desconectados.

Por ejemplo, el grupo de jumpers JP10 tiene dos jumpers que realizan la función de un interruptor. Se utilizan para conectar o desconectar los pines PS/2 CLK y PS/2 DATA a los pines RC1 y RC0 respectivamente, del microcontrolador. La conexión se realiza cuando el jumper se sitúa entre ambos contactos.

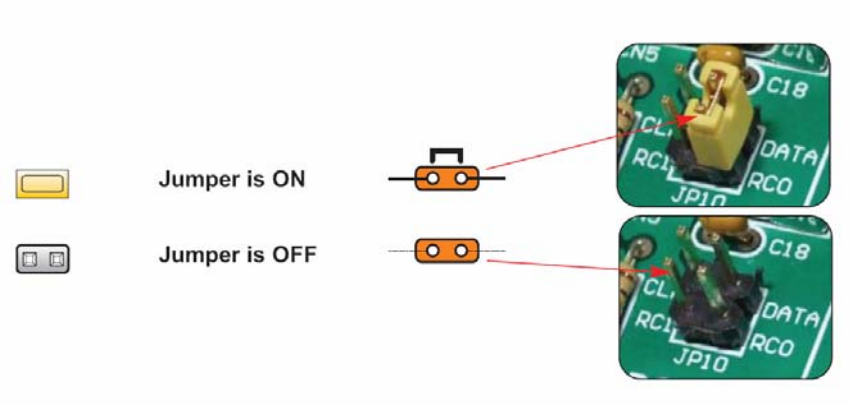


Figura 3.4: Jumper en modo interruptor

A menudo, los jumpers se utilizan como selectores entre dos posibles conexiones utilizando conectores de tres pines. Como se muestra en la Figura 3.5, el contacto del centro puede conectarse al pin de la derecha o de la izquierda dependiendo de la posición del jumper.

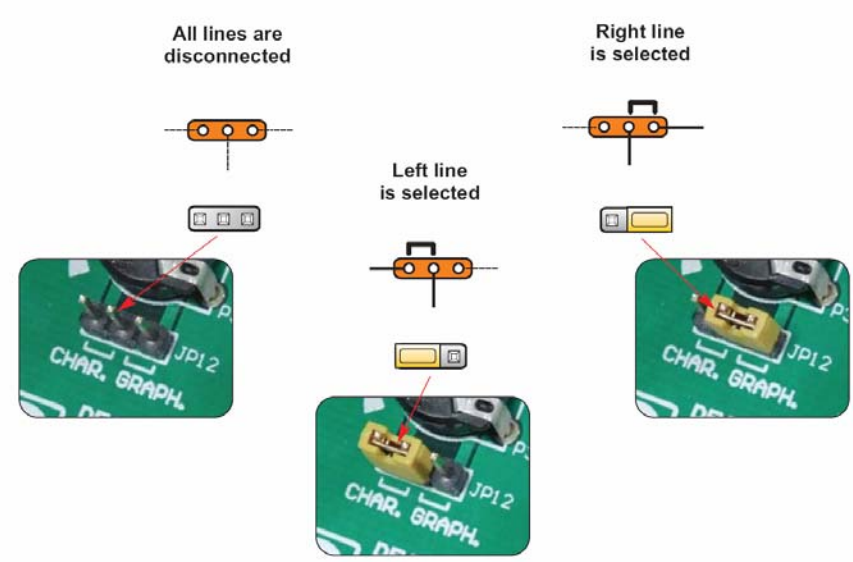


Figura 3.5: Jumper en modo multiplexor

- Zócalos MCU

Es la parte más importante del sistema de desarrollo EasyPIC4 y donde se inserta el dispositivo PIC con el que se va a trabajar.

El sistema de desarrollo EasyPIC4 viene con un microcontrolador de 40 pines, pero se puede utilizar otro microcontrolador distinto quitando dicho

microcontrolador y situando el deseado en el zócalo adecuado: DIP40, DIP28, DIP20, DIP18, DIP14 o DIP8.

Cuando se realiza la conexión de un microcontrolador en DIP18, se ha de tener en cuenta que existen dos zócalos DIP18 con diferentes pines (DIP18A y DIP18B). Por ejemplo, el PIC16F628A utiliza el zócalo DIP18A, mientras que el zócalo DIP18B se utiliza con el microcontrolador PIC18F1220. El zócalo 10F MCU se utiliza únicamente para la familia del PIC10F y el zócalo DIP8 para el resto de microcontroladores de 8 pines.

Aunque las conexiones están realizadas de forma paralela, no puede haber más de un microcontrolador en la placa al mismo tiempo.

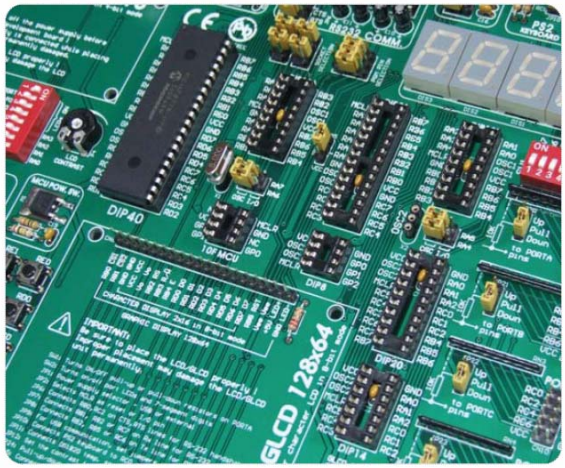


Figura 3.6: Zócalos MCU

Los pines del microcontrolador se encuentran unidos a varios dispositivos periféricos. Todos los puertos están conectados con los LEDs, pulsadores y resistencias Pull-Up/Down, permitiendo monitorizar y testear el estado digital de los pines. Pero además de estas conexiones, también presenta una conexión directa con los conectores de acceso directo al puerto. Dichos conectores se suelen utilizar para conectar periféricos externos a la placa o para proporcionar puntos útiles donde conectar una sonda digital lógica.

Algunos pines están conectados a otros periféricos como por ejemplo, al sensor de temperatura DS1820, comunicación RS-232, displays de 7 segmentos, LCD, etc.

- Fuente de alimentación

Como fuente de alimentación, se puede seleccionar entre el suministro regulado desde el cable de USB, y una fuente de alimentación externa. En caso de la fuente de alimentación mediante USB, el sistema debe ser conectado a un ordenador utilizando el cable USB de programación, mientras el jumper JP1 se encuentra en la posición derecha.

En el caso de fuente de alimentación externa, la placa EasyPIC4 produce una tensión de 5V utilizando un regulador de tensión LM7805. La fuente de energía externa puede ser AC o DC, con una tensión entre 8V y 16V, para este tipo de alimentación el jumper JP1 ha de situarse en la posición izquierda.

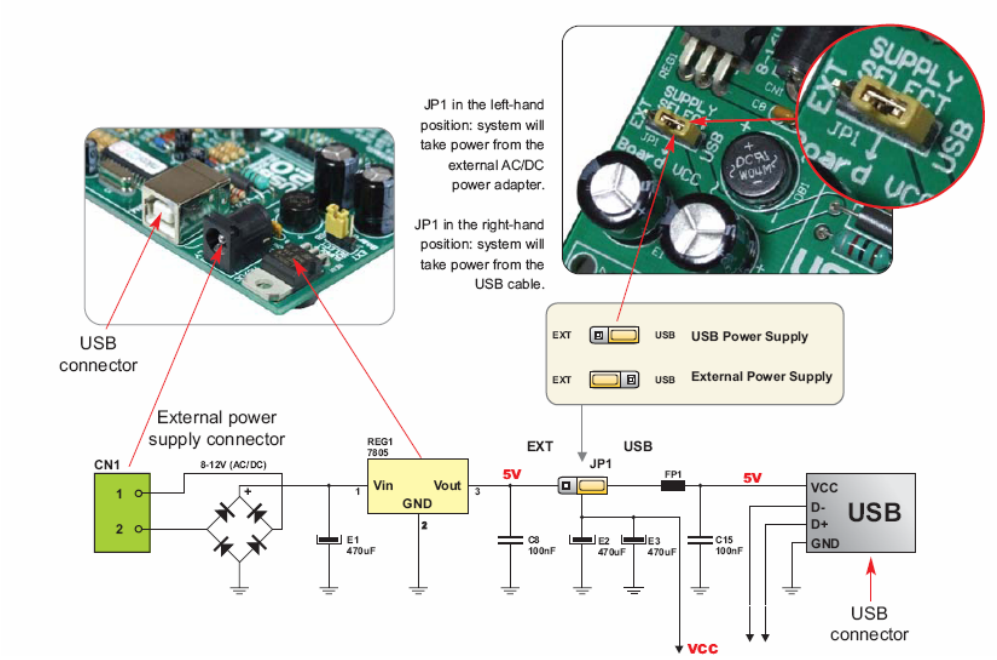


Figura 3.7: Conectores para el suministro mediante USB y fuente de alimentación externa

- Programador USB 2.0 integrado

No es necesario el uso de un equipo externo durante el proceso de programación, ya que el sistema de desarrollo EasyPIC4 tiene su propio programador USB integrado en la placa. Todo lo que se ha de realizar es conectar el cable USB al ordenador y cargar el programa en el PIC mediante el software PICFlash.

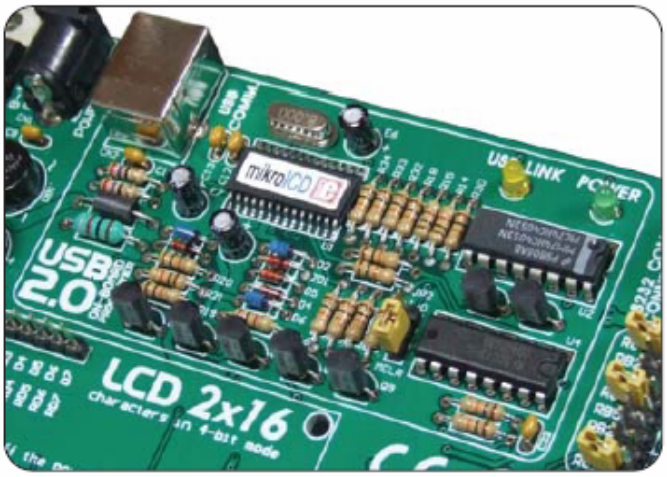


Figura 3.8: Programador USB 2.0

- Oscilador

El oscilador se encarga de generar la frecuencia principal de trabajo del microcontrolador.

Puesto que existen muchos zócalos en la placa EasyPIC4, existen dos osciladores que están conectados con dos secciones de zócalos MCU. El primer oscilador llamado OSC1, se encuentra conectado a los zócalos DIP40, DIP28, DIP18A y DIP18B. El segundo oscilador, OSC2, está conectado a DIP20, DIP14 y DIP18.

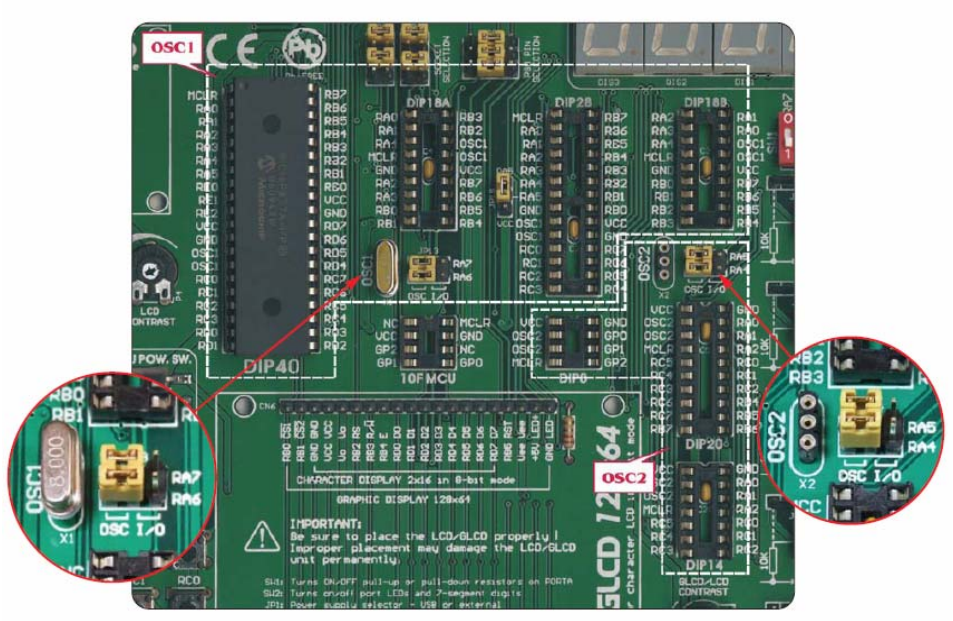


Figura 3.9: Osciladores

Como se puede apreciar en la Figura 3.9, el zócalo 10F no se encuentra conectado a ninguno de los dos osciladores, esto se debe que estos microcontroladores tienen un oscilador interno y no pueden ser utilizados con cristales externos.

En algunos microcontroladores los pines de entrada del oscilador pueden ser utilizados como pines de entrada/salida. Para implementar esta característica, la placa EasyPIC4 tiene jumpers para conectar el MCU tanto al oscilador como a los pines de E/S digitales.

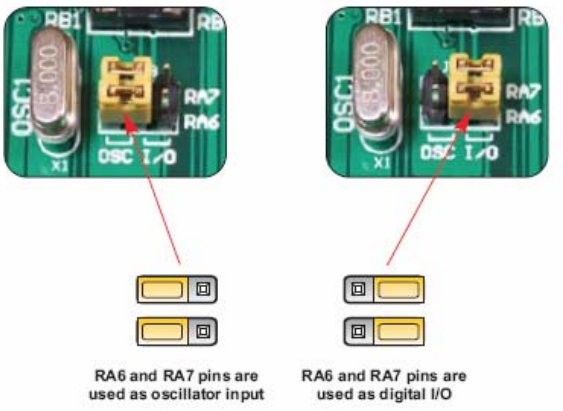


Figura 3.10: Conexión entre el oscilador y el MCU

- mikroICD

MikroICD es una herramienta sumamente eficaz para eliminar fallos en tiempo real a nivel de hardware. El depurador de mickroICD permite ejecutar un programa en un microcontrolador y ver el valor de las variables, Registros de Función Especiales (SFR) y la memoria EEPROM mientras el programa está siendo ejecutado.

El depurador mikroICD utiliza el programador que hay en la placa para comunicarse con el compilador y proporcionarle los comandos típicos para depurar:

Start Debugger	[F9]
Run/Pause Debugger	[F6]
Toggle Breakpoints	[F5]

Run to cursor	[F4]
Step Into	[F7]
Step Over	[F8]
Flush RAM	[F2]
Stop Debugger	[Ctrl+F2]

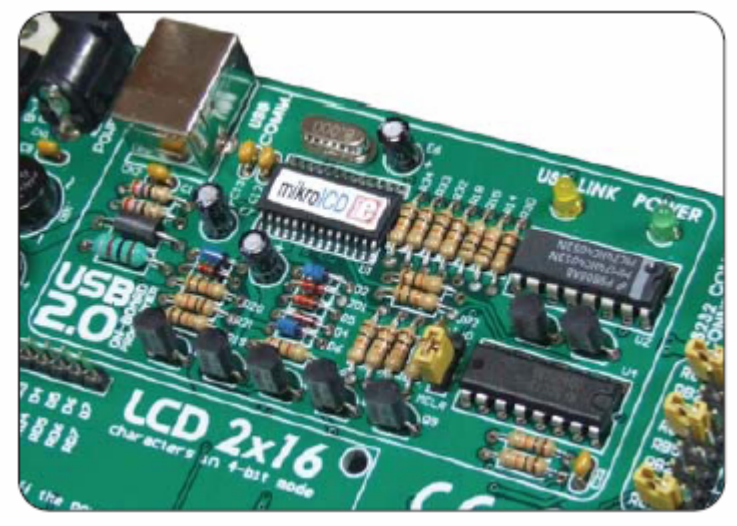


Figura 3.11: Programador USB integrado con mikroICD

- LEDs

Los diodos emisores de luz (LED) son los componentes más utilizados, principalmente para mostrar el nivel lógico de los pines a los que están conectados. El entrenador EasyPIC4 tiene 36 LEDs que están conectados a PORTA, PORTB, PORTC, PORTD y PORTE del microcontrolador.

Cada grupo de ocho LEDs puede ser habilitado o deshabilitado mediante el interruptor SW2, excepto PORTE, el cual a diferencia de los otros puertos, sólo tiene 4 LEDs y está conectado al mismo interruptor que PORTA.

Los LEDs están habilitados cuando los correspondientes interruptores de SW2 están en ON. Cuando estén habilitados, los LEDs mostrarán el estado correspondiente al pin del microcontrolador, de no ser así los LEDs estarán siempre apagados independientemente de cómo se encuentre el puerto, como si no existiese corriente a través de los LEDs.

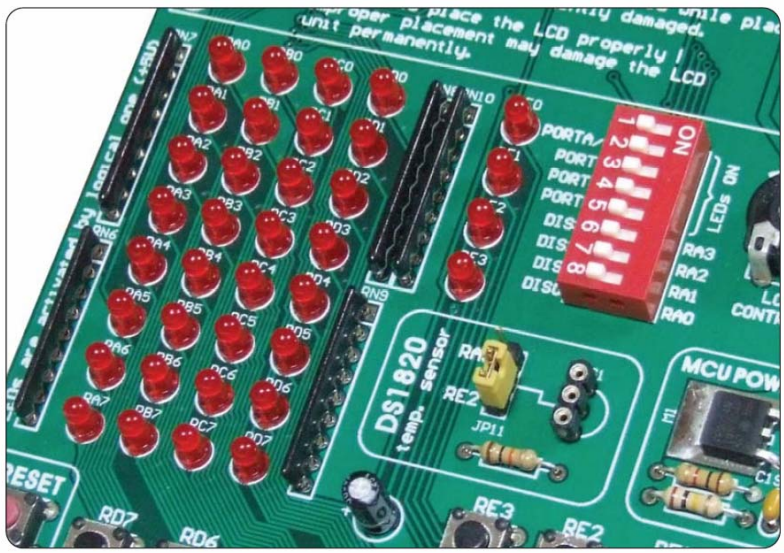


Figura 3.12: LEDs

- Pulsadores

La placa EasyPIC4 tiene 36 pulsadores, con los que se puede cambiar los estados de las entradas digitales del microcontrolador. También presenta un pulsador, RESET, que cuando es pulsado el programa del microcontrolador empieza a ejecutarse desde el principio.

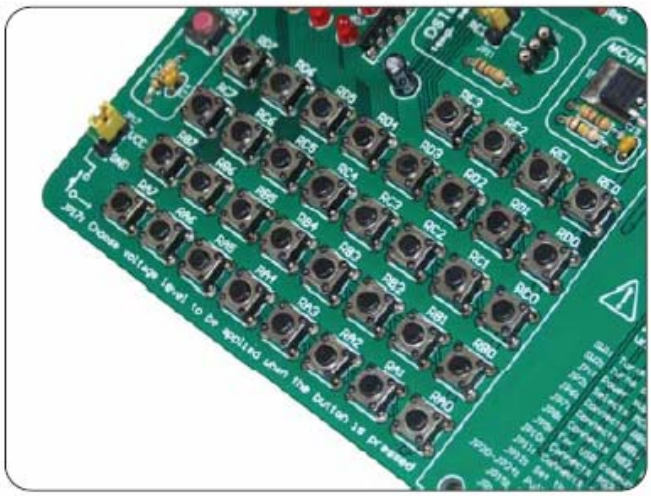


Figura 3.13: Pulsadores

El jumper JP17 determina si el botón pulsado aporta al respectivo pin un cero o un uno lógico.

Como se puede apreciar en la siguiente figura, el jumper JP21 se encuentra en la posición de Pull-Up, aunque el botón no esté apretado, la resistencia Pull-Up suministrará 5V al pin RB4 del microcontrolador.

Así, solamente cuando el botón está apretado el microcontrolador recibirá un cero lógico, si no el estado del pin será siempre un uno lógico.

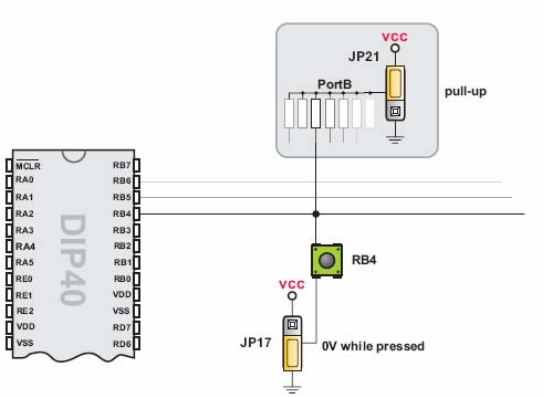


Figura 3.14: Botón con resistencia Pull-Up

Si el jumper JP21 pasa a la posición pull-down, aunque el botón no se encuentre apretado, la resistencia pull-down suministrará 0V al pin RB4 del microcontrolador.

Así, solamente cuando el botón se encuentra apretado el microcontrolador recibirá un uno lógico, si no el estado del pin será siempre un cero lógico.

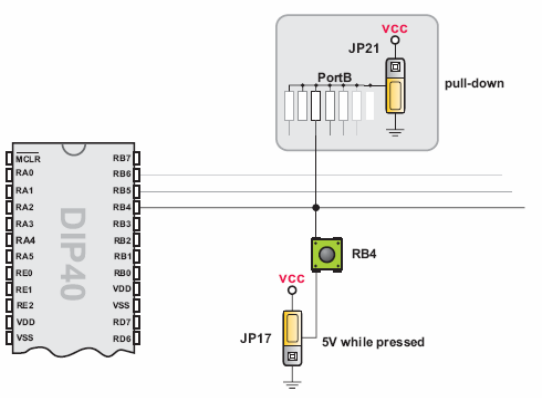


Figura 3.15: Botón con resistencia Pull-Down

- Displays de 7 segmentos

Un display de 7 segmentos está formado por siete LEDs rectangulares colocados en forma de ocho tal y como se puede apreciar en la Figura 3.16. Dependiendo del tipo de display que se utilice, la manera en que se enciendan sus segmentos será distinta. Un display de cátodo común requiere un 1 lógico para encender los segmentos, mientras que uno de ánodo común un 0.

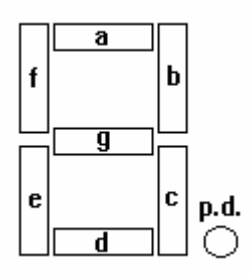
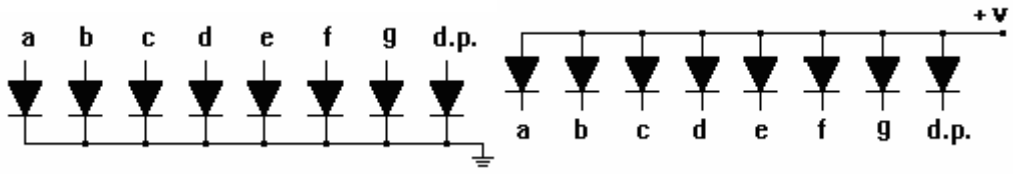


Figura 3.16: Segmentos que componen el display de 7 Segmentos

La placa EasyPIC4 tiene cuatro displays de 7 segmentos de cátodo común en modo multiplexado.



(a) **(b)**

Figura 3.17: Estructura de cátodo común (a) y ánodo común (b)

Las líneas de datos están conectadas a PORTD, mientras que el display se habilita a través de los cuatro bits menos significativos de PORTA.

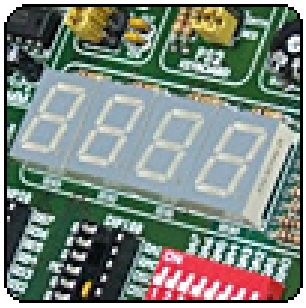


Figura 3.18: Displays de 7 segmentos

- Pantalla LCD

El componente más ampliamente utilizado para la visualización de datos es el display LCD. Dicho dispositivo puede mostrar dos líneas de 16 caracteres alfanuméricos de 5x8 píxeles.

Este potente periférico de salida permite representar cualquier tipo de mensaje compuesto de letras, números y símbolos, produciendo además diferentes efectos de visualización como desplazamientos a izquierda y derecha, parpadeos, *scrolls*, etc.

Las características generales de un módulo LCD 16x2 son las siguientes:

- Consumo muy reducido, del orden de 7.5 mW.
- Pantalla de caracteres ASCII, además de los caracteres japoneses Kanji, caracteres griegos y símbolos matemáticos.
- Desplazamiento de los caracteres hacia la izquierda o a la derecha.
- Memoria de 40 caracteres por línea de pantalla, visualizándose 16 caracteres por línea.
- Movimiento del cursor y cambio de su aspecto.
- Permite que el usuario pueda programar 8 caracteres.
- Pueden ser gobernados de 2 formas principales:
    - Conexión con bus de 4 bits.
    - Conexión con bus de 8 bits.

El módulo LCD posee una zona de memoria RAM llamada DDRAM (Data Display RAM) donde se almacenan los caracteres que se van a mostrar en la pantalla.

Tiene una capacidad de 80 bytes, 40 por cada línea, de los cuales sólo 32 se pueden visualizar a la vez (16 bytes por línea).

De las 80 posibles, las dos direcciones más importantes de la DDRAM son:

- Dirección 00h, que es el comienzo de la primera línea.
- Dirección 40h, que el comienzo de la segunda línea.

Además de la DDRAM, el LCD dispone de una zona de memoria interna, no volátil, llamada CGROM donde se almacena una tabla con los 192 caracteres que pueden ser visualizados.

Cada uno de los caracteres tiene su representación binaria de 8 bits. Para visualizar un caracter, debe recibir por el bus de datos el código correspondiente.

Tabla 3.1: Juego de caracteres del módulo LCD

Además de los caracteres mostrados en la Tabla 3.1, también permite definir 8 nuevos caracteres de usuario que se guardan en una zona de RAM denominada CGRAM (Character Generator RAM).

El LCD se comunica con el microcontrolador mediante un bus de datos de 4 u 8 bits, dependiendo del lugar donde se conecte el LCD en el entorno de desarrollo EasyPIC4.

- LCD 2x16 en modo de 4 bits

Para utilizar el bus de datos de 4 bits, el LCD se ha de colocar en la parte superior izquierda de la placa, justo encima de los LEDs. En dicha conexión, sólo se conecta cuatro líneas de datos. Es importante tener en cuenta que para poner o quitar el LCD de la tarjeta EasyPIC4, la placa tiene que estar desconectada.

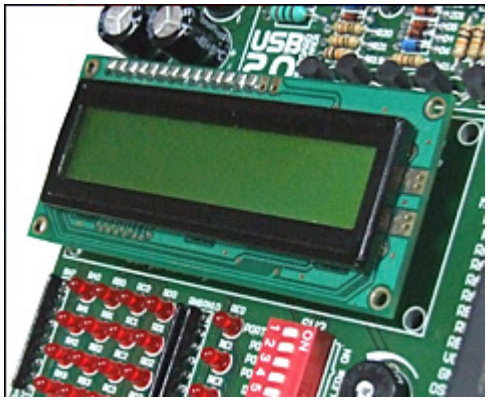


Figura 3.19: LCD 2x16 en modo de 4 bits

- LCD 2x16 en modo de 8 bits

Cuando se utiliza el LCD en modo de 8 bits, se utiliza el mismo conector que en la conexión del GLCD. Como el conector de la placa tiene 20 pines y el LCD sólo 14, se ha tener cuidado al colocar el LCD, ya que si se sitúa de manera incorrecta el LCD quedará dañado.

El LCD ha de situarse en la posición que da lugar de dejar dos pines libres en la izquierda y cuatro libres en la derecha. Antes de conectar el LCD, se ha de poner el jumper JP12 en la posición izquierda. El contraste de este dispositivo

se puede ajustar utilizando el potenciómetro P3, que se encuentra a la derecha del conector GLCD/LCD.

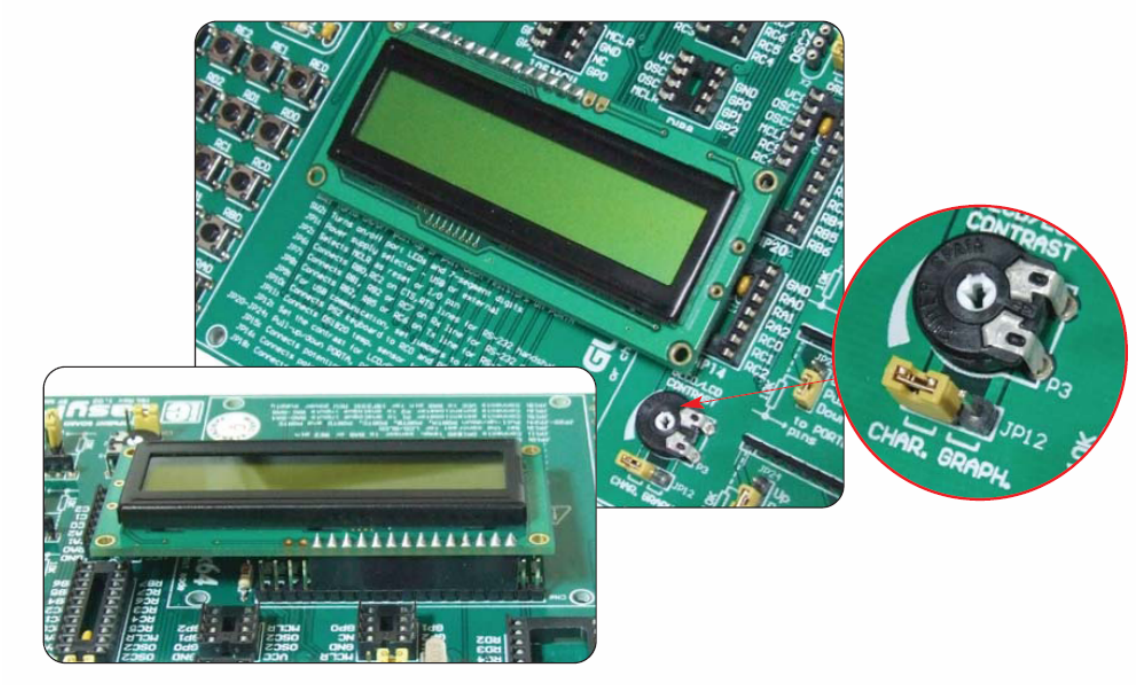


Figura 3.20: LCD 2x16 en modo de 8 bits

- LCD gráfico

Mientras que el LCD sólo puede mostrar caracteres alfanuméricos, el GLCD puede utilizarse para visualizar mensajes en formato de dibujo o mapa de bits, es decir, mensajes visuales. El GLCD más utilizado tiene una resolución de 128x64 píxeles.

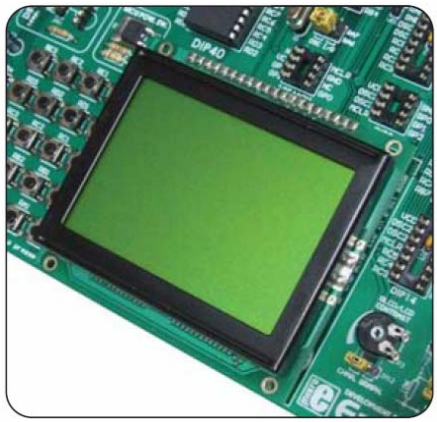


Figura 3.21: GLCD

Antes de conectar el GLCD, se ha de situar el jumper JP12 en la posición derecha y utilizando el potenciómetro P3, que se encuentra a la derecha del GLCD, se puede ajustar su contraste.

- Comunicación RS-232

La comunicación RS-232 permite transferir datos punto a punto. Se suele utilizar en aplicaciones de adquisición de datos para la transferencia de datos entre el microcontrolador y el ordenador. Puesto que los niveles de tensión de un microcontrolador y un ordenador no son compatibles con los del RS-232, se utiliza el adaptador de niveles MAX232.

Para dotar de mayor flexibilidad al sistema, el microcontrolador está conectado al MAX232 mediante dos grupos de jumpers: JP7 y JP8. El jumper JP7 se utiliza para conectar la línea Rx (recepción) a los pines RC7, RB2 o RB1. El jumper JP8 conecta la línea Tx (transmisión) a los pines RC6, RB5 o RB2.

Esta interfaz permite realizar todo tipo de comunicaciones serie entre la EasyPIC4 y cualquier otro equipo mediante el protocolo estándar RS-232. La velocidad de transferencia irá en función del tipo de microcontrolador empleado y su velocidad de trabajo.

- Comunicación USB

El conector USB de comunicación se encuentra en la esquina superior derecha de la placa EasyPIC4. Se utiliza con microcontroladores PIC específicos que soportan la interfaz USB, tales como PIC18F2450 o PIC18F4550. Se ha de tener en cuenta que el conector USB de comunicación no se puede utilizar para programar y que el conector USB de programación no puede ser utilizado para comunicarse. Para realizar la conexión entre el microcontrolador y el conector USB de comunicación, los jumpers de JP9 han de estar en la posición derecha. De esta forma, los pines RC3, RC4 y RC5 del microcontrolador se desconectan del resto del sistema y se conectan al conector USB de comunicación.

- Comunicación PS/2

Los conectores PS/2 permiten una conexión directa entre la EasyPIC4 y los dispositivos que utilizan una comunicación PS/2, tales como el ordenador, el teclado o el ratón. Por ejemplo, el microcontrolador se puede conectar al teclado para capturar las teclas pulsadas o al ordenador actuando como teclado.

Las líneas CLK y DATA se utilizan para transferir datos. En este caso, se conectan a los pines RC1 y RC0 respectivamente.

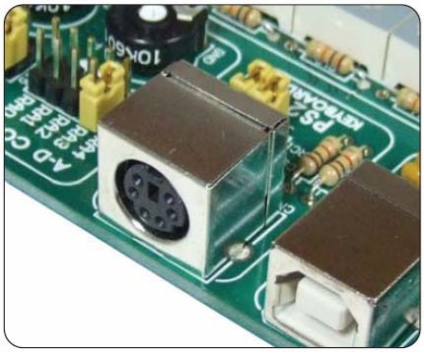


Figura 3.22: Conector PS/2

- Termómetro digital DS1820

El termómetro digital DS1820 es muy práctico para medir la temperatura ambiente, teniendo un rango de –55 ºC a 125 ºC y una precisión de +/-0.5 ºC. Dicho termómetro se puede conectar tanto al pin RA5 como al RE2 mediante el jumper JP11.

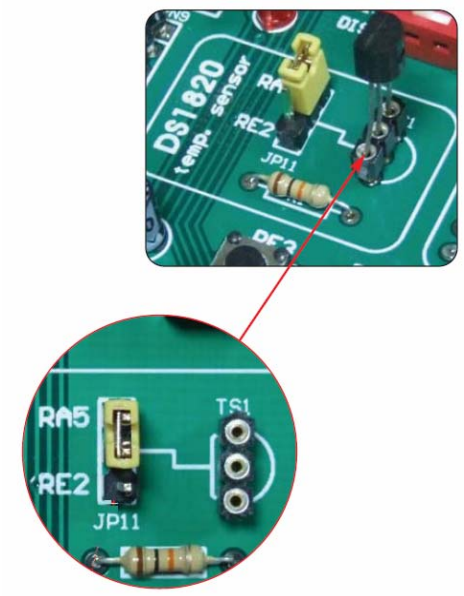


Figura 3.23: Termómetro DS1820

- Conversor A/D

El sistema de desarrollo EasyPIC4 tiene dos potenciómetros para trabajar con el conversor analógico/digital. Las salidas de ambos potenciómetros se encuentran en el rango de 0V a 5V. Dos señales analógicas pueden ser conectadas a dos pines de entrada analógicos diferentes al mismo tiempo. El grupo de jumpers JP15 habilita la conexión entre el potenciómetro P1 y uno de los siguientes pines: RA0, RA1, RA2, RA3 o RA4. Mientras que el grupo de jumpers JP16 permite la conexión entre el potenciómetro P2 y RA1, RA2, RA3, RA4 o RA5.

Para medir la señal analógica sin interferencia, se ha de poner el interruptor correspondiente de SW1 en la posición OFF. Esto deshabilitará la conexión de los pines de PORTA con las resistencias Pull-Up/Down.

Existen diversas aplicaciones del conversor A/D. El microcontrolador obtiene la señal analógica de sus pines de entrada y la convierte en un valor digital. Básicamente, se puede medir cualquier señal analógica comprendida ente 0V y 5V.

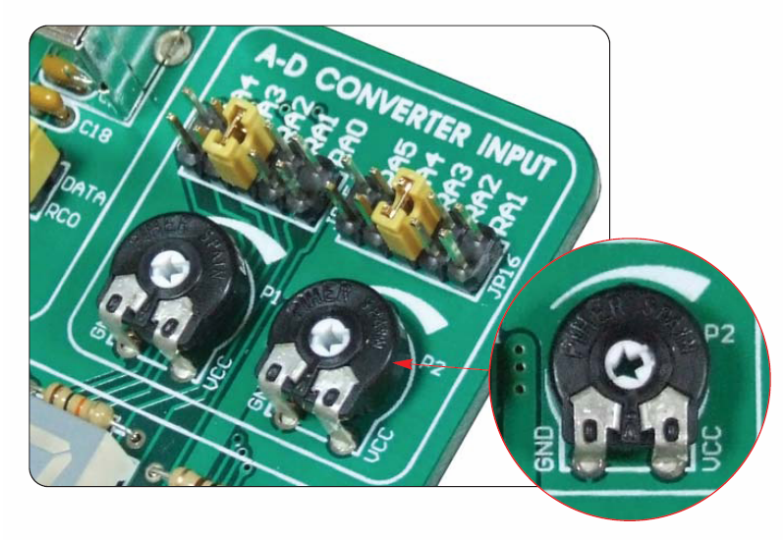


Figura 3.24: Conversor A-D

- Acceso directo a los puertos

Se puede acceder a todos los pines de entrada/salida del microcontrolador mediante los conectores que se encuentran a lo largo del lado derecho de la placa. Para cada puerto PORTA, PORTB, PORTC, PORTD y PORTE existe un conector de 10 pines que suministra VCC, GND y hasta ocho líneas de los puertos.

Estos conectores se pueden utilizar para ampliar el sistema con dispositivos externos tales como Serial Ethernet, Compact Flash, MMC/SD, ADC, CAN, RTC, RS-485, etc.

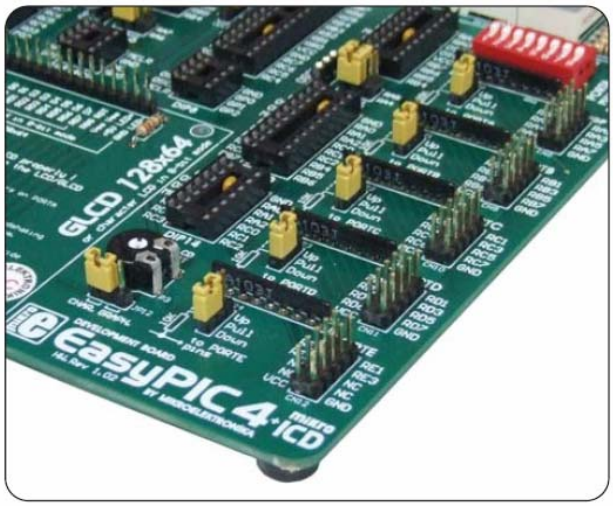


Figura 3.25: Conectores de acceso directo a los puertos

3.2. SOFTWARE DE GRABACIÓN

3.2.1. INTRODUCCIÓN

Los microcontroladores en general, y los de la empresa Microchip en particular, necesitan de un circuito electrónico que permita transferirles el programa realizado desde el ordenador. Existen muchas maneras de encarar este problema, y en general se suele utilizar alguno de los puertos disponibles en cualquier ordenador para este fin. Por ello, en el mercado es posible conseguir programadores de PICs con conexión para puerto USB, paralelo o serie (RS-232).

3.2.2. SOFTWARE DE GRABACIÓN

Como se ha comentado anteriormente, el sistema de desarrollo EasyPIC4 tiene su propio programador USB integrado en la placa, y por tanto, no es necesario el uso de un equipo externo durante el proceso de programación.

El circuito grabador del sistema de evaluación EasyPIC4 se conecta con un ordenador personal mediante cable USB o puerto serie. En el presente trabajo se ha utilizado la conexión vía USB.

Con el fin de que el ordenador realice las diferentes opciones de grabación, se ha utilizado un software que gestione dicha conexión. Dicho software, que recibe el nombre de PICFlash, es un software en inglés y está diseñado para trabajar bajo el sistema operativo Windows en un entorno gráfico con ventanas y botones.

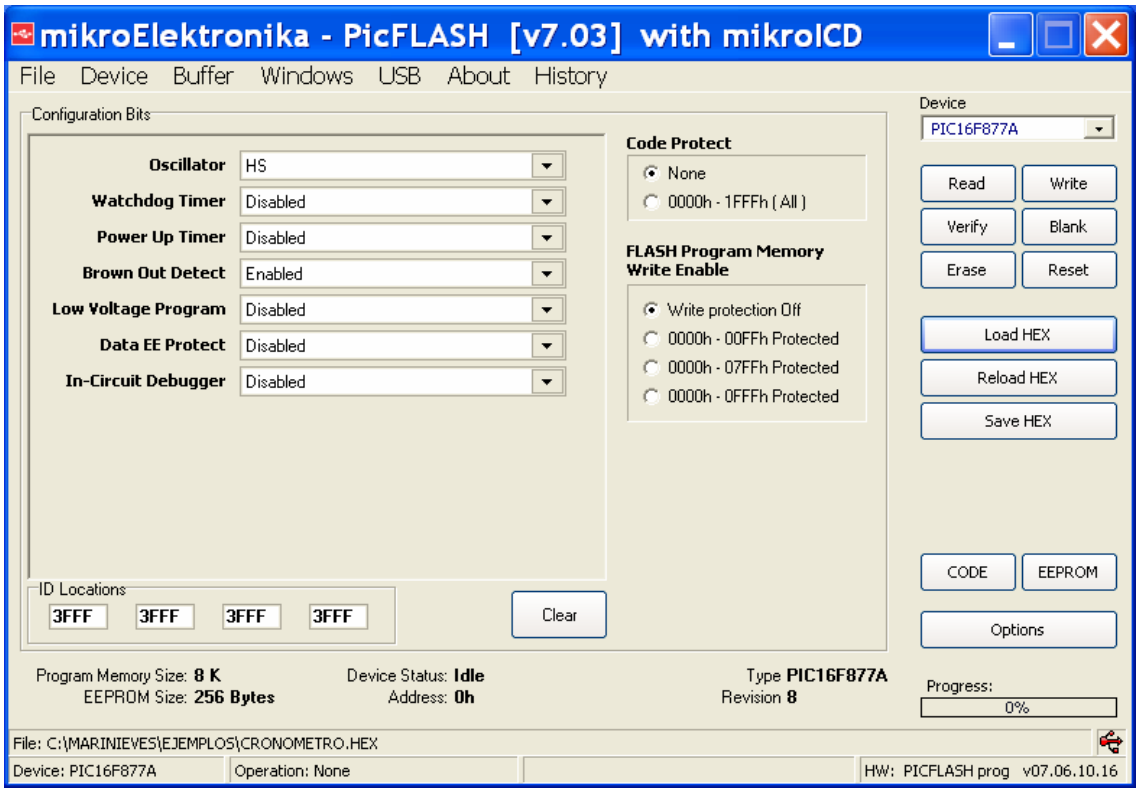


Figura 3.26: Aspecto de la pantalla de programa PICFlash

3.2.3. PROGRAMADOR PICFLASH

Junto con software adicional, el programador PICFlash representa una gran herramienta para el trabajo con los microcontroladores PIC.

El microcontrolador es conectado al programador de PICFlash vía 5 líneas, dos de las cuales son +5V y GND y las otras líneas son PGC, PGD y MCLR. La posición de estos pines varía dependiendo del tipo de microcontrolador. A diferencia de los programadores cuyo funcionamiento se basa en *bootloads* (que necesitan ceder parte de su memoria a un programa de *bootload*), PICFlash programa el microcontrolador externamente para que toda la memoria esté disponible para programación.

Los microcontroladores PIC pueden programarse con 5V y 12.5V. Cuando se programan con 5V, el pin RB3 no se utiliza para nada más que para la programación, estando no disponible para la aplicación. PICFlash programa con 12.5V generados a través del suministro de tensión de alimentación +5V

dejando así el pin libre. La conexión al ordenador se realiza vía USB, para que el programador PICFlash pueda trabajar en todos los ordenadores modernos.

CAPÍTULO 4. SOFTWARE UTILIZADO

4.1. MPLAB IDE

4.1.1. INTRODUCCIÓN

El proceso de escritura de una aplicación se describe a menudo como un ciclo de desarrollo, ya que es muy difícil que todos los pasos efectuados desde el diseño hasta la implementación se realicen correctamente a la primera. La mayoría de las veces se escribe el código, se testea y luego se modifica para crear una aplicación que funcione correctamente.

MPLAB IDE integra todas estas funciones con el fin de no tener que utilizar distintas herramientas y diferentes modos de operación.

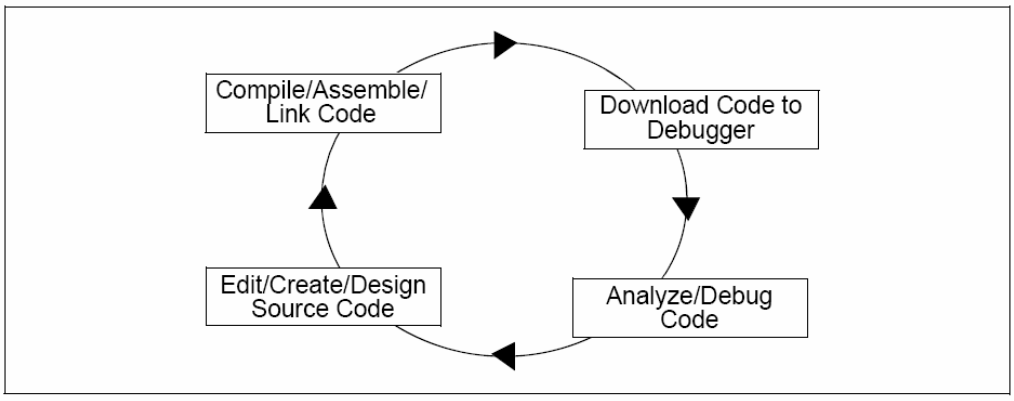


Figura 4.1: Proceso de escritura de una aplicación

El software MPLAB IDE de Microchip [5], es un entorno de desarrollo integrado bajo Windows, que permite editar, ensamblar, linkar, depurar y simular proyectos para los distintos dispositivos PIC de Microchip.

Dicho entorno incorpora todas las herramientas necesarias para la realización de cualquier proyecto:

- Editor de texto.
- Ensamblador.
- Linkador.
- Simulador.
- Menús de ayuda.

Además de las herramientas que incorpora, se pueden añadir otras como por ejemplo:

- Compiladores de C.
- Emuladores.
- Programadores.
- Depuradores.

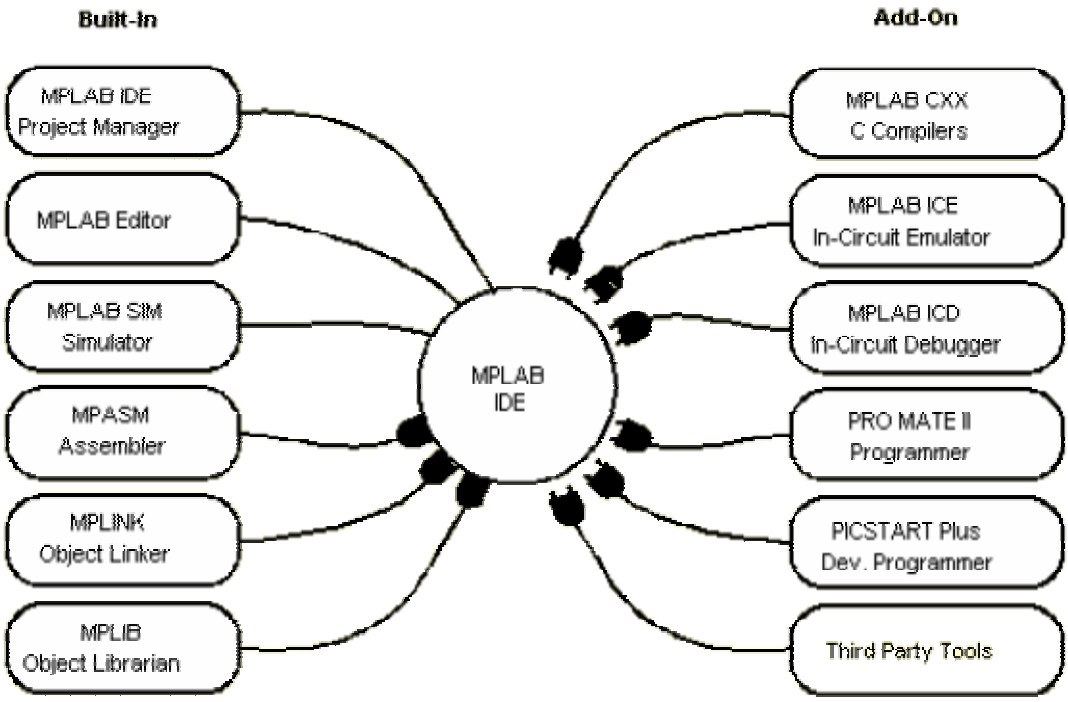


Figura 4.2: Componentes integrados y adicionales.

El entorno MPLAB es un software que junto con un emulador y un programador, forman un conjunto de herramientas de desarrollo muy completo para el trabajo o diseño con microcontroladores PIC.

4.1.2. COMPONENTES DE MPLAB IDE

4.1.2.1. Componentes integrados

A continuación se describen los componentes integrados en el software MPLAB IDE:

- Project Manager.

El componente Project Manager permite integrar y comunicar el MPLAB IDE con las herramientas de lenguaje. Organiza los distintos archivos relacionados con un programa en un proyecto. Permite crear un proyecto, editar y simular un programa. Además crea archivos objetos y permite bajar archivos hacia emuladores (MPLAB-ICE) o simuladores de hardware (SIMICE).

- Editor

El editor es un editor de texto completo que sirve, además, como ventana en el depurador.

- Ensamblador/ Linkador y Herramientas de lenguaje

El ensamblador puede ser utilizado sólo para ensamblar un único fichero, o puede usarse con el linkador (MPLINK) para crear un proyecto a partir de ficheros separados, librerías y objetos recompilados. El linkador es el responsable de situar el código compilado en áreas de memoria del microcontrolador.

MPASM es el ensamblador incorporado en el software MPLAB IDE, que se encarga de traducir los nemónicos y símbolos alfanuméricos del código escrito a código máquina, produciendo como salida un fichero .HEX.

- Depurador

El depurador de Microchip contiene *breakpoints, single stepping, watch windows* y todos los recursos de un depurador moderno para el entorno MPLAB IDE. Funciona a la par del editor con el fin de eliminar los errores existentes en el código fuente.

- Simuladores

Existe software de simulación en MPLAB IDE para todos los PICmicro y dispositivos dsPIC DSC. Estos simuladores utilizan el ordenador para simular las instrucciones y algunas funciones periféricas de dichos dispositivos.

El simulador incorporado en el entorno MPLAB IDE es MPSIM.

4.1.2.2. Módulos adicionales

Además de las herramientas que se encuentran integradas en MPLAB, se puede comprar componentes opcionales y añadirlos al software, tales como:

- Compiladores

Los compiladores de C MPLAB C18 y MPLAB C30 de Microchip, proporcionan un código integrado y optimizado. Además de estos compiladores de C, existen otras herramientas de lenguaje de diversas empresas tales como HI-TECH, IAR, microEngineering Labs, CCS y Byte Craft que son invocadas por el Project Manager para compilar código que se carga automáticamente en el depurador de objetos (*target debugger*) para testearlo y verificar.

- Programadores

PICSTART Plus, PRO MATE II, MPLAB PM3 y MPLAB ICD 2 pueden programar código en los microcontroladores. MPLAB IDE proporciona un control completo sobre el proceso de programación, tanto de códigos como de datos, así como sobre los bits de configuración para establecer los diversos modos de operación de los microcontroladores.

- Emuladores integrados

MPLAB ICE 2000 y MPLAB ICE 4000 son emuladores completos para los microPIC y los dispositivos dsPIC DSC. Se conectan al ordenador mediante los puertos de entrada/salida, y permiten el control completo sobre las operaciones del microcontrolador.

- Depurador integrado

MPLAB ICD 2 proporciona una alternativa económica al emulador. Utilizando algunos de los recursos del chip, MPLAB ICD 2 puede cargar código en el microcontrolador, utilizar *breakpoints*, *single step* y monitorizar registros y variables.

4.1.3. CARACTERÍSTICAS

El software MPLAB IDE, que se encuentra disponible en la página de Microchip [5], permite editar el archivo fuente del proyecto además de ensamblarlo y simularlo en pantalla, pudiendo ejecutarlo en modo paso a paso y ver la evolución de los registros internos, la memoria de datos, la memoria de programa, etc., según se van ejecutando las instrucciones.

A continuación se enumeran las principales características que presenta el software MPLAB:

- Crear y editar código fuente utilizando el Editor.
- Ensamblar, compilar y linkar el código fuente.
- Eliminar fallos de la lógica ejecutable, mirando el flujo de programa con el simulador integrado, o en tiempo real con emuladores o depuradores en circuito.
- Realizar medidas de tiempo con el simulador o el emulador.
- Ver variables en *Watch Windows*.

- Programar aplicaciones fijas en los dispositivos mediante programadores.

4.1.4. ESTRUCTURA DE UN PROYECTO

Un proyecto es un conjunto de programas que se integran en un único módulo con el fin de generar un código ejecutable para un determinado microcontrolador. Dichos programas pueden estar escritos en diferentes lenguajes de programación, como por ejemplo ensamblador, C, Basic, Pascal.

A continuación se muestra la estructura de un proyecto creado con MPLAB, así como sus ficheros de entrada y salida:

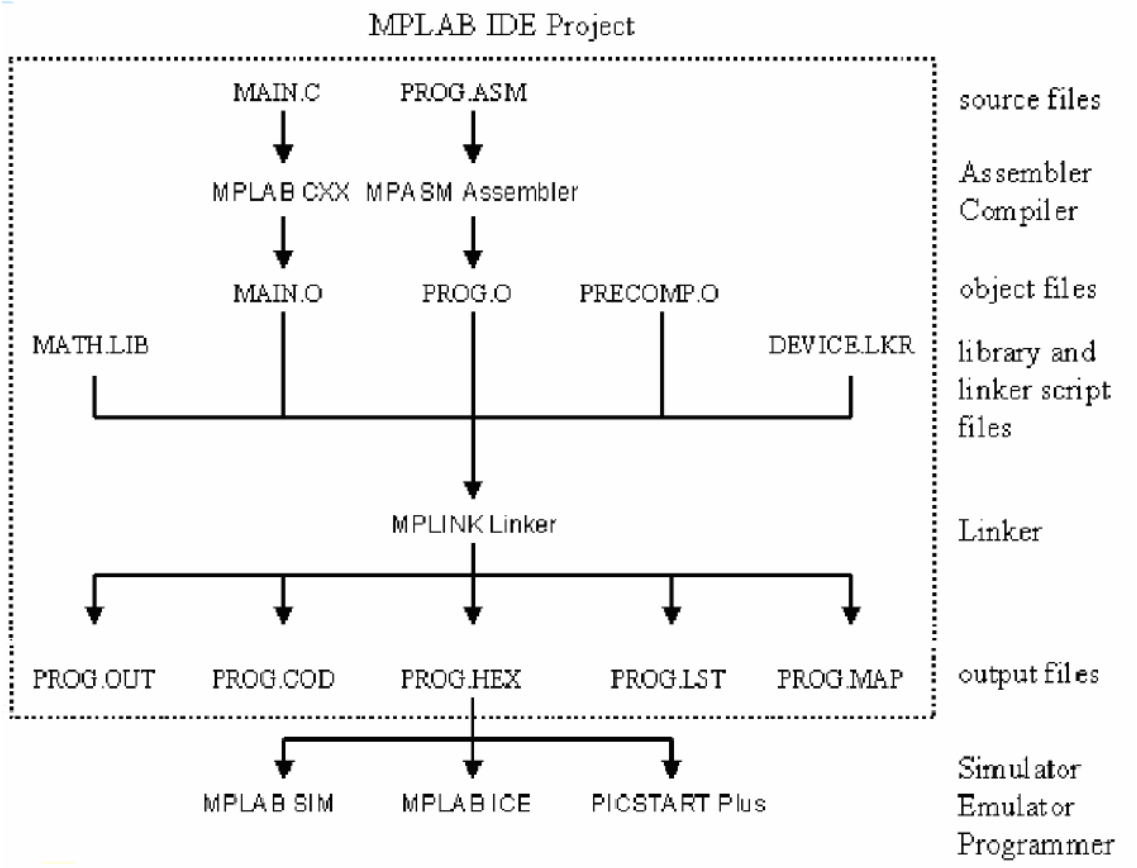


Figura 4.3: Estructura de un proyecto

Los ficheros fuente (que pueden ser varios), pueden estar escritos en ensamblador o en C, con extensiones .asm y .c respectivamente, y mediante

los programas ensamblador y compilador, se obtienen los ficheros objeto con extensión .o.

Todos los ficheros objeto, junto a otros ficheros procedentes de librerías, son linkados, generando una serie de ficheros de salida de los cuales el más importante es el ejecutable .HEX que será el que se grabará en el dispositivo.

4.2. CCS C COMPILER

4.2.1. PROGRAMACIÓN EN C

En el desarrollo de programas para microcontroladores, existen dos alternativas en cuanto al tipo de lenguaje de programación:

- Lenguaje de bajo nivel (ensamblador).
- Lenguaje de alto nivel (BASIC, C, etc.).

Los programas realizados en ensamblador suelen ser más pequeños que los realizados en lenguaje de alto nivel, aprovechan de manera más eficiente los recursos del microcontrolador, donde el obtener un código optimizado tanto en número de instrucciones como en velocidad depende del programador. Muchas veces, el tamaño del programa es un factor muy importante, puesto que la capacidad de la memoria de instrucciones de un microcontrolador es limitada.

En ocasiones, cuando los programas presentan cierta complejidad se opta por el uso de lenguajes de alto nivel para la programación de los microcontroladores, ya que la programación en lenguaje ensamblador requiere demasiado esfuerzo y tiempo.

La utilización de un lenguaje de alto nivel simplifica el desarrollo y mantenimiento de las aplicaciones. El código generado por un buen compilador está muy optimizado, produciendo código bastante eficiente.

Si se desea realizar la programación de los microcontroladores en lenguaje C, es preciso utilizar un compilador C. Dicho compilador genera el fichero necesario, .HEX, para poder programar el PIC utilizando un programador.

4.2.2. COMPILADORES C

Hoy en día existen cuatro compiladores de C fundamentales en el mercado:

- Hi-tech [12]. Este compilador, cuyo entorno de programación es bajo DOS, presenta una excelente optimización de código. Aunque soporta una gran variedad de PICs, no proporciona muchas funciones específicas para éstos.
- IAR C Compiler [13]. Esta casa proporciona una amplia variedad de compiladores y ensambladores para muchos tipos de microcontroladores bajo el entorno de Windows.
- CCS PCW, PCB, PCM, PCH C Compiler [14]. Es un compilador con el cual, se puede realizar proyectos en C sin poseer gran dominio de este lenguaje. Dicho compilador, que se desarrolla bajo Windows, proporciona muchas funciones para realizar proyectos. Además, presenta un método de trabajo a base de plantillas prediseñadas, facilitando el comienzo de la programación.
- C2C C Compiler [15]. Este compilador fue de los primeros compiladores en ofrecer entorno Windows. Aunque es bastante conocido por ser barato y tener versión en español, soporta muy pocos PICs, tiene una documentación insuficiente y su potencia es bastante baja.

4.2.3. CCS C COMPILER

A continuación se describe el Compilador C de la firma CCS [14], ya que es el entorno de desarrollo que se ha utilizado en este proyecto para compilar los programas generados para el microcontrolador.

La versión PCW admite todos los dispositivos PIC de las gamas baja (12xxx) y media (16xxx), con núcleos de 12 y 14 bits respectivamente. Además, dicho compilador incluye un entorno de desarrollo integrado en Windows que permite desarrollar todas y cada una de las etapas que compone un proyecto, desde la edición hasta la compilación pasando por la depuración de errores.

Este compilador traduce el código C del archivo fuente (.c) a lenguaje máquina para los microcontroladores PIC, generando así un archivo en hexadecimal (.HEX)

El fichero .HEX y los archivos de depuración son seleccionables y compatibles con emuladores y programadores populares, incluyendo MPLAB IDE para depuración a nivel de fuente.

4.2.3.1. Características

- Bibliotecas incorporadas que funcionan con todos los chips para Entrada/Salida por puerto serie RS-232, Entradas/Salidas discretas y pausas de precisión.
- Se integra al IDE de MPLAB y otros simuladores y editores para depuración a nivel de fuente. Los archivos .HEX estándar y archivos de depuración, aseguran compatibilidad con todos los programadores.
- Se incluyen drivers de código fuente para módulos LCD, teclados, EEPROMs serie 24xx y 94xx, relojes en tiempo real X10, DS1302 y NJU6355, RAM serie DS2223 y PCF8570, conversores A/D LTV1298 yPCF8591, sensores de temperatura, potenciómetros digitales, etc.
- Acceso a las características del hardware desde funciones en C fáciles de usar, temporizadores, conversión A/D, SSP, PSP, USB, I^2C, etc.
- Se puede insertar código ensamblador en el código fuente y puede referenciar variables de C.

- El linkado automático puede manejar múltiples páginas de código.
- Las directivas del compilador determinan si los registro tri-estado se refrescan en cada E/S o si la E/S es lo más rápida posible.
- Las constantes (incluyendo cadenas y vectores) se almacenan en la memoria de programa.
- El tipo de dato bit estándar (short int) permite que el compilador genere código muy eficiente orientado a Bit.
- Los parámetros de referencia se pueden utilizar para mejorar la lectura del código y la eficiencia de las funciones inline.
- PCW tiene un compilador de línea de comando y un editor/compilador integrado.
- Ventanas especiales muestran el mapa de la memoria RAM, listados C/ensamblador, y el árbol de llamadas.
- Funciones de interrupción soportadas en PCM/PCH. El compilador genera todo el código de inicio y limpieza, así como identifica la función de interrupción correcta a ser llamada.

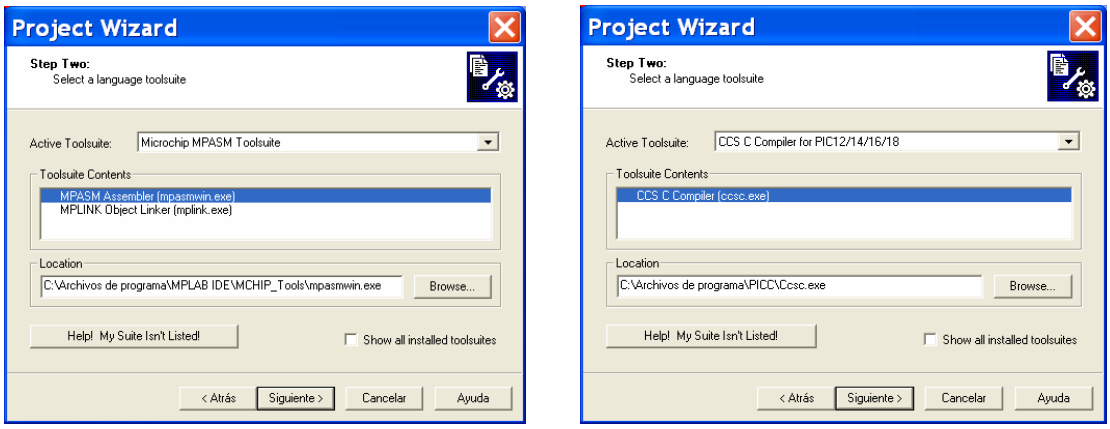
CCS C Compiler, dispone de una amplia gama de funciones de utilidad, bien documentadas, permitiendo al programador estar menos preocupado por el funcionamiento y gestión del hardware interno (registros de datos, de configuración, flags, etc.), para centrarse únicamente en la programación.

4.2.3.2. CCS C Compiler integrado en MPLAB

Como se ha comentado anteriormente, una de las características de este compilador es que se puede integrar en IDE de MPLAB para depurar a nivel de fuente. Esta característica es sumamente útil, ya que se pueden utilizar todas las herramientas del software MPLAB junto a dicho compilador, sin necesidad de tener que ejecutar varios programas.

La diferencia entre utilizar el entorno MPLAB con su ensamblador MPASM y utilizar MPLAB con el compilador de C CCS, estriba en el segundo paso de la realización del proyecto de programación, en el que la herramienta de lenguaje seleccionada en lugar de ser MPASM es CCS C Compiler.

Simplemente seleccionando dicha herramienta de lenguaje, queda integrado el compilador de C en el entorno MPLAB. Esto permite utilizar MPLAB con total normalidad, como si de un proyecto en lenguaje ensamblador se tratase, es decir, tanto las herramientas de edición, depuración, simulación, etc., se utilizan de la misma manera.



(a) **(b)**

Figura 4.4: Ventana de selección de la herramienta de lenguaje en MPLAB

a- Herramienta de lenguaje para código en ensamblador

b- Herramienta de lenguaje para código en C

4.3. PROTEUS

PROTEUS es un entorno integrado, diseñado para la realización completa de proyectos de construcción de equipos electrónicos en todas sus etapas: diseño, simulación, depuración y construcción. El software PROTEUS, desarrollado por Labcenter Electronics [16] y [17], se compone de cuatro elementos:

- ISIS, es la herramienta para la elaboración de esquemas electrónicos. Permite dibujar sobre un área de trabajo un circuito que posteriormente será simulado.

  Las principales utilidades que caracterizan a este software son, entre otras:

  - Una librería de más de 6.000 modelos de dispositivos digitales y analógicos.
  - Conexionado automático entre dos puntos del esquema.
  - Fichero de conexiones, *NetList,* compatible con la mayoría de los programas de realización de PCB.
  - Enumeración automática de componentes.
  - Etc.

- ARES, es la herramienta de rutado de PROTEUS. Se utiliza para la elaboración de placas de circuito impreso con posicionador automático de elementos y generación automática de pistas, que permite el uso de hasta 16 capas.

- PROSPICE, la herramienta de simulación de circuitos según el estándar industrial SPICE3F5. Es la versión SPICE incluida en PROTEUS y desarrollada a partir del modelo Berkeley, con extensiones para la simulación analógica y digital conjunta y la animación de circuitos.

- VSM, es un completo simulador de esquemas electrónicos que contiene microprocesador. PROTEUS es capaz de leer los ficheros con el código

ensamblado para los microprocesadores de las familias PIC, AVR, 8051, HC11, ARM/LPC200 y BASIC STAMP y simular perfectamente su comportamiento. Incluso se puede visualizar el código, interactuar en tiempo real con el hardware utilizando modelos de periféricos animados, tales como displays LED o LCD, teclados, terminales RS232, simuladores de protocolos I^2C, etc.

4.3.1. PRINCIPALES CARACTERÍSTICAS DEL SISTEMA PROTEUS

En este apartado se enumeran las características más importantes del sistema PROTEUS:

- Entorno de diseño gráfico de esquemas electrónicos fácil de utilizar.
- Entorno de simulación prospice mixto entre el estándar SPICE3F5 y la tecnología exclusiva de PROTEUS de Modelación de Sistemas Virtuales (VSM).
- Entorno de diseño de placas de circuito impreso de altas prestaciones, con bases de datos de 32 bits, posicionador automático de elementos y generación automática de pistas con tecnologías de autocorte y regeneración.
- Interfaz de usuario moderna y atractiva.
- Compatibilidad e inter-operatividad de todas las herramientas que componen el entorno PROTEUS.
- Ejecutable en los diferentes entornos Windows: 98, Me, 2000, XP.
- Herramienta de altas prestaciones.

4.3.2. MÓDULO ISIS

A continuación se describe más exhaustivamente el módulo ISIS de PROTEUS, debido a que es una de las herramientas que se ha utilizado en la realización del presente Proyecto Fin de Carrera.

Como ya se ha comentado anteriormente, el programa PROTEUS es una aplicación CAD, que entre sus componentes posee un módulo de captura de esquemas, llamado ISIS.

4.3.2.1. Características

- Permite realizar esquemas de circuitos electrónicos y generar ficheros de conexiones (*NetList*), que sirven para simular el funcionamiento del circuito o bien para realizar el diseño del circuito impreso.
- Incorpora un entorno de diseño de gran potencia, controlando la apariencia final de los dibujos. Dicha herramienta, permite realizar complejos diseños electrónicos de manera rápida y eficaz.
- Contiene una extensa librería donde se incluyen elementos estándar como resistencias, diodos, válvulas, transistores, condensadores, memorias y amplificadores operacionales.
- Con la herramienta gráfica para la gestión de encapsulados, se simplifica el establecimiento de los enlaces entre los esquemas electrónicos y los encapsulados de los dispositivos para su utilización en el diseño de placas.
- Integra la gestión de diseños multihojas, repartiendo el esquema electrónico en varia hojas y facilitando así, la creación de esquemas complejos.

- El diseño jerárquico que incorpora el módulo ISIS, también facilita la tarea de creación de esquemas complejos, ya que se puede definir un componente como un módulo que representa un determinado esquema electrónico y que se comporta de acuerdo con él. ISIS sustituye el módulo por el esquema representado, simplificando así el diseño y el tiempo utilizado en la creación.
- Capacidad de gestionar buses ahorrando tiempo y espacio, simplificando la interpretación del esquema electrónico. Esto permite conectar los extremos del bus de forma repartida entre diversas hojas, o en diferentes niveles jerárquicos.
- Todos los componentes utilizados en la creación de un circuito eléctrico poseen una lista de propiedades. El módulo ISIS permite añadir a esa lista, las propiedades que se crean convenientes.
- Selección de grupos de componentes de acuerdo con sus propiedades y manipulación de las propiedades de un determinado grupo de componentes. Por lo que intercambiar todos los dispositivos de un modelo por otro distinto es una tarea rápida y sencilla.
- Integra elementos comunes de librería:
    - Elementos simples (por ejemplo, temporizador 555).
    - Elementos complejos:
        - Elementos complejos homogéneos (por ejemplo, integrado 7405 de seis puertas inversoras).
        - Elementos complejos heterogéneos (por ejemplo, relé compuesto de bobina y varios conectores).
- El trazado automático de uniones permite crear de manera sencilla las líneas de enlace entre componentes con sólo hacer clic con el ratón sobre los pines a unir.

A continuación se muestra una imagen de la apariencia del entorno de diseño gráfico ISIS.

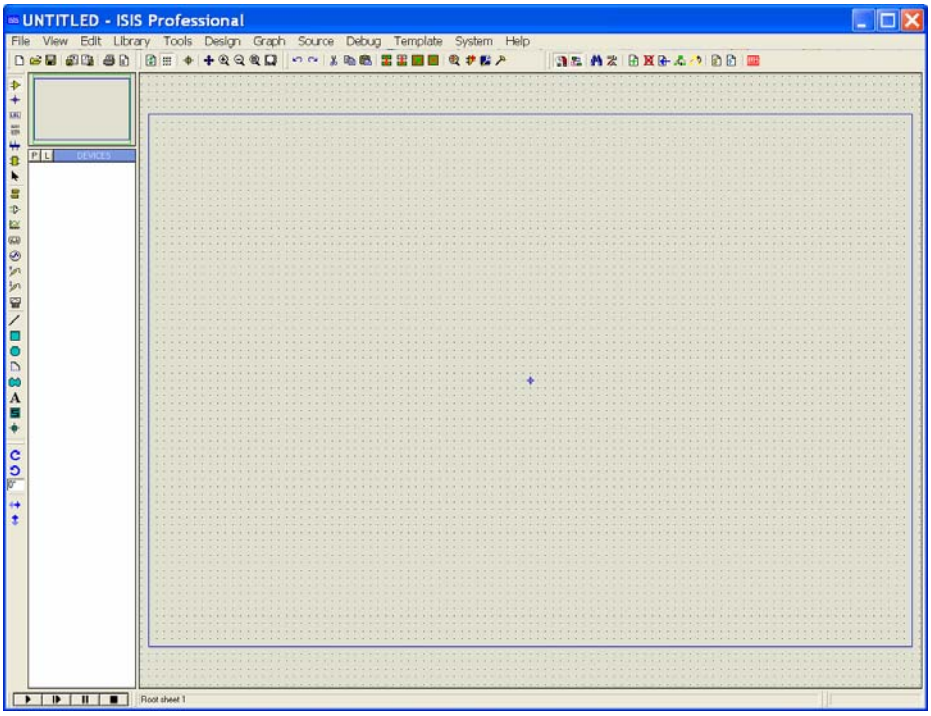


Figura 4.5: Ventana ISIS Professional

CAPÍTULO 5. EVALUACIÓN DE LA TARJETA EASYPIC4

Tras describir en el Capítulo 3 los componentes y características del sistema de desarrollo EasyPIC4, se realiza una serie de pruebas con el fin de verificar que todos los componentes que integran la placa se encuentran en perfecto estado y no presentan ningún comportamiento anómalo.

Estas pruebas consisten en cargar una serie de programas en el microcontrolador, y comprobar que el comportamiento de la placa se corresponde con el código grabado. Para realizar dichas pruebas se han utilizado los programas que vienen de ejemplo en el CD adjunto a la tarjeta EasyPIC4, y que también se encuentran disponibles en la página web de Mikroelektronika [11]. Dichos programas están escritos en lenguaje C, utilizando librerías propias del compilador mikroC.

MikroC es un compilador de C de Mikroelektronika, que al igual que el resto de compiladores de C, genera el fichero necesario .HEX para poder grabar el código en la memoria del microcontrolador utilizando un programador.

No se ha profundizado en este tipo de compilador de C, ya que no ha sido utilizado para compilar los programas ejemplo, puesto que junto a los ficheros en lenguaje C vienen los mismos programas en formato hexadecimal. Por tanto para esta parte del proyecto el software utilizado es únicamente el software PICFlash, ya que al tener el código en lenguaje máquina (.HEX) no es necesaria la utilización de ningún tipo de compilador para pasar de lenguaje C a hexadecimal, cargando así, dicho fichero directamente en la memoria del microcontrolador.

En el CD ajunto al presente proyecto se incluye tanto los programas ejemplo como el manual del software mikroC, donde se puede consultar las diversas librerías que aparecen en los distintos programas que se muestran a continuación.

5.1. PROGRAMAS UTILIZADOS PARA VERIFICAR EL ENTRENADOR

A continuación se detallan algunos de los códigos utilizados, así como una breve descripción del comportamiento del entrenador EasyPIC4. Las siguientes aplicaciones se han clasificado en función del dispositivo implicado en la ejecución del programa.

5.1.1. DISPLAYS DE 7 SEGMENTOS

Los siguientes programas muestran la utilización de los displays de 7 segmentos. Todos los displays, como se ha comentado en anteriormente, están conectados a PORTD por el que reciben los datos, de manera que el segmento “a” del display está conectado a RD0, el segmento “b” a RD1 y así sucesivamente.

Estos displays se habilitan mediante los cuatros bits menos significativos de PORTA: RA0, RA1, RA2 y RA3.

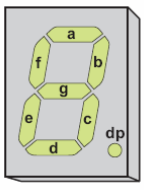


Figura 5.1: Segmentos del display de 7 segmentos

- Fichero “Seg2Cif_Static.c”

Descripción:

El siguiente código permite mostrar un número estático en dos de los cuatro displays de 7 segmentos. Ambos displays están conectados a PORTD que se refresca mediante los pines de PORTA (RA0, RA1).

Código en C:

```
unsigned short v, por1, por2;

void interrupt() {
  if (v==0) {
    PORTD = por2;      //  send mask for digit 3 to portb
    PORTA = 1;         //  turn on 1st 7 seg., turn off 2nd
    v = 1;
  } else {
    PORTD = por1;      //  send mask for digit 8 to portb
    PORTA = 2;         //  turn on 2nd 7 seg., turn off 1st
    v = 0;
    }
  TMR0 = 0;            //  clear TMRO
  INTCON = 0x20;       //  clear TMR0IF and set TMR0IE
}//~

void main() {

  OPTION_REG  = 0x80;   // pull up resistors
  PORTA       =   0;    // clear porta (make sure both displays are off)
  TRISA       =   0;    // designate porta pins as output
  PORTD       =   0;    // clear portb (make sure LEDs are off)
  TRISD       =   0;    // designate portb pins as input
  TMR0        =   0;    // clear TMRO
  por1        = 0x7F;   // mask for 8 (7 seg. display)
  por2        = 0x4F;   // mask for 3 (7 seg display)
  INTCON      = 0xA0;   // enable T0IE
  // wait for interrupt
}//~!
```

Analizando el código se puede observar que para mostrar el número se ha de estar cambiando el valor de PORTD al mismo tiempo que se cambia de RA0 a RA1 y viceversa. Es decir, mientras RA0 sea 1, el valor de PORTD ha de ser 0x4F para que se muestre en el display de las unidades el número 3, y cuando el valor de RA1 es 1, el display de las decenas muestra el dígito 8 (0x7F), que es el valor que hay en PORTD. El tiempo transcurrido con los displays en ON se regula mediante el uso del TMR0.

La salida obtenida es, por tanto, el número 83 mostrado en los dos primeros displays situados más a la derecha.

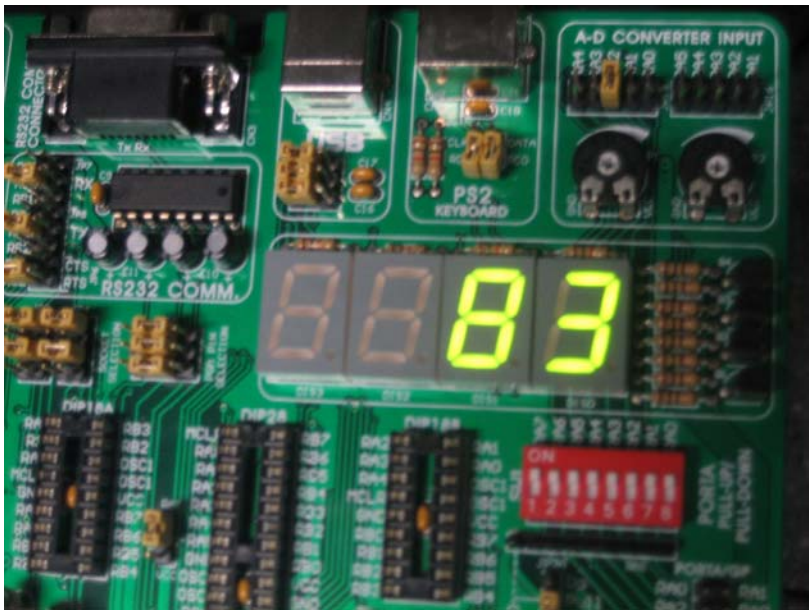


Figura 5.2: Apariencia de los displays de 7 segmentos tras grabar el programa Seg2Cif_Static

- Fichero “Seg4Cif_Static.c”

Descripción:

El siguiente fichero muestra, utilizando interrupciones, un número estático en los cuatro displays de 7 segmentos.

Código en C:

```
unsigned short zz, v;
unsigned short por[4];

void interrupt() {
  PORTD   = por[v];          // send appropriate mask to portd
  PORTA   = zz;              // turn on appropriate 7seg. display
  v++ ;
  zz <<= 1;                  //..prepare for next digit
  if (zz > 8)
    zz = 1;
  if (v > 3)
    v = 0;
  TMR0 = 0;                  // reset timer
  INTCON = 0x20;             // clear TMR0IF and set TMR0IE
}//~


void main() {
```

```
  OPTION_REG  = 0x80;        //pull up
  v          = 0;
  zz          = 1;
  TMR0        = 0;
  INTCON      = 0xA0;        // disable PEIE, INTE,   RBIE...; enable T0IE
  PORTA       = 0;           // make sure that all segments are off
  TRISA       = 0;           // designate porta pins as output
  PORTD       = 0;           // make sure that nothing is displayed
  TRISD       = 0;           // designate portd pins as output
  por[0]      = 0x66;        // mask for 4 (7seg.)
  por[1]      = 0x6D;        // mask for 6 (7seg.)
  por[2]      = 0x7D;        // mask for 5 (7seg.)
  por[3]      = 0x7F;        // mask for 8 (7seg.)
  // wait for interrupt
}//~!
```

Como se puede apreciar, el programa es similar al anterior pero en este caso en lugar de utilizar varias variables para almacenar los datos a cargar en PORTD, se ha utilizado un vector de variables, “por”, donde se encuentran todos los dígitos a mostrar en los displays. Además, se utiliza la variable “v” para seleccionar el dígito a mostrar y la variable “zz” para encender el display correspondiente.

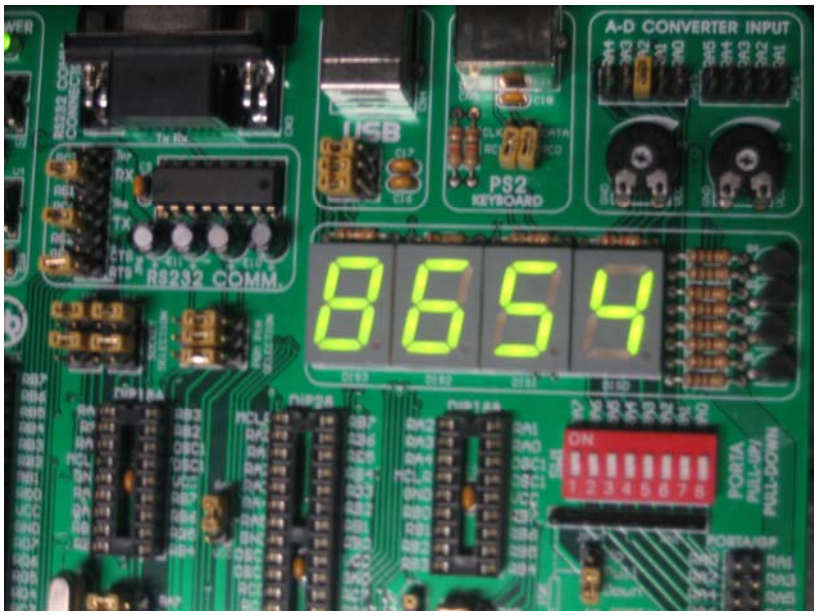


Figura 5.3: Apariencia de los displays de 7 segmentos tras grabar el programa Seg4Cif_Static

- Fichero “Display_1.c”

Descripción:

Al grabar este código en el microcontrolador, se puede observar como el display de 7 segmentos conectado al pin RA1 (segundo display de la derecha) empieza a contar segundos.

Código en C:

Fichero Display_1.c

```
#include "Display_utils.h"

unsigned short i;

void main() {
  INTCON = 0;                 // Disable PEIE,INTE,RBIE,T0IE
  PORTA  = 0;
  TRISA  = 0;
  PORTD  = 0;
  TRISD  = 0;
  do {
    for (i = 0; i<=9u; i++) {
      PORTA = 0;
      PORTD = mask(i);
      PORTA = 2;
      Delay_ms(1000);
    }
  } while (1);                //endless loop
}//~!
```

Para poder ejecutar el programa anterior se ha utilizado el siguiente archivo, que ha sido incluido en el fichero principal mediante el comando #include "Display_utils.h".

Fichero Display_utils.h

```
// Header File for <Display_utils.c>
unsigned short mask(unsigned short num);
```

Fichero Display_utils.c

```
//-------------- Returns mask for common cathode 7-seg. display
unsigned short mask(unsigned short num) {
  switch (num) {
    case 0 : return 0x3F;
    case 1 : return 0x06;
    case 2 : return 0x5B;
```

```
    case 3 : return 0x4F;
    case 4 : return 0x66;
    case 5 : return 0x6D;
    case 6 : return 0x7D;
    case 7 : return 0x07;
    case 8 : return 0x7F;
    case 9 : return 0x6F;
  } //case end
}//~
```

Como se puede observar, el fichero “Display_utils.c” devuelve un número hexadecimal que indica los segmentos del display que se han de encender en cada caso, simplificando así el código implementado en el fichero principal.

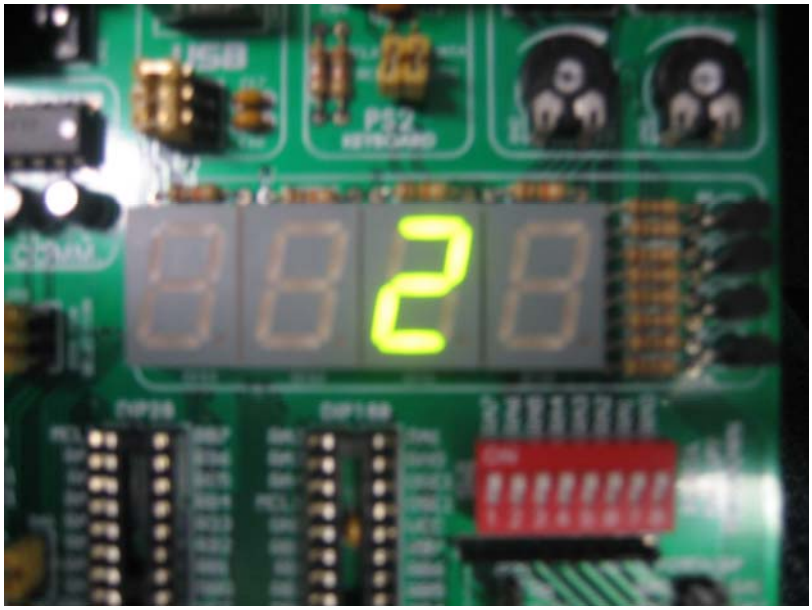


Figura 5.4: Apariencia del display de 7 segmentos a los 2 segundos de grabar el programa Display_1

- Fichero “Display7seg_03.c”

Descripción:

Este ejemplo también realiza una cuenta de segundos pero en este caso se utiliza los cuatros displays de 7 segmentos.

Código en C:

Fichero Display7seg_03.c

```
#include "Display_utils.h"

char kraj;
unsigned short zz, j, v;
unsigned int i;
unsigned short por[4];

void interrupt() {
  PORTA = 0;
  PORTD = por[v];
  PORTA = zz;              // turn on appropriate 7seg. display
  zz <<= 1;
  if (zz > 8u)
    zz = 1;               // prepare mask for digit

  v++ ;
  if (v > 3u)
    v = 0;                // turn on 1st, turn off 2nd 7seg.

  TMR0   =   0;
  INTCON = 0x20;
}//~

void main() {
  OPTION_REG = 0x80;
  j          =   0;
  v          =   0;
  zz         =   1;
  TMR0       =   0;
  INTCON     = 0xA0;       // Disable PEIE,INTE,RBIE,T0IE
  PORTA      =   0;
  TRISA      =   0;
  PORTD      =   0;
  TRISD      =   0;

  i          =   0;
  do {
    j = i / 1000u ;        // prepare digits for diplays
    kraj = mask(j);
    por[3] = kraj;
    j = (char)(i / 100u) % 10u;
    kraj = mask(j);
    por[2] = kraj;
    j = (char)(i / 10u) % 10u;
    kraj = mask(j);
    por[1] = kraj;
    j = i % 10u;
    kraj = mask(j);
    por[0] = kraj;

    Delay_ms(1000);

    i++ ;                  // increment counter
    if (i > 9999u)
      i = 0;
```

```
  } while(1);             // endless loop
}//~
```

Al igual que ocurría en el caso anterior, se realiza un fichero donde se almacenan los números que se han de cargar en PORTD para que el display de 7 segmentos muestre el número correspondiente.

Fichero Display_utils.h

```
// Header File for <Display_utils.c>
unsigned short mask(unsigned short num);
```

Fichero Display_utils.c

```
//--------------- Returns mask for common cathode 7-seg. display
unsigned short mask(unsigned short num) {
  switch (num) {
    case 0 : return 0x3F;
    case 1 : return 0x06;
    case 2 : return 0x5B;
    case 3 : return 0x4F;
    case 4 : return 0x66;
    case 5 : return 0x6D;
    case 6 : return 0x7D;
    case 7 : return 0x07;
    case 8 : return 0x7F;
    case 9 : return 0x6F;
  } //case end
}//~
```

5.1.2. LEDS

- Fichero “Led_Blinking.c”

Descripción:

Con el siguiente código, todos los diodos conectados a PORTC parpadean, cambiando su estado de ON a OFF y viceversa.

Código en C:

```
void main() {
PORTC = 0;
TRISC = 0;

while(1) {
```

```
PORTC =  ~PORTC;
Delay_ms(1000);
}
} //~!
```

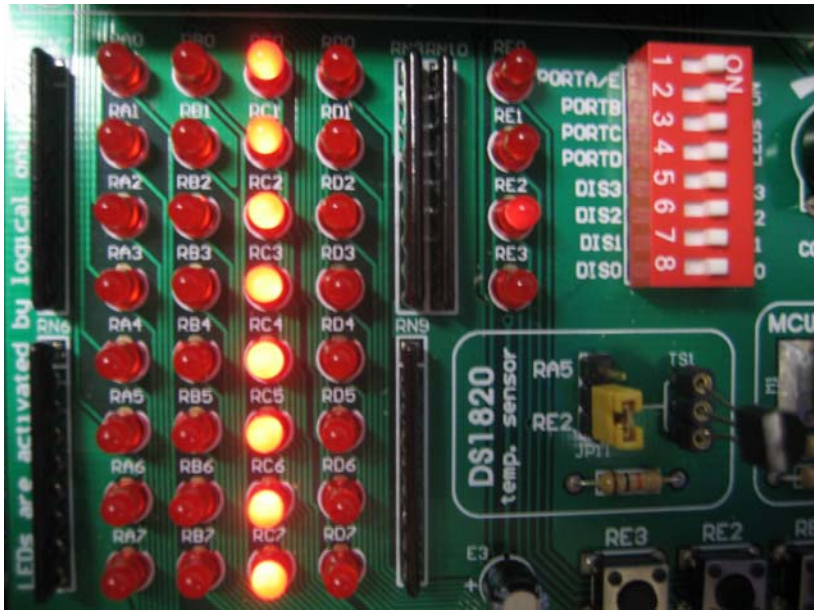


Figura 5.5: LEDs de PORTC en ON

- Fichero “Timer1_01.c”

Descripción:

Mediante el siguiente código se consigue que los LEDs conectados a PORTB parpadeen. Se ha utilizado el TMR1 para realizar la interrupción y temporizar el tiempo en el cual los LEDs se encuentran encendidos. Se puede ver como se encienden cuatro LEDs de PORTB mientras los otros cuatros están apagados y posteriormente se apagan éstos para encenderse los cuatro LEDs restantes, y así sucesivamente.

Código en C:

```
unsigned short cnt;
unsigned char tb;

void interrupt() {
  cnt++ ;
  PIR1.TMR1IF = 0;          // clear TMR1IF
}//~

void main() {
  TRISB = 0;
```

```
  T1CON = 1;
  PIR1.TMR1IF = 0;           // clear TMR1IF
  PIE1  =   1;               // enable interrupts
  PORTB = 0xF0;
  cnt =   0;                 // initialize cnt
  INTCON = 0xC0;

  do {
    if (cnt == 152) {        // if cnt is 152, then toggle portb leds and
      PORTB  = ~PORTB;       //   reset cnt
      cnt = 0;
    }
  } while (1);
}//~!
```

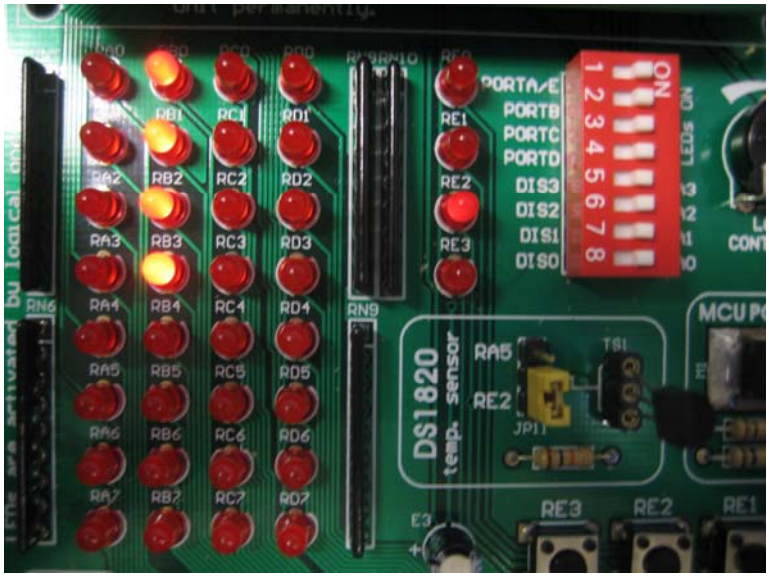


Figura 5.6: Los cuatro LEDs de PORTB en ON

- Fichero “Tmr0.c”

Descripción:

El siguiente fichero es muy similar al mostrado anteriormente. La diferencia es que en este programa se ha utilizado el TMR0 en lugar de TMR1 para hacer parpadear, cada segundo, los LEDs conectados a PORTB.

Código en C:

```
unsigned cnt;

void interrupt() {
  cnt++;                   // Increment value of cnt on every interrupt
  TMR0   = 96;
  INTCON = 0x20;           // Set T0IE, clear T0IF
}//~
```

```
void main() {
  OPTION_REG = 0x84;       // Assign prescaler to TMR0
  TRISB = 0;               // PORTB is output
  PORTB = 0xFF;            // Initialize PORTB
  TMR0  = 96;
  INTCON = 0xA0;           // Enable TMRO interrupt
  cnt = 0;                 // Initialize cnt

  do {
    if (cnt == 400) {
      PORTB = ~PORTB;      // Toggle PORTB LEDs
      cnt = 0;             // Reset cnt
    }
  } while(1);
}//~!
```

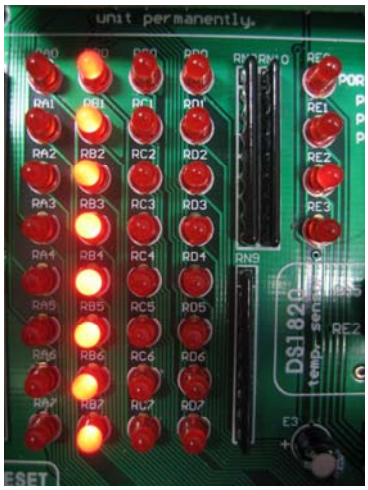


Figura 5.7: LEDs de PORTB en ON

5.1.3. LCD

- Fichero “Lcd_Test.c”

Descripción:

El siguiente código es una simple demostración de las funciones de la librería del LCD de mikroC. Lo primero que se realiza es la inicialización del LCD (PORTD, interfaz de datos de 4 bits, valores de los pines por defecto), luego se escribe el texto almacenado en *text*, “mikroElektronika”, en la primera línea y en la parte inferior, la palabra “mikroE”.

Código en C:

```
char *text = "mikroElektronika";

void main() {
  LCD_Init(&PORTD);         // Initialize LCD connected to PORTB

  LCD_Cmd(LCD_CLEAR);       // Clear display
  LCD_Cmd(LCD_CURSOR_OFF);  // Turn cursor off
  LCD_Out(1,1, text);       // Print text to LCD, 2nd row, 1st column
  Delay_ms(1000);
  LCD_Out(2,6,"mikroE");
}//~!
```

Figura 5.8: Imagen del LCD tras grabar el programa Lcd_Test.

- Fichero “ADC_on_LCD.c”

Descripción:

El siguiente programa muestra un ejemplo de la utilización del conversor analógico/digital existente en el sistema de desarrollo EasyPIC4.

El LCD presenta, durante 2 segundos, la siguiente apariencia:

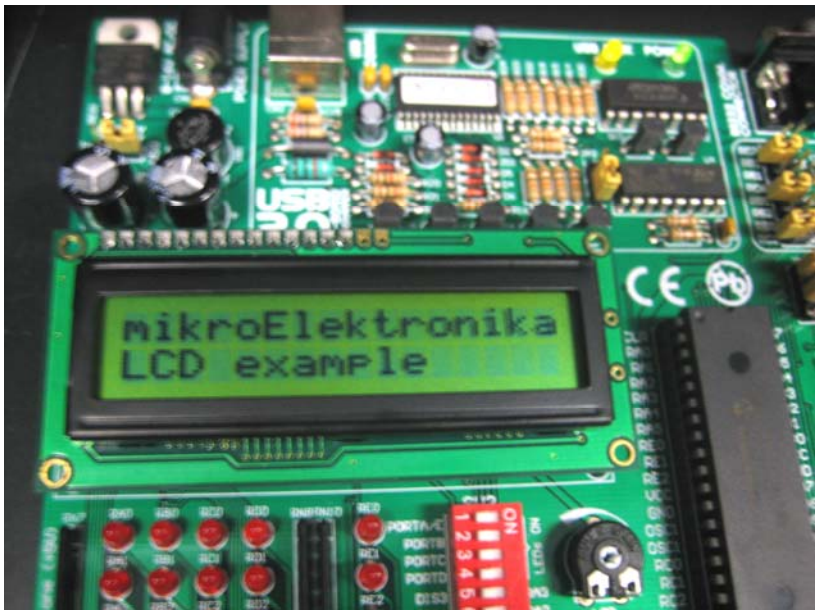


Figura 5.9: Imagen del LCD tras grabar el programa ADC_on_LCD

Posteriormente le sigue la siguiente pantalla,

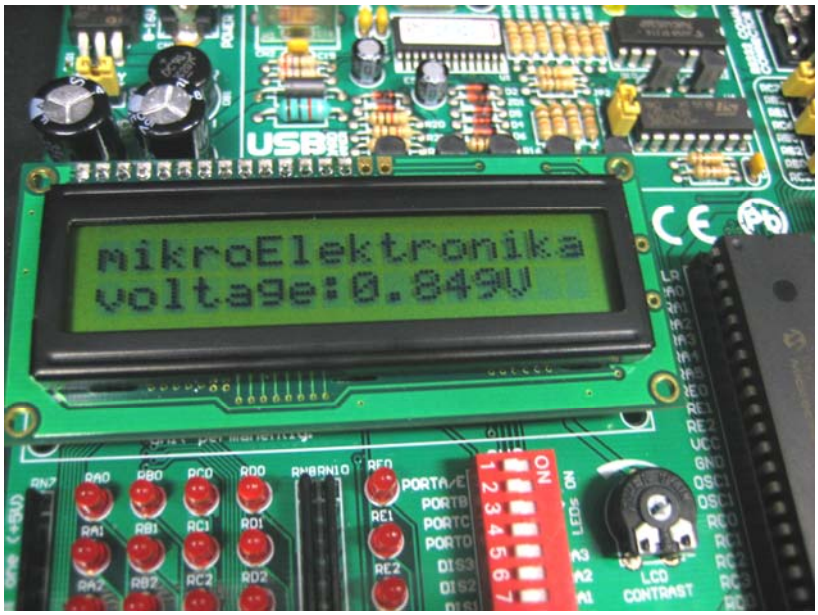


Figura 5.10: Tensión en RA2.

El valor mostrado en el display LCD refleja la tensión que tiene la entrada analógica RA2. Para ello se ha de situar el jumper JP15 en RA2 y mediante el potenciómetro P1 modificar la tensión que recibe el pin RA2. Con el fin de realizar una medida de la señal analógica sin interferencias, los interruptores SW1 deben situarse en la posición OFF.

Código en C:

Fichero ADC_on_LCD.c

```
#include"built_in.h"

unsigned char ch;
unsigned int t;
char a[17], *tc;
long tlong;

void main() {
  INTCON = 0;                    // disable all interrupts
  LCD_Init(&PORTD);              // initialize  (4-bit interface connection)
  LCD_Cmd(LCD_CURSOR_OFF);       // send command to LCD (cursor off)
  LCD_Cmd(LCD_CLEAR);            // send command  to LCD (clear LCD)

  tc = "mikroElektronika";       // assign text to string a
  LCD_Out(1,1,tc);               // print string a on LCD, 1st row, 1st column
  tc = "LCD example";            // assign text to string a
  LCD_Out(2,1,tc);               // print string a on LCD, 2nd row, 1st column

  OPTION_REG  = 0x80;
  ADCON1     = 0x82;             // configure VDD as Vref, and analog channels
  TRISA      = 0xFF;             // designate porta as input
  TRISC  = 0;
  TRISD  = 0;
  Delay_ms(2000);
  tc  = "voltage:";              // assign text to string a
  while (1) {
    t  = ADC_read(2);            // get ADC value from 2nd channel
    LCD_Out(2,1,tc);             // print string a on LCD, 2nd row, 1st column

    tlong  = t * 5000;           // use (int) multiplication instead of (long)
    asm {                        //   and fill the upper two bytes manually
      MOVF STACK_2,W
      MOVWF _tlong+2
      MOVF STACK_3,W
      MOVWF _tlong+3
    }
    t = tlong >> 10;

    ch     = t / 1000;           // prepare value for diplay
    LCD_Chr(2,9,48+ch);          // write ASCII at 2nd row, 9th column
    LCD_Chr_CP('.');

    ch    = (t / 100) % 10;
    LCD_Chr_CP(48+ch);

    ch    = (t / 10) % 10;
    LCD_Chr_CP(48+ch);

    ch    = t % 10;
    LCD_Chr_CP(48+ch);
    LCD_Chr_CP('V');

    Delay_ms(1);
  }
}//~!
```

Fichero Built_in.h

```
#define Lo(param) ((char *)&param)[0]
#define Hi(param) ((char *)&param)[1]
#define Higher(param) ((char *)&param)[2]
#define Highest(param) ((char *)&param)[3]
#define lo(param) ((char *)&param)[0]
#define hi(param) ((char *)&param)[1]
#define higher(param) ((char *)&param)[2]
#define highest(param) ((char *)&param)[3]
```

5.1.4. GLCD

- Fichero “Glcd_Test.c”

Descripción:

A continuación se muestra el código grabado en el microcontrolador para ver el comportamiento del display GLCD.

En el siguiente código se utilizan diversas funciones de tratamiento del LCD gráfico. Todas estas funciones se encuentran descritas, como se ha comentado anteriormente, en el manual de mikroC.

Código en C:

Fichero Glcd_Test.c

```
//Declarations---------------------------------------------------------------
#include "bmp1.h"         //incluye la librería bmp1 donde se
//-------------------------------------------------------------end-declarations

void delay2S(){
  delay_ms(2000);
}

void main() {
  unsigned short ii;
  unsigned int jj;
  char *someText;

  //Glcd_Init(&PORTB, 2,0,3,5,7,1, &PORTC);     // Init for EasyPIC2 board
  //Glcd_Init(&PORTB, 2,3,4,5,7,6, &PORTD);      // Init for EasyPic4 board
  Glcd_Init(&PORTB, 0,1,2,3,5,4, &PORTD);      // Init for EasyPIC4board
  Glcd_Fill(0x00);

  lMainLoop:
  Glcd_Image( maska_bmp );
  Delay2S(); Delay2S();

  Glcd_Fill(0x00);
```

```
  Glcd_Line(120,1, 5,60, 1);
  Glcd_Line(12,42, 5,60, 1);
  delay2S();

  Glcd_Rectangle(12,20, 93,57, 1);
  delay2S();

  Glcd_Line(120,12, 12,60, 1);
  delay2S();

  Glcd_H_Line(5,15, 6, 1);
  Glcd_Line(0,12, 120,60, 1);
  Glcd_V_Line(7,63, 127, 1);
  delay2S();

  for (ii = 1; ii <= 10; ii++)
    Glcd_Circle(63,32, 3*ii, 1);
  delay2S();

  Glcd_Box(12,20, 70,57, 2);
  delay2S();

  someText = "BIG:LETTERS";
  Glcd_Write_Text(someText, 5,3, 2);
  delay2S();

  someText = "SMALL:NOT:SMALLER";
  Glcd_Write_Text(someText, 20,5, 1);
  delay2S();

  goto lMainLoop;

}//~!
```

Fichero bmp1.h

```
extern const unsigned short
 mikro_test_bmp[1024],
 maska_bmp[1024];
```

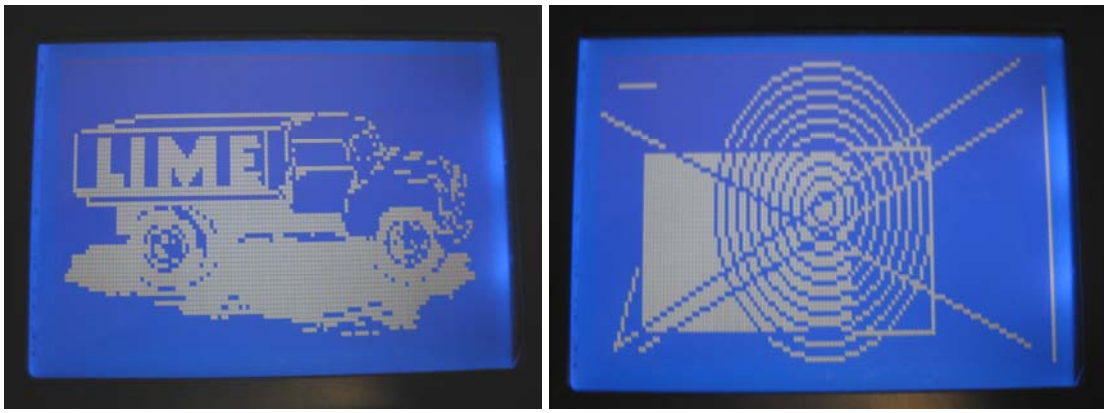


Figura 5.11: Imágenes mostradas en este programa

CAPÍTULO 6. DESARROLLO DE PROGRAMAS PARA LOS MÓDULOS LCD Y GLCD

6.1. INTRODUCCIÓN

El presente capítulo tiene como finalidad el describir cuáles han sido los métodos seguidos y las aplicaciones desarrolladas con los módulos LCD y GLCD, y el microcontrolador PIC16F877A.

El objetivo final buscado es la creación de varios programas que muestren el funcionamiento de dichos módulos, así como las distintas funciones empleadas en la elaboración de las distintas aplicaciones.

6.2. ELECCIÓN DEL ENTORNO DE PROGRAMACIÓN DEL MICROCONTROLADOR

A la hora de elegir el software de programación, se ha tenido en cuenta la necesidad de programar en un lenguaje de alto nivel y en particular en lenguaje C, que es el más difundido para este tipo de aplicaciones. La necesidad de utilizar un lenguaje de alto nivel radica en las complicaciones que pueden surgir sobre todo en la implementación de las rutinas de comunicación con distintos periféricos. Dada la complejidad de dichas rutinas, se ha considerado que el uso del lenguaje ensamblador podría acarrear muchas complicaciones.

Por ello, al principio se decidió utilizar el software mikroC, que es el entorno de programación que se adquiere al comprar el entrenador EasyPIC4. El software mikroC Compiler 6.2.0.0 es un compilador de C que “traduce” el código C del archivo fuente (.C) a lenguaje máquina, generando así un archivo en formato hexadecimal (.HEX).

El motivo de elección de este compilador se debe a que fue un software desarrollado por la misma corporación que desarrolla la placa de evaluación EasyPIC4, Mikroelektronika [11], y por tanto facilitaba todas las librerías necesarias para poder utilizar los distintos módulos y dispositivos del entrenador EasyPIC4, así como los distintos programas ejemplo que han sido utilizados para verificar el comportamiento del sistema de desarrollo, a partir de los cuales se podría desarrollar nuevo software. El tener todas las funciones necesarias para poder realizar programas con los diversos dispositivos que contiene la tarjeta de evaluación, facilita la tarea de programación, ya que no ha de ser creado ningún driver para poder utilizar ese dispositivo.

Tal es así que se programaron e implementaron aplicaciones sencillas manejando puertos, displays de 7 segmentos, LCD, etc. Sin embargo, surgió una dificultad: el software libre de mikroC estaba limitado y sólo compilaba programas con un determinado número de líneas de código. Por tanto, programas muy complejos no podían ser compilados al no disponer de la licencia completa del programa. Esta es la razón por la cual se decidió trabajar desde MPLAB utilizando otro compilador.

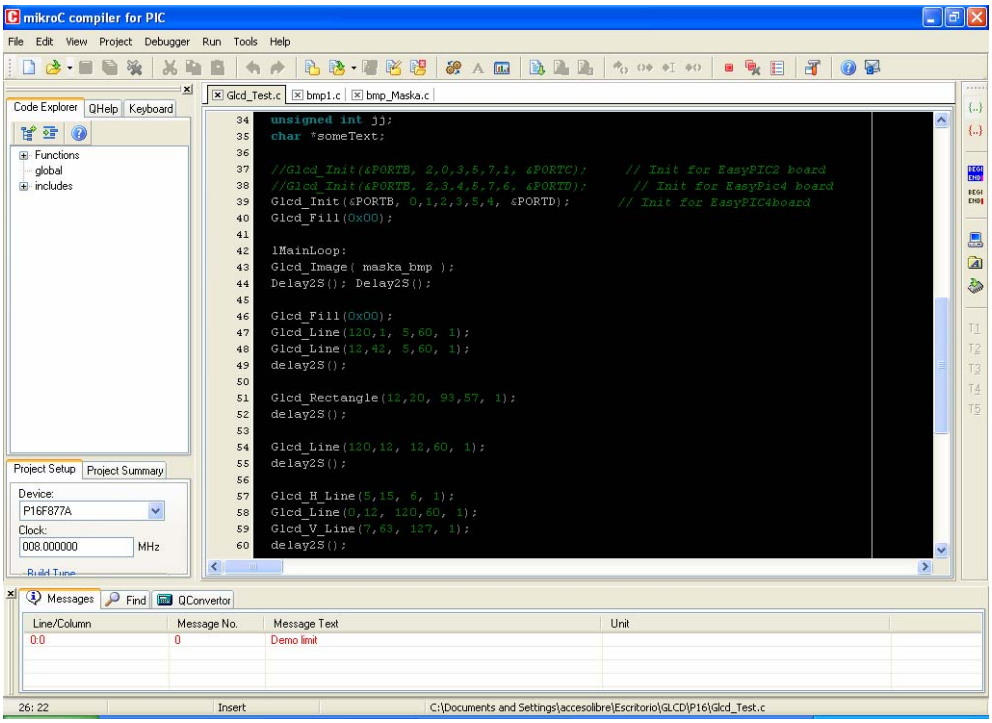


Figura 6.1: Apariencia del entorno mikroC. Ventana de mensajes con “Demo Limit”

El atractivo de utilizar MPLAB se basa en que dicho software fue desarrollado por Microchip, empresa encargada de desarrollar la familia de microcontroladores PIC. Otro motivo por el cual se optó por la utilización de dicho entorno fue, su extensa difusión en entornos de programación de microcontroladores, utilizándose en gran medida en aplicaciones docentes.

En el caso del entorno MPLAB, la principal dificultad radica en que la versión del software con distribución gratuita no posee integrado un compilador de C, por lo que no puede ser utilizado si no se cuenta con la licencia correspondiente.

Teniendo en cuenta lo anterior, se optó por utilizar la versión MPLAB v 6.4, de distribución gratuita, a la que se le puede añadir un compilador en C. Dicha versión, disponible en la página web de Microchip [5], se incluye en el CD-ROM adjunto a la memoria de este Proyecto Fin de Carrera. Además en el CD también se encuentra el manual del software MPLAB donde se detalla, tanto el proceso de instalación, como los pasos a seguir para la realización de proyectos.

El siguiente paso, fue la elección del compilador de lenguaje C a utilizar, para ello se analizaron los compiladores que soporta el software MPLAB de las distintas empresas: HI-TECH [12], IAR [13], MicroEngineering Labs [18], CCS [14] y Byte Craft [19]. Tras analizar los compiladores más importantes, se decidió utilizar el compilador CCS C Compiler, que aunque es un compilador de pago, posee gran cantidad de funciones y librerías para trabajar con distintos módulos: LCD, GLCD, teclados, sensores, etc. Además, al realizar dicha elección también se tuvo en cuenta que para poder crear programas para el microbot Moway de la empresa Bizintek Innova [20] y [21], eran necesarias distintas librerías para el manejo de la odometría y motores, las cuales están implementadas para dicho compilador de C.

Este compilador posee una extensa bibliografía sobre el manejo de las funciones. En el CD-ROM adjunto a la memoria del presente proyecto, se encuentra el manual del compilador donde se halla una guía completa sobre su utilización, el uso de sus funciones específicas y varios ejemplos de aplicación.

6.3. DESARROLLO DE LOS DRIVERS PARA SU USO CON EL COMPILADOR CCS C COMPILER

El primer paso antes de comenzar a realizar cualquier programa utilizando el LCD y el GLCD, es verificar si las librerías, “LCD.c” y “GLCD.c” que contiene el software CCS C Compiler son compatibles con los módulos LCD y GLCD que presenta el entrenador EasyPIC4.

Analizando los drivers y tras la realización de programas sencillos para comprobar el uso de las distintas funciones, se observa que dichas librerías no son las apropiadas para desarrollar programas con estos modelos de displays, por lo que es necesario crear nuevas librerías cuyos parámetros se correspondan con las características de los módulos.

6.3.1. DRIVER PARA EL MÓDULO LCD

Para crear un driver en lenguaje C para el compilador C de CCS que funcione con el módulo LCD del entrenador EasyPIC4 en modo 4 bits, lo primero que se ha de analizar son las conexiones que presenta dicho dispositivo con el microcontrolador, ya que las funciones de la librería utilizan distintos pines para la realización de las tareas.

La forma de conexión del LCD al PIC viene determinada por la siguiente tabla:

Pin Nº	Símbolo	Conexión	Descripción
1	GND	GND	Tierra de alimentación
2	Vcc	Vcc	Alimentación de 5V
3	Vee	Vee	Contraste del cristal líquido. Se conecta a un potenciómetro a través del cual se aplica una tensión variable entre 0 y +5V, que permite regular el contraste del cristal líquido.
4	RS	RD2	Selecciona entre el registro de control y el registro de datos: RS=0 Selección del registro de control RS=1 Selección del registro de datos
5	R/W	GND	Señal de lectura/escritura: R/W=0 Escritura (Write) R/W=1 Lectura (Read)
6	E	RD3	Señal de activación del módulo LCD
7-10	D0:D3	GND	Bus de datos bi-direccional. Se realiza la transferencia de información entre el módulo LCD y el microcontrolador
11-14	D4:D7	RD4:RD7	

Tabla 6.1: Tabla de conexionado del LCD

A continuación se muestra el esquema de conexión entre el display GLCD y el PIC16F877A.

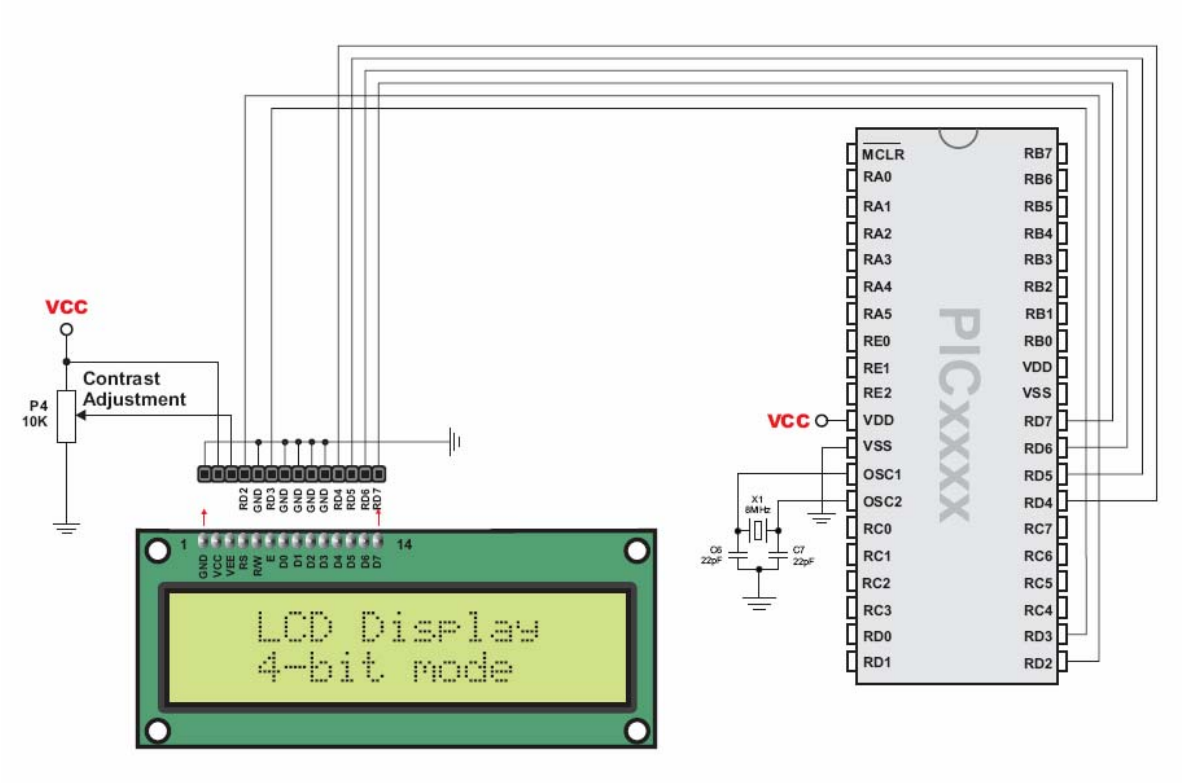


Figura 6.2: Conexión del LCD mediante bus de 4 bits

Una vez analizados los pines y sus respectivas conexiones, se procede a la modificación de la librería “LCD.c” presente en el software CCS. Al intentar modificarla se observa que la realización de dicha modificación no es del todo inmediata, por lo que se buscan drivers equivalentes al “LCD.c” que posean una estructura más sencilla para poder adaptar la librería a las características del dispositivo utilizado.

Tras realizar varias pruebas y crear distintos programas con el fin de comprobar dichas librerías, se desarrolla un driver con el nombre de miLCD.c basado en el driver “flex_lcd.c” que se encuentra disponible en el CD de CCS y en su página web [14]. Esta librería es una adaptación del driver LCD de CCS, que permite especificar los pines del LCD al principio del driver, característica muy útil cuando el dispositivo no puede ser utilizado con el driver “LCD.c”.

El código correspondiente al fichero “miLCD.c” se encuentra disponible en el Anexo A del presente documento.

6.3.2. DRIVER PARA EL MÓDULO GLCD

Como se ha comentado anteriormente, a la hora de crear cualquier driver para un dispositivo, se han de tener claras las conexiones existentes y el funcionamiento de cada pin.

La forma de conexión del GLCD al PIC viene determinada por la siguiente tabla:

Pin Nº	Símbolo	Conexión	Descripción
1	/CS1	RB1	Chip Selection
2	/CS2	RB0	
3	GND	GND	Tierra de alimentación
4	Vcc	Vcc	Alimentación
5	Vo	Vo	Contraste del cristal líquido
6	RS	RB2	Selecciona entre el registro de control y el registro de datos
7	R/W	RB3	Señal de lectura/escritura
8	E	RB4	Señal de activación del módulo LCD
9-16	D0:D7	RD0:RD7	Bus de datos bi-direccional
17	RST	RB5	Señal de reset
18	Vee	Vee	
19	LED+	Voltage(4.2V)	Ánodo. Back-Light anode
20	LED-	GND	Cátodo. Back-Ligth cathode

Tabla 6.2: Tabla de conexionado del GLCD

A continuación se muestra el esquema de conexión entre el display GLCD y el PIC16F877A.

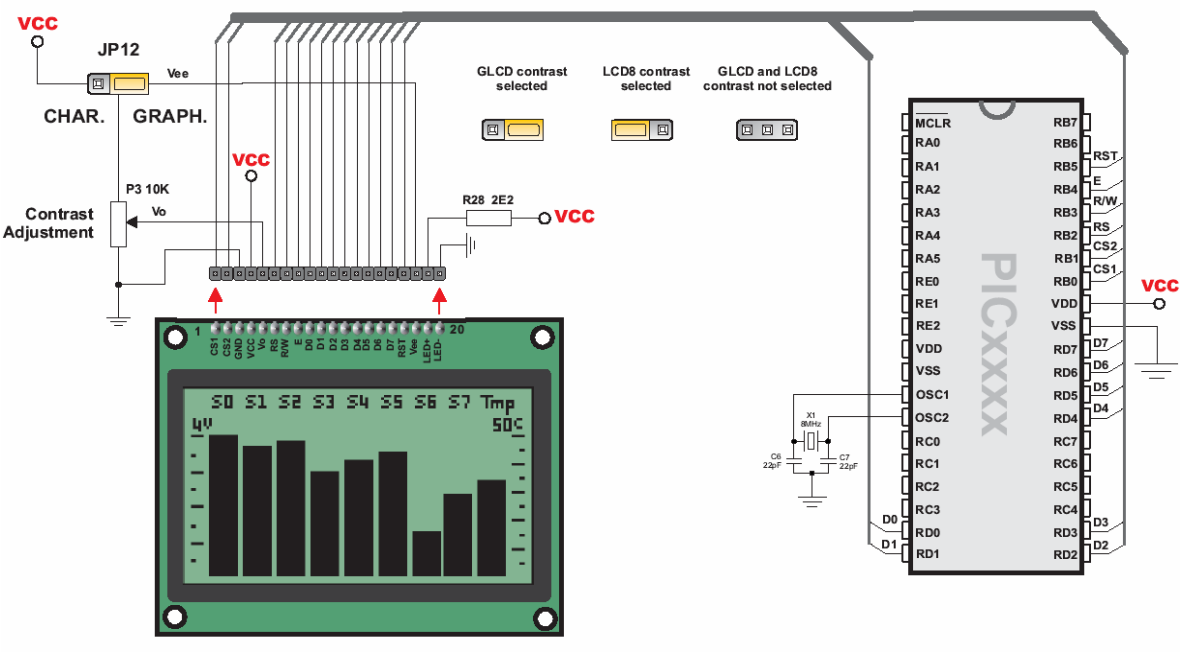


Figura 6.3: Conexión del GLCD

Puesto que no es posible la utilización del driver presente en CCS, y tras definir las conexiones y funciones de cada pin, se desarrolla una librería compatible con el GLCD de la tarjeta EasyPIC4 para su uso en la creación de programas.

El driver “miGLCD.c” creado, se basa en las librerías “HDM64GS12.c” y “GLCD.c” presentes en los drivers de CCS.

Puesto que el módulo gráfico LCD posee el controlador KS0108, se ha utilizado la estructura de la librería “HDM64GS12.c” suministrada por CCS, la cual contiene los drivers necesarios para la utilización de un Hantronix HDM64GS12 con el controlador KS0108. A dicha librería, se le han modificado los pines de E/S e intercambiado los pines CS1 y CS2 para ajustarla a las características del display GLCD de la placa didáctica EasyPIC4. Sin embargo, esta librería posee muy pocas funciones, por lo que se ha completado con funciones de la librería GLCD.c para poder desarrollar ejemplos más elaborados que impliquen la utilización de funciones más complejas.

Por tanto, el driver “miGLCD.c” desarrollado, disponible en el Anexo A del presente documento, es una combinación de ambas librerías. La librería “miGLCD.c” es un driver muy útil para el manejo del display gráfico, ya que

contiene diversas funciones para representar píxeles, líneas, rectángulos, círculos e incluso texto.

6.4. APLICACIONES REALIZADAS

6.4.1. EJEMPLOS DE APLICACIÓN CON EL MÓDULO LCD

En este apartado se detallan distintos programas creados en lenguaje C que reflejan el uso de las funciones presentes en el driver “miLCD.c”. Como se ha comentado anteriormente, el Anexo A de este documento muestra las funciones utilizadas de dicha librería durante el desarrollo del proyecto. El resto de funciones que no se encuentran reflejadas en esta librería (por ser funciones que contiene el propio software), se pueden encontrar detalladas en el manual de CCS C Compiler incluido en el CD adjunto a la memoria del presente proyecto o en la web de CCS [14].

- Fichero “LCD_Micro.c”

Descripción:

A continuación se muestra el código creado para visualizar en el display LCD la palabra “Microcontrolador” y en la parte inferior “>PIC16F877A<”.

Figura 6.4: Imagen del LCD

Código:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                        LCD_Micro.c
////
//// PFC Mª Nieves Robles Botella
//// Utilización del display LCD
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <16F877A.h>            //Incluye la librería para trabajar con el PIC


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)
#include <miLCD.c>

void main() {

    lcd_init();                         //Inicializa el display LCD


    lcd_gotoxy(1,1);                    //Sitúa el cursor en la primera posición de la línea 1
    lcd_putc("Microcontrolador");       //Muestra la palabra “Microcontrolador”

    lcd_gotoxy(3,2);                    //Se sitúa en la posición 3 de la segunda línea
    lcd_putc(">PIC16F877A<");
}
```

Características principales del programa:

En este programa se muestra la utilización de las funciones *lcd_gotoxy* y *lcd_putc*. Dichas funciones son las que se encargan de situar el cursor en una zona del módulo LCD y de escribir los caracteres indicados en la función.

Como se puede observar en el código anterior, antes de utilizar el LCD se ha de inicializar mediante la función *lcd_init* que se encuentra en la librería “miLCD.c”.

- Fichero “LCD_hola.c”

Descripción:

A continuación se muestra el código desarrollado para visualizar la palabra “HOLA” cambiando de posición a lo largo de la primera línea del módulo LCD. La palabra “HOLA” se va desplazando de derecha a izquierda por la primera línea del LCD, mientras que en la segunda línea aparece “*/*/*” sin movimiento.

Figura 6.5: Imagen del LCD

Código:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                        LCD_hola.c
////
//// PFC Mª Nieves Robles Botella
//// Utilización del display LCD
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <16F877A.h>                          //Incluye la librería para trabajar con el PIC

#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)                     //8MHz
#include <miLCD.c>

void main() {
     int i=16;                                //Variable para situar el cursor
     lcd_init();                              //Inicializa el display LCD

     for(i=16;i>=16-3;i--){
          lcd_gotoxy(i,1);
          lcd_putc("HOLA");
          lcd_gotoxy(7,2);
          lcd_putc("*/*/*");
          delay_ms(350);
          lcd_putc("\f");
     }

     while(1){                                //Bucle infinito
          for(i=16-4;i>=1;i--){
               lcd_gotoxy(i,1);
               lcd_putc("HOLA");
               lcd_gotoxy(7,2);
               lcd_putc("*/*/*");
               delay_ms(350);
               lcd_putc("\f");               //Borra el LCD
          }

          if(i==0)                  //Si i es 0 (está al ppio del LCD) mostrar el ppio. de la
          {                         //palabra en la parte dcha. y el final por la parte izqda.
```

```
                lcd_gotoxy(1,1);
                lcd_putc("OLA");
                lcd_gotoxy(16,1);
                lcd_putc("H");
                lcd_gotoxy(7,2);
                lcd_putc("*/*/*");
                delay_ms(350);
                lcd_putc("\f");

                lcd_gotoxy(1,1);
                lcd_putc("LA");
                lcd_gotoxy(15,1);
                lcd_putc("HO");
                lcd_gotoxy(7,2);
                lcd_putc("*/*/*");
                delay_ms(350);
                lcd_putc("\f");

                lcd_gotoxy(1,1);
                lcd_putc("A");
                lcd_gotoxy(14,1);
                lcd_putc("HOL");
                lcd_gotoxy(7,2);
                lcd_putc("*/*/*");
                delay_ms(350);
                lcd_putc("\f");
        }
    }
}
```

Características principales del programa:

Para la correcta visualización de la palabra “HOLA” desplazándose a lo largo del módulo LCD, se ha realizado un bucle infinito en el que se va escribiendo la palabra en una posición distinta dependiendo de la posición anterior. Cuando la palabra llega al extremo izquierdo de la pantalla, i=0, se empieza a mostrar el final de ésta por este lado y por el extremo derecho aparece el resto de la palabra.

En este código además de utilizar las funciones *lcd_gotoxy* y *lcd_putc*, utilizadas anteriormente, se ha hecho uso de la función *delay_ms* para mantener la palabra 350 ms. en el display antes de ser borrada.

6.4.2. EJEMPLOS DE APLICACIÓN CON EL MÓDULO GLCD

Con el fin de simular programas utilizando el display gráfico LCD del entrenador EasyPIC4, se han desarrollado los ficheros “cronometro.c” y “tension_RA2.c”.

- Fichero “cronometro.c”

Los cronómetros son relojes de alta precisión muy utilizados tanto por navegantes para determinar la longitud geográfica y calcular su posición en alta mar, como en competiciones deportivas, midiendo fracciones de tiempo normalmente cortos y con gran precisión. También se aplican con frecuencia dichos dispositivos tanto en ciencia como en tecnología.

Descripción:

En este código se muestra la realización de un cronómetro con una precisión de décimas de segundo.

Una vez grabado el programa en la memoria del microcontrolador, aparece la pantalla principal (ver Figura 6.6), donde se muestra tanto las horas, como los minutos, segundos y décimas de segundo con valor cero, además de una imagen que simula una persona corriendo en la parte derecha de la pantalla.

Figura 6.6: Imagen del GLCD.

El cronómetro empieza a contar desde cero y la imagen (simulación de la persona), comienza a moverse cuando se presiona el pulsador RC0. La temporización se detiene al pulsar el botón RC1. Una vez parado el cronómetro se puede resetear, comenzando desde cero, mediante RC2 o bien seguir con

la temporización desde el punto donde se detuvo pulsando RC0. En la Figura 6.7 se puede ver una imagen del cronómetro.

Figura 6.7: Imagen del GLCD

Cuando el valor de temporización llega a 99 horas, 59 minutos, 59 segundos y 9 décimas aparece la pantalla de la Figura 6.8, en la cual, se muestra el mensaje “Pulse C0” para reiniciar la temporización.

Figura 6.8: Imagen del GLCD

Código:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                    Cronómetro.c
////
//// PFC Mª Nieves Robles Botella
//// Utilización del display GLCD
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <16F877A.h>                                    //Incluye la librería 16F877A
```

```
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)

#include <miGLCD.c>

#define ONOFF0 PIN_C0
#define ONOFF1 PIN_C1
#define ONOFF2 PIN_C2

void displayMu01()                              //primera imagen (Simulación de la persona)
{
    //Cabeza
    glcd_rect(103,32, 104, 34,ON,ON);
    glcd_pixel(102,33 , ON);
    glcd_pixel(105,33 , ON);
    //Cuerpo
    glcd_rect(104,35, 105, 37,ON,ON);
    //Brazo
    glcd_pixel(103,38 , ON);
    glcd_pixel(102,39 , ON);
    glcd_pixel(105,38 , ON);
    //Brazo
    glcd_line(106, 36, 108, 36, ON);
    glcd_pixel(109,37 , ON);
    //Recta del cuerpo vertical
    glcd_line(106, 37, 106, 42, ON);
    glcd_line(107, 39, 107, 40, ON);
    glcd_line(105, 43, 104, 44, ON);
    glcd_line(104, 45, 102, 45, ON);
    glcd_line(108, 41, 109, 42, ON);
    glcd_pixel(110,42 , ON);
    glcd_pixel(110,46 , ON);
    glcd_line(111, 43, 111, 45, ON);
    glcd_pixel(105,44 , ON);
}

void displayMu02()                              //segunda imágen
{
    //Cabeza
    glcd_rect(103,32, 104, 34,ON,ON);
    glcd_pixel(102,33 , ON);
    glcd_pixel(105,33 , ON);
    //Cuerpo
    glcd_rect(104,35, 105, 38,ON,ON);
    //Brazo
    glcd_pixel(103,39 , ON);
    //Brazo
    glcd_line(106, 36, 107, 36, ON);
    glcd_line(109, 38, 109, 39, ON);
    glcd_pixel(108,37 , ON);
    //Recta del cuerpo vertical
    glcd_line(106, 38, 106, 40, ON);
    glcd_line(107, 39, 107, 40, ON);
    glcd_line(108, 41, 108, 42, ON);
    glcd_line(109, 42, 111, 42, ON);
    glcd_pixel(110,44 , ON);
    glcd_pixel(111,43 , ON);
    glcd_line(105, 41, 104, 42, ON);
    glcd_line(104, 43, 106, 45, ON);
    glcd_line(103, 45, 105, 45, ON);
```

```
}

void displayMu03()                          //tercera imagen
{
    //Cabeza
    glcd_rect(103,32, 104, 34,ON,ON);
    glcd_pixel(102,33 , ON);
    glcd_pixel(105,33 , ON);
    //Cuerpo
    glcd_rect(104,35, 105, 37,ON,ON);
    //Brazo
    glcd_pixel(105,38 , ON);
    glcd_pixel(105,39 , ON);
    glcd_pixel(104,39 , ON);
    //Brazo
    glcd_line(108, 38, 108, 39, ON);
    //Recta del cuerpo vertical
    glcd_line(106, 36, 106, 44, ON);
    glcd_line(107, 37, 106, 44, ON);
    glcd_line(108, 42, 106, 44, ON);
    glcd_line(107, 39, 108, 46, ON);
    glcd_line(105, 42, 104, 44, ON);
    glcd_line(105, 44, 105, 45, ON);
    glcd_pixel(104,45 , ON);
    //Pie
    glcd_line(106, 46, 110, 46, ON);
    glcd_pixel(110,45 , ON);
    glcd_pixel(109,44 , ON);
}

void displayMu04()                          //cuarta imagen
{
    //Cabeza
    glcd_rect(102,32, 103, 34,ON,ON);
    glcd_pixel(101,33 , ON);
    glcd_pixel(104,33 , ON);
    //Cuerpo
    glcd_rect(103,35, 104, 37,ON,ON);
    //Recta vertical
    glcd_line(105, 36, 105, 42, ON);
    glcd_line(106, 39, 106, 40, ON);
    //Brazo
    glcd_line(101, 39, 102, 38, ON);
    glcd_pixel(104,38 , ON);
    //Brazo
    glcd_line(106, 36, 107, 36, ON);
    glcd_line(108, 37, 109, 38, ON);
    //Pierna
    glcd_line(107,41, 110, 44,ON);
    glcd_line(109,42, 110, 43,ON);
    glcd_line(109,46, 110, 45,ON);
    //Pie
    glcd_rect(101,44, 103, 45,ON,ON);
    glcd_rect(103,43, 104, 44,ON,ON);
}

void borrarMu()
{
    glcd_rect(100,30, 115, 50,ON,OFF);
}
```

```
void main() {
    int i,j,k,h=0;
    char segT[]="00:00:0 ";
    char  a[]="00:00:0";
    char cer0[]="0";
    char b[]="00";
    char c[]="00";
    char cer00[]="00";
    char d[]="00";
    char ho[]="horas";
    char dng[]="Pulse C0";

    glcd_init(ON);                          //Se inicializa el GLCD

    glcd_rect(1,1,127,63,OFF,ON);           //Dibuja un rectángulo
    delay_ms (500);                         //Espera 500ms

    glcd_bar(2,5,127,5,5,ON);               //Dibuja una barra horizontal de grosor 5 píx.
    glcd_bar(2,59,127,59,5,ON);             //Dibuja una barra horizontal de grosor 5 píx.
    delay_ms(2000);

while(1){
    glcd_text57(10,21,cer00,2,ON);          //Aparece el número 00
    glcd_text57(7,11,ho,1,ON);              //Muestra la palabra horas
    glcd_text57(40,25,segT,1,ON);           //Aparece 00:00:00

    displayMu01();                          //Muestra el primer muñeco


    if (input (ONOFF0))                     //Cuando se pulsa el RC0 empieza a contar
    {
        while(1){
            for(k=1;k<=60;k++)          //Minutos
            {
                for(j=1;j<=60;j++)          //Segundos
                {
                    for(i=0;i<10;i++)           //Décimas
                    {
                        sprintf(a, "%d",i );        // Convierte el entero i en texto

                        glcd_rect(76,25, 87, 35, ON, OFF);      // Borra el numero anterior
                        glcd_text57(76,25,a,1,ON);              //Muestra las décimas
                        delay_us(19321);

                        if (i==0)
                        {
                            borrarMu();                         //Borra la imagen anterior
                            displayMu01();                      //Muestra la primera imagen
                        }
                        if (i==3)
                        {
                            borrarMu();                         //Borra la imagen anterior
                            displayMu02();                      //Muestra la segunda imagen
                        }

                        if (i==6)
                        {
                            borrarMu();                         //Borra la imagen anterior
                            displayMu03();                      //Muestra la tercera imagen
```

```
                }
                if (i==9)
                {
                    borrarMu();                         //Borra la imagen anterior
                    displayMu04();                      //Muestra la cuarto imagen
                }
                if(input (ONOFF1))                      //Si se pulsa RC1
                {
                    while (!input (ONOFF0))  //Mientras no se haya pulsado RC0 se espera
                    {
                        if(input (ONOFF2))     //Si se pulsa RC2 reinicia la cuenta
                        {
                            glcd_rect(40,25, 87, 35, ON, OFF);
                            glcd_text57(40,25,segT,1,ON);
                            glcd_rect(10,21, 32, 41, ON, OFF);
                            glcd_text57(10,21,cer00,2,ON);
                            i=0;
                            j=1;
                            k=1;
                            h=0;
                        }
                    }
                }

            }   //Cierra el for con i

            if(j<=9)                            //Si el número es de una unidad
            {
                sprintf(b, "%d",j );                        // Convierte el entero j en texto
                glcd_rect(58,25, 69, 35, ON, OFF);          // Borra el número anterior
                glcd_text57(58,25,cer0,1,ON);       //Muestra el 0 en la decena del número
                glcd_text57(64,25,b,1,ON);          //Muestra los segundos

            } else{

                sprintf(b, "%d",j );                        // Convierte el entero j en texto
                glcd_rect(58,25, 69, 35, ON, OFF);          // Borra el número anterior
                glcd_text57(58,25,b,1,ON);                  //Muestra los segundos
            }

        }//Cierra el for de j

            glcd_rect(58,25, 69, 35, ON, OFF);              //Borra los segundos
            glcd_text57(58,25,cer00,1,ON);

        if(k<=9)
        {
            sprintf(c, "%d",k );                        // Convierte el entero k en texto
        glcd_rect(40,25, 51, 35, ON, OFF);
            glcd_text57(40,25,cer0,1,ON);
            glcd_text57(46,25,c,1,ON);                  //Muestra los minutos

        } else{
            sprintf(c, "%d",k );                        // Convierte el entero k en texto
            glcd_rect(40,25, 51, 35, ON, OFF);          // Borra el numero anterior
            glcd_text57(40,25,c,1,ON);                  //Muestra los minutos

        }//Cierra el else

    }//Cierra el for de k
```

```
            glcd_rect(40,25, 51, 35, ON, OFF);              // Borra el número anterior
            glcd_text57(40,25,cer00,1,ON);

            h=h+1;                                          //Se incrementa en una unidad las horas

            if(h<=9)
            {
                sprintf(d, "%d",h );                        // Convierte el entero h en texto

                    glcd_rect(10,21, 35, 41, ON, OFF);      // Borra el número anterior
                    glcd_text57(10,21,cer0,2,ON);
                    glcd_text57(21,21,d,2,ON);

            } else{
            if(h==100)
                {
                    glcd_text57(35,40,dng,1,ON);    //Aparece el texto Pulse C0
                    while (!input (ONOFF0))         //Mientras no se haya pulsado C0 espera
                    {
                        glcd_rect(40,25, 87, 35, ON, OFF);
                        glcd_text57(40,25,segT,1,ON);
                        glcd_rect(10,21, 32, 41, ON, OFF);
                        glcd_text57(10,21,cer00,2,ON);
                        i=1;
                        j=1;
                        k=1;
                        h=0;

                    delay_us(3000);     //Parpadea
                    }
                glcd_rect(34,34,85,47,ON,OFF);                      //Borra el texto Pulse C0
                }
            if(h==0)
                {   glcd_rect(10,21, 35, 41, ON, OFF);
                    glcd_text57(10,21,cer00,2,ON);
                }
            else{
                sprintf(d, "%d",h );                        // Convierte el entero h en texto
                glcd_rect(10,21, 35, 41, ON, OFF);          // Borra el número anterior
                glcd_text57(10,21,d,2,ON);
             }
        }
        }//Cierra el while

    }

}

}
```

Características principales del programa:

La principal característica de este código es su gran precisión a la hora medir tiempos. En el Anexo B se realiza un estudio temporal de dicha aplilcación donde se recoge el error cometido.

Con el fin de calcular el valor que se ha de pasar a la función *delay_us* para medir segundos, se ha empleado la herramienta “stopwatch” que presenta el entorno MPLAB. Dicha herramienta es muy útil a la hora de realizar una medición de tiempos de manera rápida y sencilla, puesto que solamente es necesario indicar el fragmento de código que se desea medir (mediante breakpoints) y la frecuencia del microcontrolador, obteniendo el tiempo invertido en ello.

A continuación se muestra los valores de temporización obtenidos (en un segundo) en dos de las iteraciones que realiza el programa cuando el delay es 19321 µs.

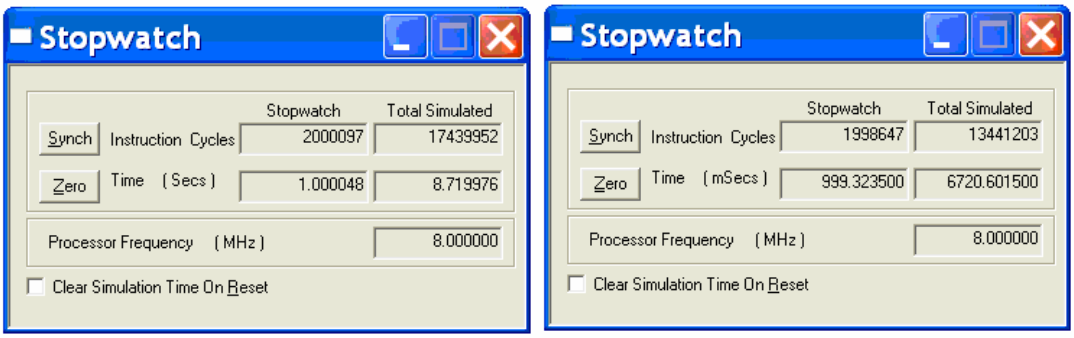


Figura 6.9: Duración del bucle que cuenta las décimas de segundo

Como se puede observar en la Figura 6.9, los tiempos de iteración, además de ser distintos, no coinciden exactamente con un 1 segundo. Esto se debe a que no todas las funciones que se ejecutan presentan el mismo periodo de duración.

Una de las funciones que varía dependiendo del momento de ejecución del programa es *glcd_text57*. Dicha función realiza la escritura de texto en el módulo GLCD, y por tanto dependiendo del número (dígito) que se escriba puede precisar más o menos números de píxeles para representar dicho número, causa por la cual existe esa variación de tiempo. En el Anexo B se detalla la duración de dicha función en la representación de distintos números.

Otra función cuya duración depende de los píxeles a dibujar y del momento de ejecución del programa, es la de dibujar la figura de la persona. Para realizar la simulación de una persona corriendo, se han utilizado las funciones definidas al

principio del código del fichero “cronómetro.c”: displayMu01, displayMu02, displayMu03, displayMu04, en las cuales, se ha representado mediante píxeles la figura de un cuerpo humano en cuatro posiciones distintas. Alternado las imágenes a lo largo del tiempo se consigue el efecto óptico deseado (imagen de una persona corriendo).

A continuación se muestra la duración de las funciones de dibujar y borrar la imagen.

Función	Duración
displayMu01	10.81 ms
displayMu02	11.12 ms
displayMu03	15.40 ms
displayMu04	13.05 ms
borrarMu	82.91 ms

Tabla 6.3: Funciones dibujar persona y tiempos

Además de que cada función presenta una duración diferente, se ha de tener en cuenta que dicha parte de código no se ejecuta siempre ya que sólo se dibujan las figuras en las décimas 0, 3, 6 y 9 (cuando i=0, i=3. i=6 ó i=9), por lo que la duración de las décimas de segundos varían.

Para solventar dicho problema y que todas las décimas de segundo presentasen la misma duración, se modificó el código haciendo que en cada décima se dibujase una imagen. El problema que presentaba este programa es que incluso eliminado el *delay*, la duración de los segundos era mayor a la unidad, por lo cual, se optó por dejar el código original que no presenta retraso, ya que el error que cometido es pequeño.

La información del error cometido a causa de la diferencia de duración de las distintas funciones, se encuentra reflejada en el Anexo B del presente documento.

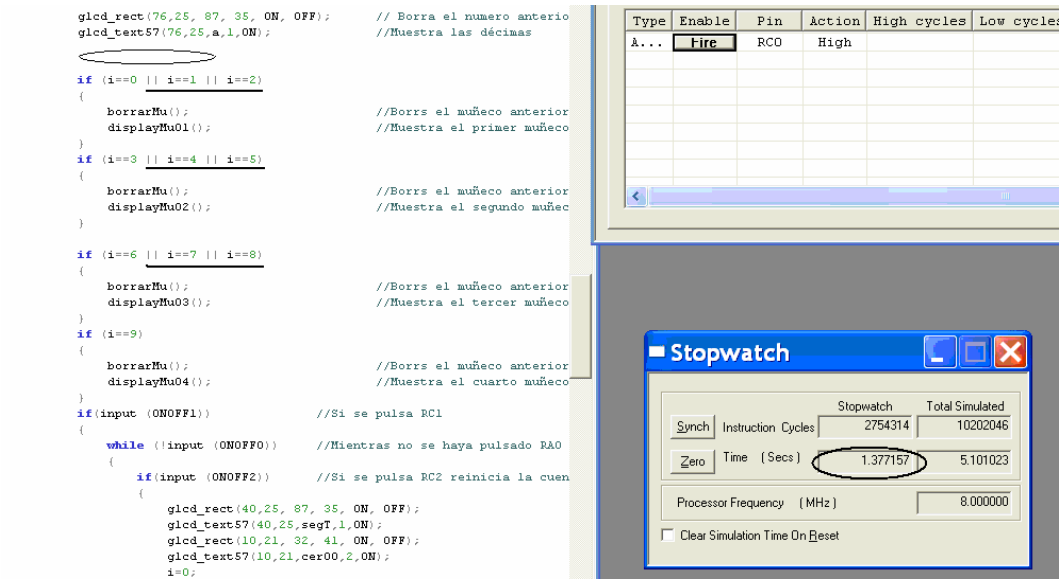


Figura 6.10: Duración de un segundo al dibujar la imagen en todas las décimas

- Fichero “tension_RA2.c”

Descripción:

El código desarrollado a continuación, muestra la lectura en el módulo GLCD de la tensión existente en el pin RA2, la cual es modificada utilizando el potenciómetro P1.

Una vez volcado el programa sobre el PIC16F877A, aparece una pantalla de inicio con el índice de las opciones que se pueden elegir.

Figura 6.11: Imagen del GLCD. Menú

Cuando se pulsa RC1, opción de visualización de la tensión en formato barra, aparece el valor de la tensión existente en RA2, comprendido entre 0V y 5V, y una barra horizontal que aumenta o disminuye proporcionalmente a dicha tensión. La Figura 6.12 muestra la apariencia del GLCD.

Figura 6.12: Imagen del GLCD. Opción Formato Barra

Para salir de esta pantalla se ha de pulsar RC0 volviendo de nuevo al menú.

Si por el contrario, en lugar de visualizar la barra horizontal, se desea contemplar una representación de la tensión en función del tiempo, se ha de pulsar RC2. Mediante esta opción se muestra una pantalla donde se representa en función del tiempo el valor de la tensión, en una gráfica comprendida entre 0V y 5V. Mediante esta función se puede observar el valor que poseía RA2 unos instantes previos a su modificación, ya que queda representado en la gráfica, tal y como aparece en la Figura 6.13.

Figura 6.13: Imagen del GLCD. Opción Formato Gráfica

Código:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                    Tension_RA2.c
////
//// PFC Mª Nieves Robles Botella
//// Utilización del display GLCD
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#include <16F877A.h>
#include <miGLCD.c>


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)

void indice()
{
     char  op1[] = "-BARRA: RC1";
     char  op2[] = "-OSCILOSCOPIO: RC2";
     glcd_text57(2,38,op1,1,ON);
     glcd_text57(2,48,op2,1,ON);
     glcd_rect(81, 19, 115, 26, ON, OFF);                    // Borra la tensión anterior
}

void main() {
                                                             //Pantalla inicial
     char text[]= "TeNsIoN eN RA2";
     char ex[]="Salir RCO";
     char cer[]="0";
     char cin[]="5";
     char voltage[9];
     float tn;
     int adc=0, adcAnt=0, i=1, pulsC0=0;
     char  voltText[] = "V";

     setup_adc_ports(ALL_ANALOG);          //Establece todos los pines adc como analógicos
     setup_adc(ADC_CLOCK_INTERNAL);        //Habilita el módulo A/D
     set_adc_channel(2);                   //En la instrucción de read_adc lee el canal 2 RA2

     glcd_init(ON);                                          //Inicializa el GLCD

     glcd_rect(1,1,126,33,OFF,ON);                           //Dibuja dos rectángulos
     delay_ms(500);
     glcd_rect(3,3,124,31,OFF,ON);
     delay_ms (1500);
     glcd_text57(10,10,text,1,ON);                           //Muestra el texto almacenado en text
     glcd_text57(10,20,ex,1,ON);

     indice();

     while(1){
       if(input(PIN_C1))          //Si se pulsa RC1, muestra barra proporcional a la tensión
       {
            glcd_rect(1,63,126,35, ON, OFF);                 //Borra parte de la pantalla
            glcd_text57(106, 19, voltText, 1, ON);           // Muestra la letra "V"

            while(!input(PIN_C0)) {                          //Mientras no se haya pulsado RC0
```

```
            adc = read_adc();                           // Lee el valor ADC
            tn=(float)adc * .01960784;
            sprintf(voltage, "%f", tn );                // Convierte el entero adc en texto
            voltage[4] = '\0';                          // Muestra sólo 3 dígitos
            glcd_text57(81, 19, voltage, 1, ON);        // Muestra la tensión actual
            glcd_text57(0,53,cer,1,ON);                 //Muestra el número 0
            glcd_text57(122,53,cin,1,ON);               //Muestra el número 5
            glcd_rect(0,44,127,51,OFF,ON);
            glcd_pixel(128/2,45,ON);
            glcd_pixel(128/2,50,ON);
            glcd_rect(0, 45, adc/2, 50, ON, ON);        // Muestra la barra actual

            if (adc!=adcAnt)
            {
                glcd_rect(81, 19, 105, 26, ON, OFF);            //Borra la tensión anterior
                glcd_text57(81, 19, voltage, 1, ON);            // Muestra la tensión actual
                glcd_rect(adc/2, 45, adcAnt/2, 50, ON, OFF);    // Borra la barra anterior
                glcd_rect(0, 50, adc/2, 45, ON, ON);            // Muestra la barra actual
                adcAnt = adc;

            }

        }//fin while
        if(input(PIN_C0))
        {
            glcd_rect(0,63,127,35, ON, OFF);
            indice();
        }
    }

    if(input (PIN_C2))                          //Si se pulsa RC2, muestra "osciloscopio"
    {
        pulsC0=0;
        glcd_rect(0,63,126,35, ON, OFF);

        while(!input(PIN_C0)) {                 //Mientras no se haya pulsado RC0

            glcd_rect(0,63,127,35, ON, OFF);            //Borra la pantalla
            //Muestra los ejes y los valores 0 y 5
            glcd_line(8,63,8,35, ON);
            glcd_line(8,63,120,63, ON);
            glcd_text57(0,56,cer,1,ON);
            glcd_pixel(7,63,ON);
            glcd_text57(0,35,cin,1,ON);
            glcd_pixel(7,35,ON);

            while(!input(PIN_C0)) {                 //Mientras no se haya pulsado RC0
                adc = read_adc();                   //Lee el valor ADC
                tn=(float)adc * .01960784;
                sprintf(voltage, "%f",tn );         //Convierte el entero adc en texto
                voltage[4] = '\0';                  //Muestra sólo 3 dígitos
                glcd_pixel(i+8,(int)(63-28/(5/tn)),ON);     //Dibuja el pto correspondiente
                glcd_rect(81, 19, 105, 26, ON, OFF);        //Borra la tensión anterior
                glcd_text57(81, 19, voltage, 1, ON);        //Muestra la tensión actual
                glcd_text57(106, 19, voltText, 1, ON);      //Muestra la letra "V"

                if(i==112)              //Si i=112, está toda la pantalla dibujada
                {
                    i=0;                //Pone a 0 i, numero de píxeles dibujados
                    glcd_rect(9,35,120,62, ON, OFF);        //Borra el área de dibujo
```

```
                    }
                    i++;
                    if(input(PIN_C0))                    //Si se pulsa RC0
                    {
                        pulsC0=1;                        //Pone a uno la variable pulsC0
                    }//fin if

                }//fin while

            }
            if(pulsC0)
            {
                glcd_rect(0,63,127,35, ON, OFF);
                indice();
                i=0;                    //la próxima vez que se acceda empiece a dibujar desde 0
            }//fin if
        }//fin while
    }
}
```

Características principales del programa:

Como se puede apreciar, el código del programa consta de un bucle infinito que se encarga de leer constantemente los pines RC1 y RC2 con el fin de realizar una función u otra.

Tanto en el código para mostrar una barra como con el código para mostrar un osciloscopio, se está leyendo constantemente el valor analógico de tensión de RA2 mediante la función *read_adc*, que posteriormente se convierte en formato texto y se muestra en el display mediante las funciones *sprintf* y *glcd_text57*.

Como se puede observar en la aplicación desarrollada, la tensión que se muestra en el display no es realmente la que se lee directamente mediante la función *read_adc*, ya que con esta función el valor máximo que se obtiene es 254, que es lo que corresponde a 5V. Por ello se utiliza el factor 0.01960784, para mostrar en el módulo GLCD un valor de tensión comprendido entre 0V y 5V.

```
tn=(float)adc * .01960784
```

En el CD ajunto a la memoria de este Proyecto Fin de Carrera, se incluyen tanto las aplicaciones mostradas anteriormente, como los programas de lectura de tensión de RA2 desarrollados para visualizar de diferentes maneras la tensón que se lee en ese pin, éstos también se encuentran disponibles en el

Anexo C del presente documento. Dichos programas son la base del programa “tensión_RA2.c”.

CAPÍTULO 7. CIRCUITO VIRTUAL DEL ENTRENADOR EASYPIC4

7.1. INTRODUCCIÓN

Este capítulo se basa en la creación, en el programa de simulación PROTEUS, de un circuito virtual que representa el funcionamiento del sistema de desarrollo EasyPIC4.

La creación de este circuito virtual, con las mismas características que tiene el entrenador, supone una herramienta muy útil a la hora de visualizar el comportamiento de los distintos componentes presentes en la placa EasyPIC4.

Esto permite simular programas cargados sobre un microcontrolador virtual sin necesidad de tener el sistema físico presente, haciendo de ello una herramienta adecuada para utilizar desde cualquier ordenador. Además, la utilización de un sistema de estas características fomenta la creatividad, la motivación, el esfuerzo personal así como el trabajo fuera de las instalaciones de los laboratorios, creando una mayor implicación por parte de los estudiantes.

7.2. ENTORNO DE SIMULACIÓN

Con el fin de realizar un circuito virtual que simule el funcionamiento de la placa didáctica de evaluación EasyPIC4, se ha utilizado el entorno de desarrollo

PROTEUS 6 PROFESSIONAL [17]. Este entorno, es un programa de simulación que permite crear diversos circuitos virtuales de forma sencilla, con una interfaz de usuario moderna y atractiva.

Dicho entorno de simulación ha sido utilizado debido a sus numerosas ventajas frente a la utilización de otras herramientas e incluso frente a los propios circuitos electrónicos. Ya que la utilización de un circuito virtual que recrea el funcionamiento de un sistema electrónico real, es mucho más cómodo y funcional que el propio sistema físico.

Laboratorio de electrónica	Laboratorio virtual PROTEUS
Los equipos de medida, simulación y prototipos necesitan laboratorios dedicados en exclusividad. Donde equipar dichos laboratorios con simuladores de circuitos tiene un coste alto.	Puede ser utilizado en salas de ordenadores de usos múltiples situadas en cualquier lugar del campus.
Equipar los laboratorios con simuladores de circuitos tiene un coste alto.	Coste de la licencia.
Los instrumentos de medida como osciloscopios, analizadores lógicos y generadores de patrones son muy caros.	Incluye 11 instrumentos virtuales de serie.
El número de personas que pueden realizar prácticas a la vez se ve limitado al número de equipos de hardware disponibles.	No existe limitación en el número de usuarios.
Modificar o actualizar los quipos para realizar prácticas es una tarea laboriosa y costosa.	Realizar prácticas nuevas con diseños diferentes e innovadores es una tarea sencilla.
Los errores pueden generar daños en los equipos con los consiguientes gastos adicionales en compra de hardware	Los componentes virtuales son indestructibles
Sólo se puede trabajar con los equipos dentro del laboratorio	Se puede trabajar desde cualquier lugar donde se tenga instalado el programa.
Pérdida de tiempo cableando circuitos, para poder comprobar el funcionamiento del programa desarrollado.	Los circuitos pueden ser realizados previamente, de forma rápida y sencilla.

Tabla 7.1: Laboratorio de electrónica frente a laboratorio virtual

La elaboración del esquema del circuito electrónico de la placa didáctica de evaluación EasyPIC4 se ha realizado utilizando el entorno ISIS de PROTEUS. Como ya se ha comentado anteriormente, el programa PROTEUS, posee entre sus utilidades una variada instrumentación virtual que facilita la creación y el análisis de circuitos, así como la simulación mediante el módulo ISIS de PROTEUS.

El primer paso realizado para simular cualquier circuito electrónico, y en concreto el circuito virtual EasyPIC4, es diseñar el hardware sobre el que se hace “correr” el programa o software desarrollado. Para ello, se han elegido de las librerías de componentes que posee el entorno ISIS, los diversos elementos involucrados en el entrenador EasyPIC4, tales como:

- Resistencias
- LEDs
- PIC16F877A
- Interruptores
- Pulsadores
- LCD
- GLCD
- Displays de 7 segmentos
- Etc.

Una vez colocados todos los componentes en la zona de edición, se realiza el proceso de unión de los distintos elementos. Las conexiones entre los terminales de los componentes dispuestos en la hoja de trabajo se han realizado mediante hilos (*wire*) y uniones (*junctiondot*), y mediante buses utilizando etiquetas (*label*).

Como se puede apreciar en las Figuras 7.9 y 7.11, la utilización de buses en el diseño de conexionado de los distintos elementos facilita la legibilidad, además de que el circuito virtual creado ocupa un menor espacio.

Una vez finalizada la tarea de creación del circuito, existen diversos caminos a seguir con el fin de simular un programa:

- Generar el fichero hexadecimal en PROTEUS: Si se opta por crear el .HEX desde el propio PROTEUS, se ha de definir el entorno en el que se encuentra el programa que se va a escribir. Para ello se ha de elegir en la ventana de *Define Code Generation Tools*, el ensamblador o compilador que se utilizará para generar el fichero ejecutable a partir del fichero fuente.

  El fichero fuente se puede crear tanto en el editor que viene por defecto como en otro:

  - Al utilizar el editor se puede escribir y ensamblar el código del programa sin necesidad de salir del entorno PROTEUS.
  - Otra opción sería la de escribir el programa en otro entorno y con otro editor e indicar en PROTEUS la herramienta de ensamblado a utilizar.

- Generar el fichero hexadecimal en otro programa: esta opción trata simplemente de la creación del fichero ejecutable en otro programa independientemente de PROTEUS.

Una vez generados los ficheros ejecutables, se asocia al microcontrolador el programa .HEX, para ello se selecciona el microcontrolador para editarlo mostrando el siguiente cuadro de diálogo de la Figura 7.1.

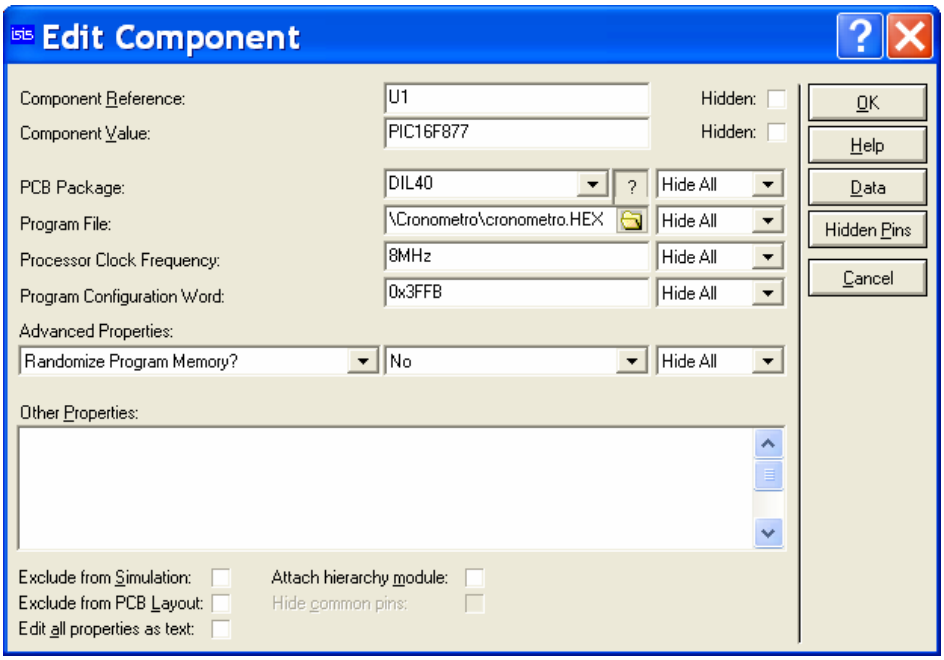


Figura 7.1: Cuadro de diálogo del PIC16F877

Sobre este cuadro de diálogo, se indica el programa que se desea ejecutar en el microcontrolador, la frecuencia de reloj, palabra de configuración, etc.

Llegados a este punto, ya se puede pasar a la simulación y animación del circuito. Mediante los botones de control presentes en la barra de animación se controla el proceso de simulación.

Figura 7.2: Barra de animación

7.3. CREACIÓN DEL HARDWARE DEL CIRCUITO VIRTUAL EASYPIC4

En el siguiente apartado se describe el proceso seguido para la creación del hardware del sistema virtual EasyPIC4. Se muestra los dos principales esquemas realizados para recrear el funcionamiento de la placa didáctica de evaluación:

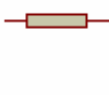
- entrenador_EasyPIC4_1 (ver Figura 7.9).
- entrenador_EasyPIC4_2 (ver Figura 7.11).

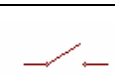
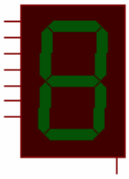
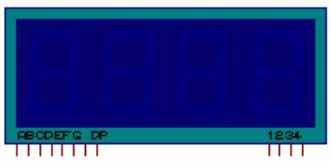
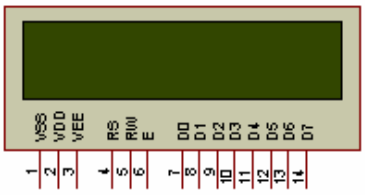
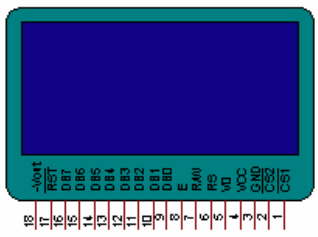
7.3.1. COMPONENTES UTILIZADOS

A continuación se muestran los elementos utilizados para la creación del circuito virtual entrenador_EasyPIC4_1 y entrenador_EasyPIC4_2, así como las librerías de las cuales se han obtenido dichos componentes.

La elección de dichos dispositivos se ha realizado en base a la similitud de los elementos virtuales, de los que contaban las extensas y diversas librerías de ISIS, con los componentes de la placa didáctica EasyPIC4.

En la siguiente tabla se puede observar los distintos elementos que componen el sistema virtual EasyPIC4.

Dispositivo	Librería	Descripción	Imagen
MINRES10K	RESISTORS	10K 0.6W Resistor (Maplin Stock Code=M10K)	
2N2369	BIPOLAR	NPN Transistor	

CLASX2-100N	CAPACITORS	100n Class X2 Metallised Polypropene Film Capacitor	
7414	74STD	Hex.Schmitt-Triggered Inverters	
SWITCH	ACTIVE	Interactive SPST Switch (Latched Action)	
BUTTON	ACTIVE	SPST Push Button	
LED-YELLOW	ACTIVE	Animated LED model (Yellow)	
7SEG-COM-CAT-GRN	DISPLAY	Green,7-Segment common Cathode	
7SEG-MPX4-CC-BLUE	DISPLAY	Blue, 4 Digit, 7-Segment Cathode Display	
LM016L	DISPLAY	16x2 Alphanumeric LCD	
AMPIRE 128X64	DISPLAY	128X64 Graphical LCD with KS0108 controllers	

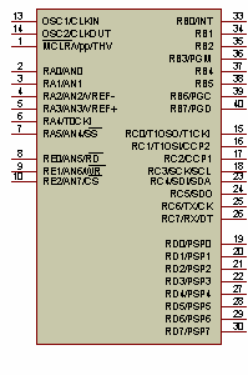
PIC16F877	MICRO	PIC16 Microcontroller	

Tabla 7.2: Dispositivos que componen el circuito virtual EasyPIC4.

Tanto estos elementos como cualquier otro dispositivo que se desee incluir en el área gráfica, se busca utilizando el cuadro de diálogo “Pick Devices” (ver Figura 7.3), herramienta de gran utilidad que permite localizar y clasificar los dispositivos con rapidez.

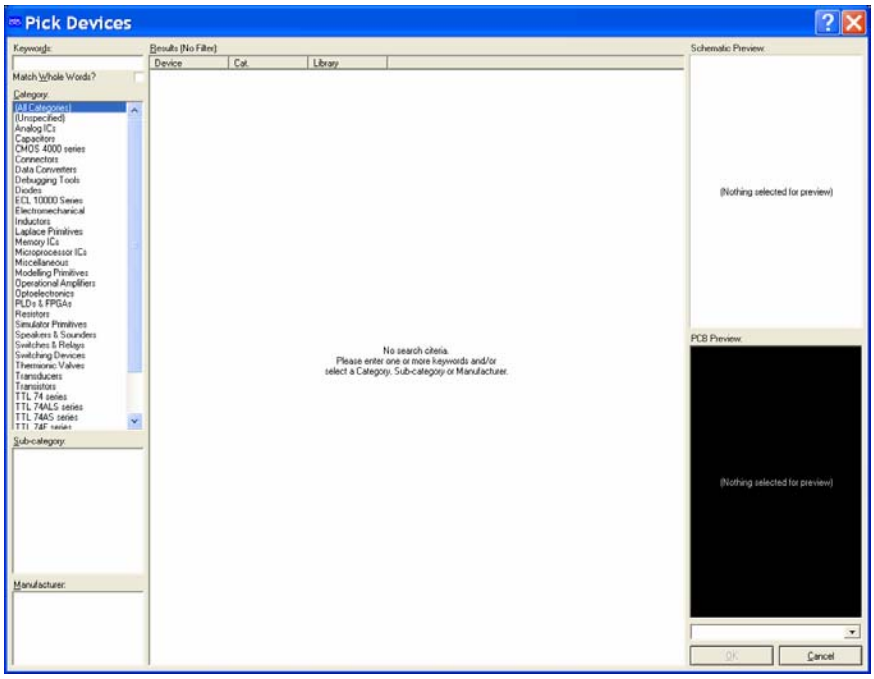


Figura 7.3: Cuadro de diálogo “Pick Devices”, correspondiente a la búsqueda de dispositivos

7.3.2. CIRCUITO VIRTUAL EASYPIC4 CON ELEMENTOS ANALÓGICOS

En un primer momento se comenzó a realizar un circuito que se asemejase, en la medida de lo posible, tanto físicamente (imagen y distribución de los componentes) como en lo que al comportamiento se refiere, a la placa didáctica EasyPIC4.

El primer componente colocado fue el microcontrolador PIC16F877, ya que la herramienta ISIS, no tiene incluido en sus librerías el PIC16F877A. La utilización del PIC16F877 en lugar del PIC16F877A no supone problema alguno, puesto que la única diferencia existente entre este microcontrolador y el del entrenador EasyPIC4 es la presencia, en este último, del módulo de comparación analógica y referencia de tensión interna.

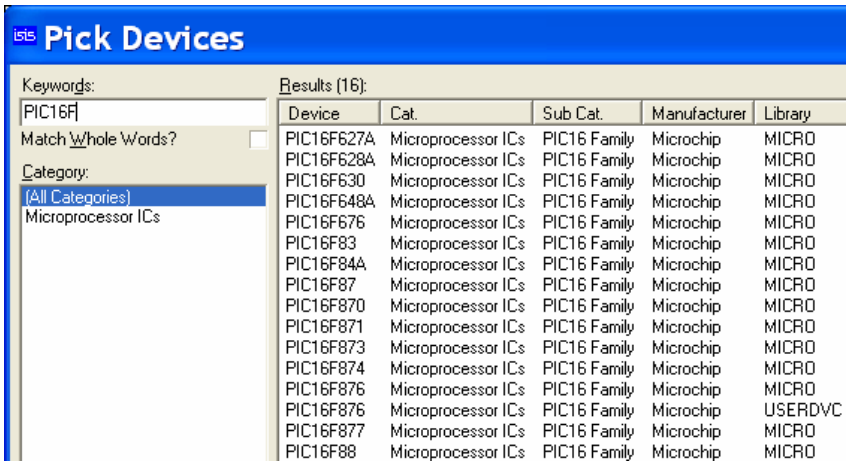


Figura 7.4: PICs PIC16F disponibles para la realización del circuito

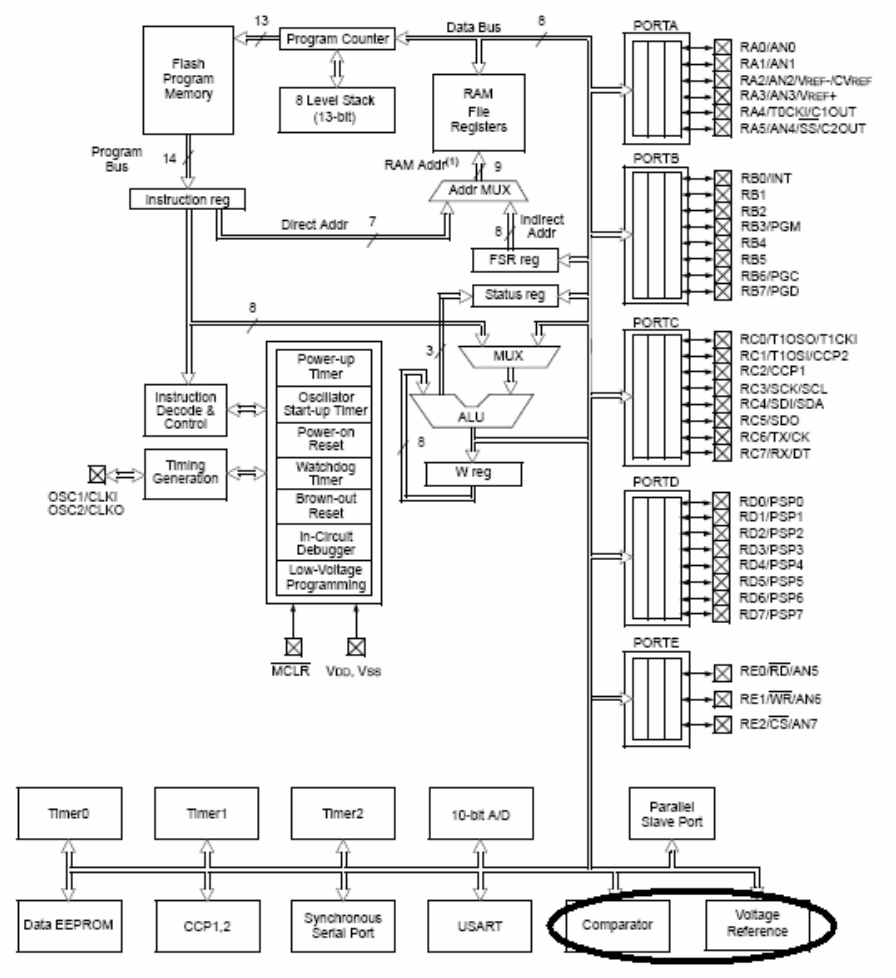


Figura 7.5: Diferencia entre el PIC16F877 y la versión avanzada PIC16F877A

A continuación se incorporó el circuito de reset, que consta de un pulsador, una resistencia de 10KΩ y un condensador de 100nF, y los pulsadores correspondientes a la pines de entrada de los puertos del microcontrolador.

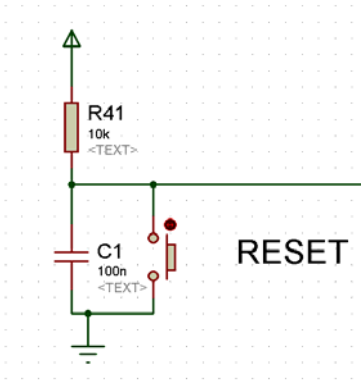


Figura 7.6: Circuito de reset

Posteriormente se añadieron los LEDs, que se conectarán a los puertos del microcontrolador para mostrar el estado digital de los pines. En este circuito, entrenador_EasyPIC4_1, los LEDs utilizados fueron los analógicos LED-YELLOW donde la única diferencia que existe con respecto a los reales es el color, que en lugar de ser rojos son amarillos, color más visible a la hora de la simulación.

El LED es un diodo y como tal posee un cátodo y un ánodo, y a diferencia de las lámparas incandescentes que están especificadas para una tensión de trabajo determinada, los LEDs están especificados para una corriente de trabajo determinada. Por tanto, se debe limitar esta corriente con una resistencia. La forma típica de conectar un LED a una fuente de tensión se muestra en la siguiente figura:

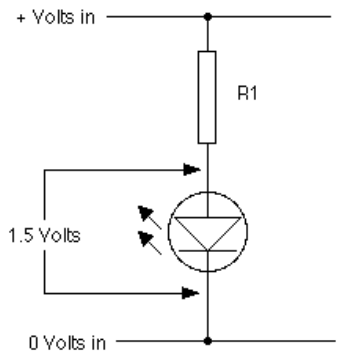


Figura 7.7: Conexión típica de un LED

El cátodo se conecta a tierra y el ánodo a la resistencia. La forma de calcular dicha resistencia viene dada por la ley de Ohm:

$$\text{Resistencia} = (V\_bat - V\_led)/ I\_led$$

donde:

V_bat es la tensión de entrada.

V_led es la tensión del LED (tensión que cae en los extremos del LED).

I_led es la intensidad que circula por el LED.

La corriente que circula por los LEDs depende del modelo utilizado: típicamente es 1,5V para un LED infrarrojo, 1,8V para un LED rojo, 2V para un LED

amarillo, 2,3V para un LED verde y 3,8V para un LED azul. Es decir, a mayor frecuencia de emisión, mayor es la tensión.

En este circuito se ha conectado LEDs amarillos por donde circula una corriente de 10 mA, por lo que el valor de la resistencia será:

$$R = (5\ V. - 2\ V.)/0{,}01\ A = 300\ \Omega.$$

Éste es el valor límite que puede tener la resistencia, por lo que en el circuito realizado se optó por la utilización de resistencias de 330 Ω, puesto que es el valor más cercano al 10% superior, ya que no se debe de sobrepasar la tensión V_led.

Para completar el circuito virtual sólo resta añadir los displays de 7 segmentos, el módulo LCD y el GLCD.

En este primer circuito virtual se utilizaron cuatro displays de 7 segmentos de cátodo común 7SEG-COM-CAT-GRN. Estos displays necesitan resistencias para limitar el paso de corriente, ya que al estar compuestos por LEDs la intensidad que soportan es limitada, por lo que el valor de las resistencias es el mismo que para los LEDs. Además, como se puede observar en la Figura 7.9, cada display de 7 segmentos tiene asociadas sus propias resistencias, ya que de no ser así existían errores de simulación.

El módulo LCD LM016L se ha conectado en modo 4 bits, donde los datos se transmiten por medio de los pines D7, D6, D5, D4. El resto de los pines de datos del display LCD, D3, D2, D1 y D0, se conectan a tierra.

El módulo GLCD que se ha utilizado es un dispositivo que se encuentra en la librería DISPLAY con el nombre AMPIRE 128X64. Dicho módulo es un GLCD gráfico de 128x64 con el controlador KS0108 integrado. Aunque este modelo no coincide exactamente con el modelo del entrenador EasyPIC4, puesto que presenta solamente 18 pines y el del entrenador 20, el comportamiento es idéntico al que presenta el módulo de la placa didáctica, por lo que simula perfectamente todos los programas creados para la placa EasyPIC4.

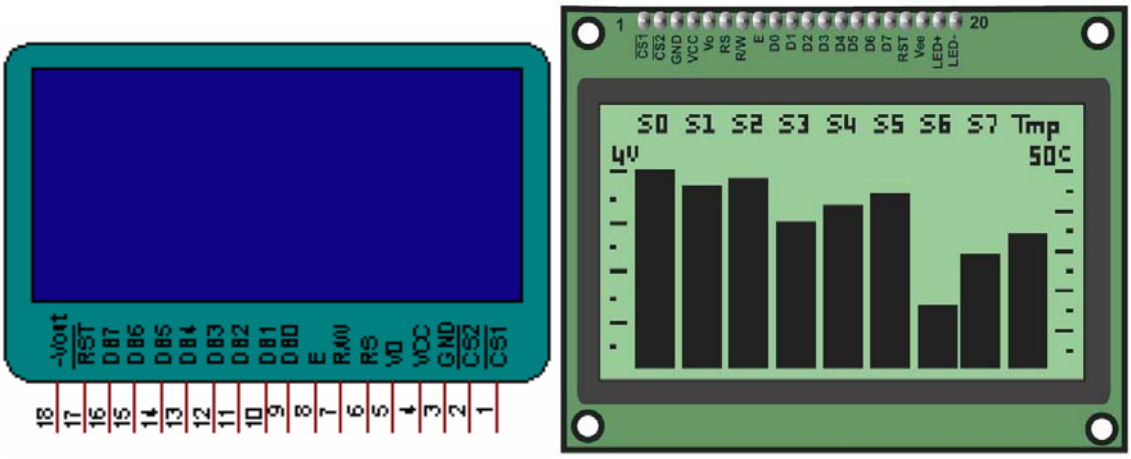


Figura 7.8: Pines de los GLCDs

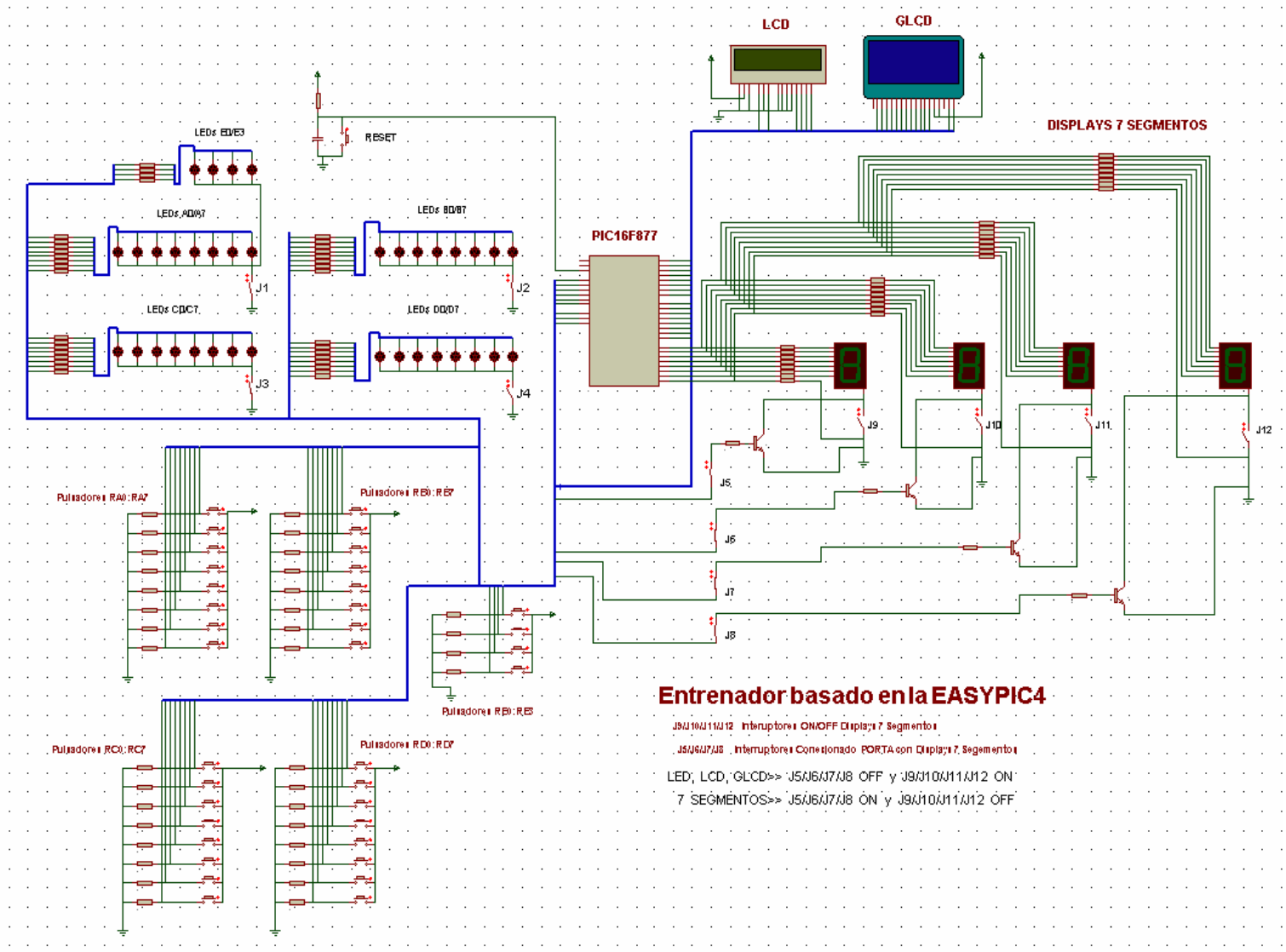


Figura 7.9: Circuito virtual entrenador_easyPIC4_1

7.3.3. CIRCUITO VIRTUAL EASYPIC4 CON ELEMENTOS DIGITALES

Tras la simulación del circuito virtual entrenador_easyPIC4_1, se observa que el funcionamiento es el correcto pero la velocidad de simulación es excesivamente lenta incluso después de modificar los parámetros de simulación, *Animated Circuits Configuration y Default Simulator Options.*

Con el fin de solventar el problema del tiempo de simulación, se analiza cada componente que forma el circuito virtual, observando que los elementos que hacen que la simulación de los diversos programas se ralentice son los LEDs y los displays de 7 Segmentos, ya que al suprimirlos del circuito y simularlo de nuevo, el tiempo de simulación se aproxima más al de una simulación real. Esto es debido a que los dispositivos utilizados en entrenador_easyPIC4_1 son analógicos, lo que hace que la simulación no se realice en tiempo real.

Una vez analizados los componentes que hacen que la simulación sea más lenta de lo normal, se procede a la sustitución de dichos dispositivos por sus equivalentes digitales. Para el caso de los LEDs lo único que se ha de hacer, es seleccionar el tipo de modelo digital en lugar del analógico.

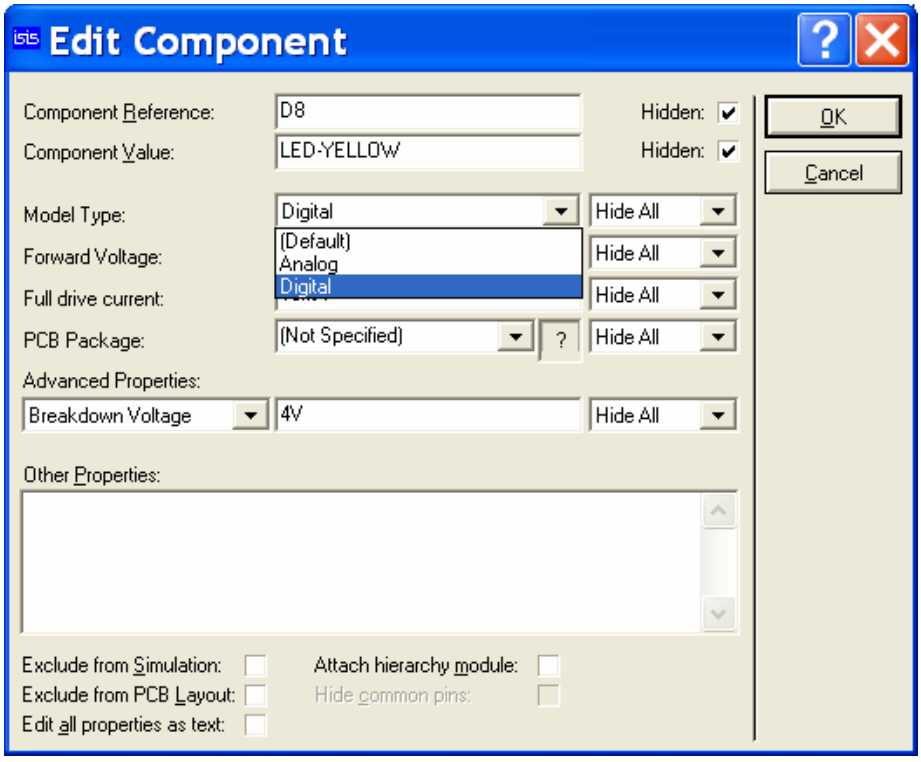


Figura 7.10: Edición de los LEDs

La modificación de los displays de 7 segmentos no es tan inmediata como para el caso de los LEDs, ya que en el cuadro de diálogo de edición de los displays no existe la posibilidad de seleccionar la opción analógico o digital, por lo que se ha de suprimir los cuatro displays de 7 segmentos, situando en su lugar un módulo 7SEG-MPX4-CC-BLUE, que presenta en un mismo dispositivo los cuatro displays de 7 segmentos. Con estas modificaciones del entrenador_easyPIC4_1 se crea el circuito virtual entrenador_easyPIC4_2 que además de simularse prácticamente en tiempo real, presenta una apariencia más clara que su circuito predecesor.

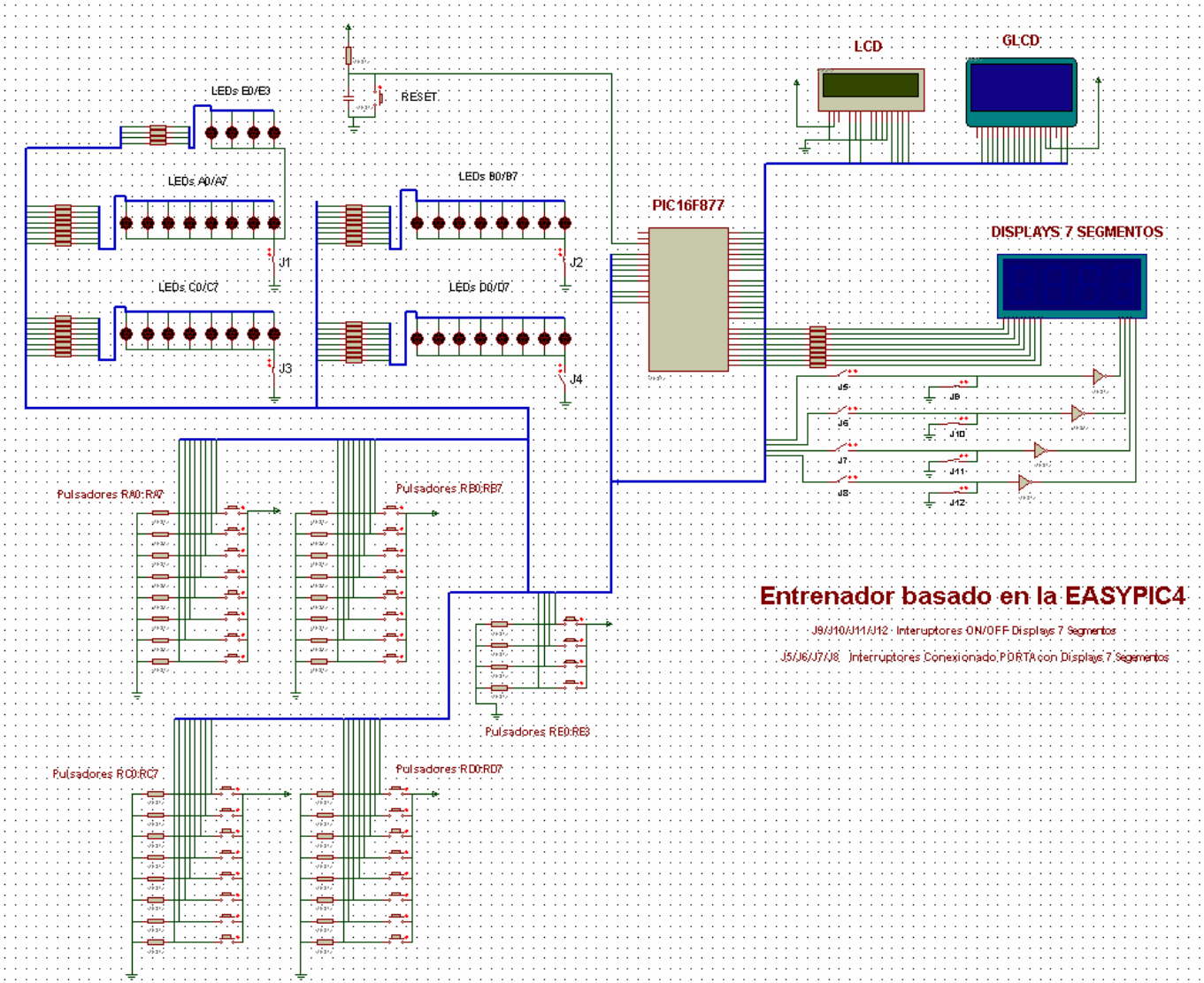


Figura 7.11: Circuito virtual entrenador_easyPIC4_2

CAPÍTULO 8. MICROBOT MOWAY

8.1. DESCRIPCIÓN FÍSICA DEL MICROBOT MOWAY

8.1.1. INTRODUCCIÓN

El microbot Moway, es un pequeño robot autónomo dotado de una serie de sensores que le ayudan a desenvolverse en un entorno real. A su vez cuenta con un grupo motor que le permite desplazarse en un terreno liso comandado a través del bus de comunicaciones I^2C.

Figura 8.1: Robots Moway

Todos estos periféricos están conectados a un microcontrolador que se encarga de gobernar el robot. Este pequeño robot cuenta además con opciones de ampliación a través de un bus de expansión por I^2C/SPI. En él se puede conectar, por ejemplo, un módulo de comunicaciones inalámbrico, una cámara de video, una tarjeta de prototipos o cualquier otro dispositivo que se considere interesante para el desarrollo de una tarea.

8.1.2. COMPONENTES DEL MICROBOT MOWAY

En este apartado se detallan los elementos que componen el robot Moway:

- Procesador
- Sistema motriz
- Grupo de sensores e indicadores
- Sistema de alimentación
- Conector de expansión

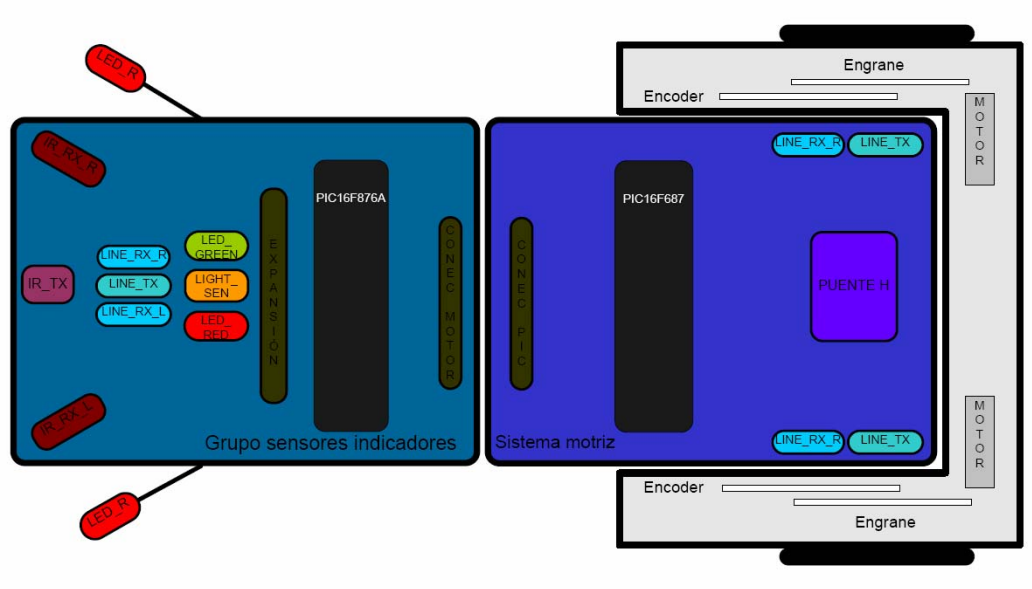


Figura 8.2: Representación de las partes de Moway

8.1.2.1. Procesador

El microbot Moway está gobernado por un microcontrolador PIC16F876A del fabricante Microchip Technology [5] que trabaja a 4Mhz. De sus puertos de entrada/salida cuelgan todos los periféricos distribuidos por el robot. A continuación se muestra una tabla con la asignación de pines del microcontrolador.

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Luz
RA1	I	Receptor infrarrojo derecho
RA2	I	Receptor sensor línea derecho
RA3	I	Receptor infrarrojo izquierdo
RA4	O	LED superior rojo
RA5	I	Receptor sensor línea izquierdo
PORTB		
RB1	O	Trasmisor sensores línea
RB2	O	Trasmisor infrarrojo
RB4	O	LED inferior derecho
RB6	O	LED superior verde
PORTC		
RC6	O	LED inferior izquierdo
RC7	I/O	Pad libre

Tabla 8.1: Conexiones PIC-Sensores

8.1.2.2. Sistema motriz

El robot móvil Moway dispone de un grupo servo-motor doble para poder desplazarse. Consta de una parte electrónica y otra mecánica. La parte electrónica se encarga principalmente de controlar la velocidad de los motores y la parte mecánica permite el desplazamiento con una potencia suficiente para que se desplace sin problemas en diferentes entornos.

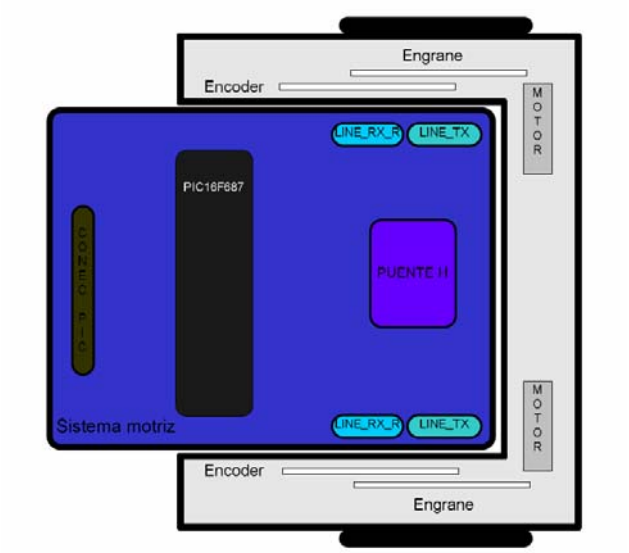


Figura 8.3: Sistema motriz: electrónica y mecánica

El grupo servo-motor tiene diversas funcionalidades:

- Control de velocidad: Controla la velocidad de cada motor por separado.
- Control de tiempo: Controla el tiempo en cada comando con una precisión de 100ms.
- Control de distancia recorrida: Controla la distancia recorrida en cada comando con una precisión de 1.7mm.
- Cuentakilómetros general: Cuenta la distancia recorrida desde el comienzo de los comandos.
- Control de ángulo: Control de ángulo cuando se produce la rotación de Moway.

8.1.2.3. Grupo de sensores e indicadores

Este grupo consta de diferentes sensores e indicadores luminosos conectados al microprocesador de Moway, con los que el robot interactúa con el mundo exterior:

- Dos sensores de línea.
- Dos sensores detectores de obstáculos.
- Sensor de luz.
- Un conector de expansión.
- Cuatro diodos LED.
- Un pad libre.

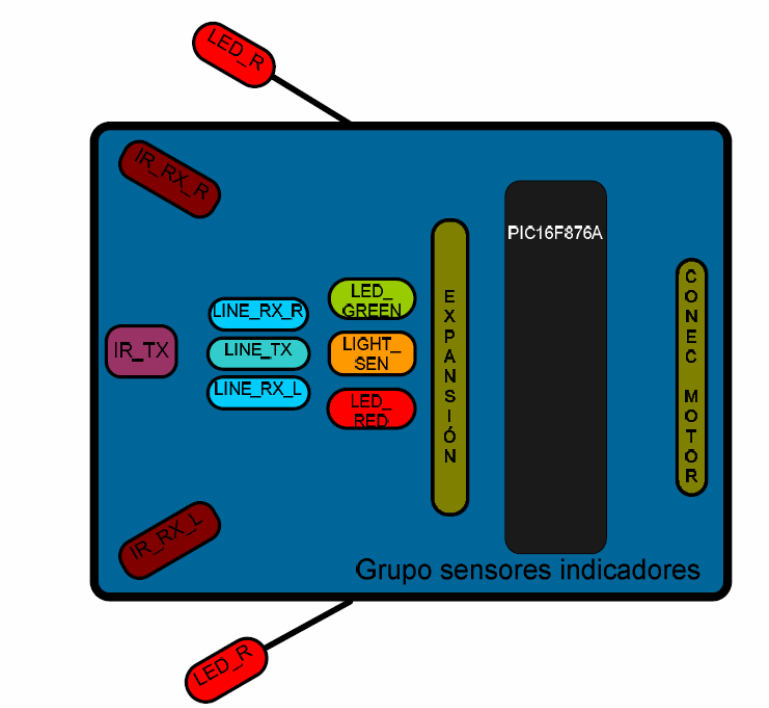


Figura 8.4: Grupo de sensores e indicadores

- Sensores de línea

Los sensores de línea son dos optoacopladores de reflexión montados en la parte inferior delantera del robot. Utilizan la reflexión de luz infrarroja para detectar el tono del suelo en el punto en que se encuentra el robot.

Estos dos sensores están conectados a dos de los puertos analógicos del microcontrolador de manera que no sólo podemos detectar contrastes fuertes en el suelo, como líneas blancas sobre fondo negro, sino que es posible discernir entre diferentes tonos.

En las siguientes imágenes podemos ver los tres casos que se pueden dar:

- Superficie clara: La superficie blanca hace que toda la luz infrarroja se refleje y por lo tanto a la salida del transistor en modo común se obtiene un voltaje bajo.

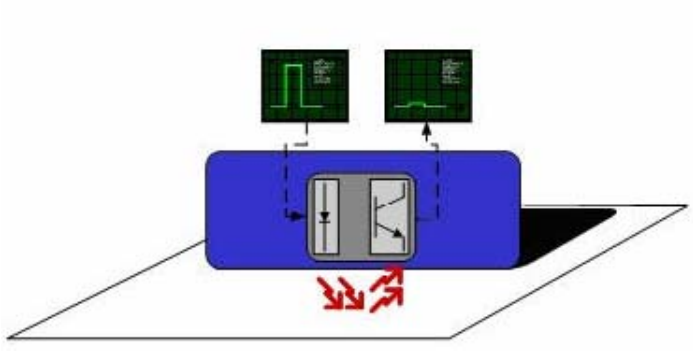


Figura 8.5: Sensor de línea en superficie clara

- Superficie de color: La superficie de color hace que parte de la luz emitida se refleje obteniendo un voltaje intermedio en la entrada del canal analógico del microcontrolador. De esta manera es fácil identificar colores.

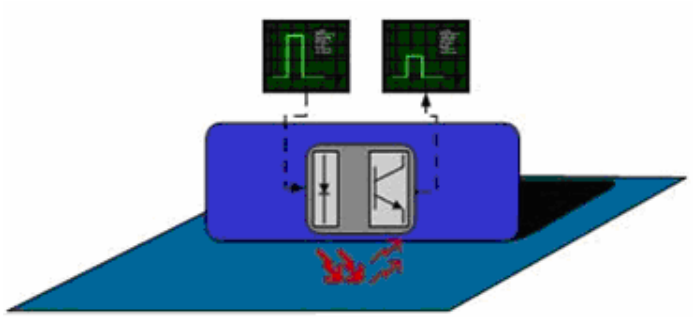


Figura 8.6: Sensor de línea en superficie de color

- Superficie oscura: La superficie oscura hace que se refleje muy poca luz teniendo un voltaje alto a la salida del sensor.

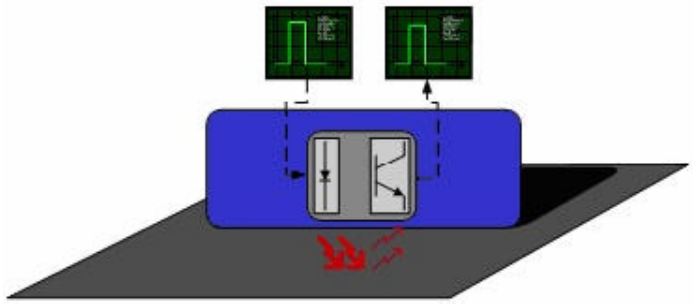


Figura 8.7: Sensor de línea en superficie oscura

En la Tabla 8.2 se muestra las conexiones del PIC con el sensor de línea.

Pin PIC	I/O	Sensor
PORTA		
RA2	I	Receptor sensor línea derecho
RA5	I	Receptor sensor línea izquierdo
PORTB		
RB1	O	Trasmisor sensores de línea izquierdo y derecho

Tabla 8.2: Conexiones Sensores de línea- PIC

- Sensores detectores de obstáculos

Al igual que los sensores de línea, los sensores detectores de obstáculos utilizan también la luz infrarroja para detectar objetos situados en la parte delantera de Moway. El material del frontal es un filtro infrarrojos que bloquea parte de la componente infrarroja de la luz ambiente. El sensor está compuesto por una fuente de luz infrarroja y dos receptores colocados en ambos extremos de Moway.

La salida de los receptores está conectada a las entradas analógicas del microcontrolador por lo que no sólo se detecta la presencia de algún objeto (modo digital) sino que también se puede medir la distancia al mismo (modo analógico).

El funcionamiento del sensor es similar al sensor de línea. El emisor de luz emite un pulso de una duración de 70us. que, si existe un obstáculo, el receptor capta utilizando una etapa de filtrado y amplificación. La distancia de alcance en digital es de unos 3cm y se recomienda un entorno claro para aumentar la posibilidad de reflexión de la luz infrarroja.

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Luz
RA1	I	Receptor infrarrojo derecho
RA3	I	Receptor infrarrojo izquierdo
PORTB		
RB2	O	Trasmisor infrarrojo

Tabla 8.3: Conexiones sensor antichoque-PIC

- Sensor de luz

Permite conocer la intensidad de luz que entra por una pequeña apertura con forma de media luna en la parte superior del chasis.

- LEDs frontales

Los dos leds frontales, que se han dispuesto uno en cada lateral de la parte frontal, se encuentran detrás del filtro infrarrojo delantero de manera que sólo se visualizan cuando se activan.

Pin PIC	I/O	Sensor
PORTB		
RB4	O	LED inferior derecho
PORTC		
RC6	O	LED inferior izquierdo

Tabla 8.5: Conexión PIC-Led frontales

- LED superior bicolor

Este indicador doble comparte la misma apertura en la parte superior del robot que el sensor de luz. Están conectados a dos salidas digitales del microcontrolador.

Pin PIC	I/O	Sensor
PORTA		
RA4	O	LED superior rojo
PORTB		
RB6	O	LED superior verde

Tabla 8.6: Conexión PIC-Led superior

8.1.2.4. Sistema de alimentación

La batería de Moway, que se encuentra en el interior, es una pequeña célula de LiPo recargable. La duración de la batería depende en gran medida de los sensores activos y del tiempo de utilización de los motores.

En el CD adjunto a la memoria del presente Proyecto Fin de Carrera se incluye el manual de Moway, donde se puede encontrar más información referente a los componentes y el software utilizado para programarlo. Dicho manual se puede consultar en la página web de Moway [21].

8.2. PRIMEROS EJEMPLOS DE PROGRAMAS

En este apartado se realiza la verificación de los componentes del robot móvil Moway y de los programas ejemplo de los que se dispone al instalar el software.

Para ello se han cargado varios ficheros en el microbot, comprobando el funcionamiento tanto de los sensores como de los motores.

8.2.1. INTRODUCCIÓN

Como se muestra a continuación, para cada una de las aplicaciones realizadas, se dispone de tres ficheros distintos:

- Dos programas en ensamblador, uno utilizando librerías de variables en posición fija, y el otro utilizando librerías reubicables.

  La diferencia entre utilizar una librería u otra es, como su propio nombre indica, que utilizando librerías de posición fija las variables se almacenan en una dirección determinada, mientras que cuando se utilizan librerías reubicables, el programador no sabe en qué parte de la memoria principal se situará el programa cuando se ejecute.

- Un programa en C equivalente a los dos ficheros anteriores.

8.2.2. PROGRAMAS UTILIZADOS PARA VERIFICAR EL MICROBOT MOWAY

8.2.2.1. Funcionamiento de los sensores

- Ficheros “ASM_SEN_01”, “ASM_SEN_02” y “CCS_SEN_01”

Descripción:

Mediante los siguientes programas, los LEDs inferiores del robot móvil Moway se encienden cuando se detecta un obstáculo.

A continuación se muestran los ficheros:

- ASM_MOT_01: Código en ensamblador utilizando las librerías con variables en posición fija lib_sen_moway_10.
- ASM_MOT_02: Código en ensamblador utilizando las librerías reubicables lib_sen_moway_11.

  El código ASM_MOT_02 es idéntico al de ASM_MOT_01, salvo por la instrucción #include “lib_sen_moway_10.inc” que se sustituye por #include “lib_sen_moway_11.inc”.
- CCS_MOT_01: Código en C.

Código en ensamblador:

```
;*****************************************************************************************************************
        list p=16F876A
;*      Definición de los registros internos del PIC
#include "P16F876A.INC"
;*      Vector de reset
        org             0x00
        goto    INIT
;*      Memoria de programa  *
        org             0x05
;************************

;***************************[LIBRERÍAS DE MOWAY]*************************************************
#include "lib_sen_moway_10.inc"             ; #include "lib_sen_moway_11.inc"
;*****************************************************************************************************************

;*********************************[MAIN]*********************************************************
INIT
        ;Se configuran los sensores
        call    SEN_CONFIG
```

```
MAIN

   ;Parpadeo del Led verde
         call      LED_TOP_GREEN_ON_OFF

BUCLE
         ;Se comprueba los sensores de obstaculo
   call  SEN_OBS_DIG

         ;Si hay un obstaculo en la derecha se enciende el LED derecho
         btfsc     SEN_OBS_R,0
         call      LED_R_ON
         btfss     SEN_OBS_R,0
         call      LED_R_OFF

         ;Si hay un obstaculo en la izquierda se enciende el LED izquierdo
         btfsc     SEN_OBS_L,0
         call      LED_L_ON
         btfss     SEN_OBS_L,0
         call      LED_L_OFF

         goto      BUCLE
;*****************************************************************************************************************

 END
```

Código en C:

```
//*****************************************************************************************************************

#include <16F876A.h>                    //Microcontrolador de Moway
#DEVICE ADC= 8
#use delay(clock=4000000)

//***************************[LIBRERÍAS DE MOWAY]*****************************************************
#include "lib_sen_moway_10.h"
//*****************************************************************************************************************

//************************************[MAIN]*************************************************************
void main()
{

   //Se configuran los sensores
   SEN_CONFIG();

   //Se espera 1s
   delay_ms(1000);
   //Parpadeo del Led verde
   LED_TOP_GREEN_ON_OFF();


   while(1){

      //Se comprueba los sensores de obstaculo
      SEN_OBS_DIG();

      //Si hay un obstaculo en la izquierda se enciende el LED izquierdo
      if(SEN_OBS_L==1)
```

```
      LED_L_ON();
    else
      LED_L_OFF();

    //Si hay un obstaculo en la derecha se enciende el LED derecho
    if(SEN_OBS_R==1)
      LED_R_ON();
    else
      LED_R_OFF();


  }

}
```

8.2.2.2. Funcionamiento de los motores

- Ficheros “ASM_MOT_01” “ASM_MOT_02” “CCS_MOT_02”

Descripción:

Mediante los siguientes códigos se consigue que el robot Moway ejecute diferentes movimientos utilizando para ello los motores. Los movimientos realizados son:

- Movimiento hacia delante al máximo de velocidad durante 3 segundos.
- Movimiento hacia atrás al 15% de velocidad recorriendo un espacio de 170 mm.
- Cambiar velocidad (90% atrás) al motor izquierdo y hacer una distancia de 170 mm.
- Cambiar velocidad (80% adelante) al motor derecho durante 3 segundos.
- Cambiar velocidad (40% adelante) al motor derecho durante 3 segundos.
- Rotación respecto al centro a la derecha al 80% de velocidad durante 3 segundos.
- Rotación respecto la rueda izquierda adelante al 20% de velocidad 180.72º.

- Curva adelante a derechas al 50% con un radio de 10 durante 3 segundos.
- Curva hacia atrás izquierdas al 80% con un radio de 15 durante 170mm.
- Recto hacia atrás al 100% de velocidad indefinidamente.

A continuación se muestran los ficheros:

- ASM_MOT_01: Código en ensamblador utilizando las librerías con variables en posición fija lib_sen_moway_10.
- ASM_MOT_02: Código en ensamblador utilizando las librerías reubicables lib_sen_moway_11.

  El código ASM_MOT_02 es idéntico al de ASM_MOT_01, salvo por la instrucción #include “lib_sen_moway_10.inc” que se sustituye por #include “lib_sen_moway_11.inc”.
- CCS_MOT_01: Código en C.

Código en ensamblador:

```
;*****************************************************************************************************************
        list p=16F876A
;*      Definición de los registros internos del PIC
#include "P16F876A.INC"
;*      Vector de reset
        org             0x00
        goto    INIT
;*      Memoria de programa  *
        org             0x05
;**************************[LIBRERÍAS DE MOWAY]*********************************************
#include "lib_mot_moway_10.inc"                 ; #include "lib_sen_moway_11.inc"
;*****************************************************************************************************************

INIT

        ;Configuración del microcontrolador para poder mandar comandos al motor
        call    MOT_CONFIG

        ;*****************************[MAIN PROGRAM]*************************************************
MAIN

BUCLE



        ;Recto adelante al 100% de velocidad 3 segundos (100ms x 30)
        movlw           .100
```

```
movwf       MOT_VEL
movlw       .30
movwf       MOT_T_DIST_ANG
bsf             MOT_CON,FWDBACK
bsf             MOT_CON,COMTYPE
call        MOT_STR
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1


;Recto hacia atras al 15% de velocidad 170mm (1.7mm x 100)
movlw       .15
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bcf             MOT_CON,FWDBACK
bcf             MOT_CON,COMTYPE
call        MOT_STR
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1



;Cambiar velocidad (90% atrás) al motor izquierdo y hacer una distancia de 170 mm
;(1.7mm x 100)
movlw       .90
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bcf             MOT_CON,FWDBACK
bcf             MOT_CON,COMTYPE
bcf             MOT_CON,RL
call        MOT_CHA_VEL
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1

;Cambiar velocidad (80% adelante) al motor derecho durante 3 segundos
;(100ms x 30)
;Cambiar velocidad (40% adelante) al motor derecho durante 3 segundos
;(100ms x 30)
movlw       .80
movwf       MOT_VEL
movlw       .30
movwf       MOT_T_DIST_ANG
bsf             MOT_CON,FWDBACK
bsf             MOT_CON,COMTYPE
bsf             MOT_CON,RL
call        MOT_CHA_VEL

movlw       .40
movwf       MOT_VEL
movlw       .30
movwf       MOT_T_DIST_ANG
bsf             MOT_CON,FWDBACK
bsf             MOT_CON,COMTYPE
bcf             MOT_CON,RL
call        MOT_CHA_VEL
```

```
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1



;Rotación respecto al centro a la derecha al 80% de velocidad durante 3 segundos
;(100ms x 30)
movlw       .80
movwf       MOT_VEL
movlw       .30
movwf       MOT_T_DIST_ANG
movlw       0x01
movwf       MOT_CENWHEEL
bsf             MOT_CON,FWDBACK
bsf             MOT_CON,COMTYPE
bsf             MOT_CON,RL
call        MOT_ROT
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1

;Rotación respecto la rueda izquierda adelante al 20% de velocidad 180.72º
movlw       .20
movwf       MOT_VEL
movlw       .50
movwf       MOT_T_DIST_ANG
movlw       0x00
movwf       MOT_CENWHEEL
bsf             MOT_CON,FWDBACK
bcf             MOT_CON,COMTYPE
bcf             MOT_CON,RL
call        MOT_ROT
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1


;Curva adelante a derechas al 50% con un radio de 10 durante 3 segundos
;(100ms x 30)
;VEL_I=60
;VEL_D=40
movlw       .50
movwf       MOT_VEL
movlw       .30
movwf       MOT_T_DIST_ANG
movlw       .10
movwf       MOT_RAD
bsf             MOT_CON,FWDBACK
bsf             MOT_CON,COMTYPE
bsf             MOT_CON,RL
call        MOT_CUR
;Hasta que el comando no termine no se hace nada
btfss       MOT_END
goto        $-1

;Curva hacia atras izquierdas al 80% con un radio de 15 durante 170mm
;(1.7mm //x 100)
;VEL_I=95
;VEL_D=65
```

```
        movlw       .80
        movwf       MOT_VEL
        movlw       .100
        movwf       MOT_T_DIST_ANG
        movlw       .15
        movwf       MOT_RAD
        bcf             MOT_CON,FWDBACK
        bcf             MOT_CON,COMTYPE
        bcf             MOT_CON,RL
        call        MOT_CUR
        ;Hasta que el comando no termine no se hace nada
        btfss       MOT_END
        goto        $-1


        ;Recto hacia atras al 100% de velocidad indefinidamente
        movlw       .100
        movwf       MOT_VEL
        movlw       .0
        movwf       MOT_T_DIST_ANG
        bcf             MOT_CON,FWDBACK
        bsf             MOT_CON,COMTYPE
        call        MOT_STR


        ;Se va preguntando al motor cuando llega el comando a los 3 segundos de
        ;ejecución. Cuando pasen tres segundos paramos el motor.
        movlw       STATUS_T
        movwf       MOT_STATUS_COM
FEEDBACK
        call        MOT_FDBCK
        movfw       MOT_STATUS_DATA_0
        sublw       .30
        btfsc       STATUS,C
        goto        FEEDBACK

        call    MOT_STOP


        goto    BUCLE
        ;*****************************************************************************
        END
```

Código en C:

```
//*************************************************************************************************************

  #include <16F876A.h>                  //Microcontrolador de Moway
  #DEVICE ADC= 8
  #use delay(clock=4000000)

//***************************[LIBRERÍAS DE MOWAY]*****************************************************
  #include "lib_mot_moway_10.h"             //Se incluye la librería de motores
//*************************************************************************************************************

//************************************[MAIN]*********************************************************
void main()
  {
```

```
  //Se configura el microcontrolador para mandar comandos al motor
  MOT_CONFIG();


while(1){

  Delay_ms(2000);

  //Recto adelante al 100% de velocidad 3 segundos (100ms x 30)
  MOT_STR(100, FWD, TIME, 30);
  while(!input(MOT_END)){}
  //Recto hacia atras al 15% de velocidad 170mm (1.7mm x 100)
  MOT_STR(15, BACK, DISTANCE, 100);
  while(!input(MOT_END)){}


  //Cambiar velocidad (90% atrás) al motor izquierdo y hacer una distancia de 170 mm
  //(1.7mm //x 100)
  MOT_CHA_VEL(90, BACK, LEFT, DISTANCE, 100) ;
  while(!input(MOT_END)){}

  //Cambiar velocidad (80% adelante) al motor derecho durante 3 segundos
  //(100ms x 30)
  //Cambiar velocidad (40% adelante) al motor izquierdo durante 3 segundos
  //(100ms x 30)
  MOT_CHA_VEL(80, FWD, RIGHT, TIME, 30) ;
  MOT_CHA_VEL(40, FWD, LEFT, TIME, 30) ;
  while(!input(MOT_END)){}


  //Rotación respecto al centro a la derecha al 80% de velocidad durante 3 segundos
  //(100ms x 30)
  MOT_ROT(80, FWD, CENTER, RIGHT, TIME, 30) ;
  while(!input(MOT_END)){}
  //Rotación respecto la rueda izquierda al 20% de velocidad 180º  (360º/(100/50) = 180
  MOT_ROT(20, BACK, WHEEL, LEFT, ANGLE, 50) ;
  while(!input(MOT_END)){}


  //Curva a derechas al 50% con un radio de 10 durante 3 segundos
  //(100ms x 30)
  MOT_CUR(50, FWD, 10, RIGHT, TIME, 30) ;
  while(!input(MOT_END)){}
  // Curva a izquierdas al 80% con un radio de 15 durante 170mm
  //(1.7mm //x 100)
  MOT_CUR(80, BACK, 15, LEFT, DISTANCE, 100) ;
  while(!input(MOT_END)){}


  //Recto hacia atras al 15% de velocidad 170mm (1.7mm x 100)
  MOT_STR(100, BACK,TIME,0);
  MOT_FDBCK(STATUS_T);
  while(MOT_STATUS_DATA_0<30){
  MOT_FDBCK(STATUS_T);
  }
  MOT_STOP();

}
```

```
}
```

- Ficheros “Moway_first_project_ASM” y “Moway_first_project_CCS”

Descripción:

Tanto utilizando el código en ensamblador como en lenguaje C, se consigue que el microbot Moway se mueva recto y evite obstáculos girando 180º.

Código en ensamblador:

```
        list p=16F876A
;*      Definición de los registros internos del PIC
#include "P16F876A.INC"
;*      Vector de reset
        org             0x00
        goto    INIT
;*      Memoria de programa   *
        org             0x05
;************************
;*      Incluir las librerías de Moway
#include "lib_mot_moway_10.inc"
#include "lib_sen_moway_10.inc"


INIT
        ;Se configuran los sensores
        call    SEN_CONFIG

        ;*****************************[MAIN PROGRAM]*********************************
MAIN

        ;Parpadeo del Led verde
        call    LED_TOP_GREEN_ON_OFF

        ;Se mueve hacia adelante
        movlw           .70
        movwf           MOT_VEL
        movlw           .0
        movwf           MOT_T_DIST_ANG
        bsf                     MOT_CON,FWDBACK
        bsf                     MOT_CON,COMTYPE
        call            MOT_STR

BUCLE

        ;Se comprueba los sensores de obstaculo
        call            SEN_OBS_DIG

        ;Si hay un obstaculo en la derecha se enciende el LED derecho
        btfsc           SEN_OBS_L,0
        call            OBS_SI

        ;Si hay un obstaculo en la izquierda se enciende el LED izquierdo
        btfsc           SEN_OBS_R,0
```

```
        call            OBS_SI

        goto    BUCLE

        ;*******************************************************************************
OBS_SI
        call            LED_TOP_RED_ON_OFF
        ;Rotación a la derecha respecto al centro al 70% de velocidad 180.72º
        movlw           .70
        movwf           MOT_VEL
        movlw           .50
        movwf           MOT_T_DIST_ANG
        movlw           0x01
        movwf           MOT_CENWHEEL
        bsf                     MOT_CON,FWDBACK
        bcf                     MOT_CON,COMTYPE
        bsf                     MOT_CON,RL
        call            MOT_ROT
        ;Hasta que el comando no termine no se hace nada
        btfss           MOT_END
        goto            $-1
        ;Se mueve hacia adelante
        movlw           .70
        movwf           MOT_VEL
        movlw           .0
        movwf           MOT_T_DIST_ANG
        bsf                     MOT_CON,FWDBACK
        bsf                     MOT_CON,COMTYPE
        call            MOT_STR
        return

 END
```

Código en C:

```
#include "Moway_first_project_CCS.h"

#include "lib_sen_moway_10.h"
#include "lib_mot_moway_10.h"
void main()
{

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);

  // TODO: USER CODE!!

  //Se configuran los sensores
  SEN_CONFIG();

  //Se espera 1s
  delay_ms(1000);
```

```
  //Parpadeo del Led verde
  LED_TOP_GREEN_ON_OFF();
  //Se mueve hacia adelante
  MOT_STR(70,FWD,TIME,0);
  while(1){
    //Se comprueba los sensores de obstaculo
    SEN_OBS_DIG();
    //Si hay un obstaculo en la izquierda se enciende el LED izquierdo y rota
    if(SEN_OBS_L==1){
      LED_L_ON();
      MOT_ROT(70,FWD,CENTER,RIGHT,ANGLE,50);
      while(!input(MOT_END)){}
      MOT_STR(70,FWD,TIME,0);
    }
    else
      LED_L_OFF();
    //Si hay un obstaculo en la derecha se enciende el LED derecho y rota
    if(SEN_OBS_R==1){
      LED_R_ON();
      MOT_ROT(70,FWD,CENTER,RIGHT,ANGLE,50);
      while(!input(MOT_END)){}
      MOT_STR(70,FWD,TIME,0);
    }
    else
      LED_R_OFF();
  }
}
```

8.3. COMBATE DE SUMO

En la actualidad la realización de combates de sumo entre robots es una práctica muy extendida, existiendo numerosas competiciones en diversos lugares del mundo para medir la fuerza, la táctica y la agilidad de los microbots. Por ello, se ha considerado oportuno la realización de una primera aproximación a un combate entre dos robots Moway. Dicho combate consiste en una lucha entre dos robots sobre una tarima (también denominada *ring*), con el objetivo de sacar al robot oponente del recinto.

A continuación se describen los programas desarrollados para los microbots: “sumo_atacante.c” y “sumo_defensivo.c”. La diferencia existente entre ambos códigos reside en el comportamiento que presenta cada robot móvil cuando detecta al contrincante, ya que uno de ellos al detectar al otro robot móvil

intentará desplazarlo para sacarlo del *ring*, mientras que el otro intentará esquivar la colisión.

- Fichero “sumo_atacante.c”

Descripción:

La siguiente aplicación desarrollada muestra el comportamiento del robot móvil Moway en una pelea de sumo. El robot móvil Moway 01, que es al microbot que se le ha cargado dicho código, dispone de un modo de exploración mediante el cual se mueve lentamente sin salirse del *ring*. Cuando detecta algo por alguno de sus sensores sus movimientos se vuelven mucho más rápidos y eficaces, girándose hacia el lado por donde se ha detectado al adversario para así situarse en una posición que le permita empujar al otro robot fuera de la tarima.

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////            sumo_atacante.c
////
////PFC Mª Nieves Robles Botella
////Ejemplo de utilización de Moway
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "sumo.h"

#include "lib_sen_moway_10.h"
#include "lib_mot_moway_10.h"
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);

  //Se configuran los sensores
  SEN_CONFIG();
  MOT_CONFIG();

  //Se espera 1s
  delay_ms(1000);
  //Parpadeo del Led verde
  LED_TOP_GREEN_ON_OFF();
  //Se mueve hacia adelante
  MOT_STR(20,FWD,TIME,0);                              //Se mueve hacia delante al 70%
```

```
  while(1){
    //Se comprueba los sensores de obstáculo
     SEN_LINE_DIG();                    //Devuelve si detecta línea negra sobre suelo claro
     SEN_OBS_DIG();
     //Si detecta línea porla izquierda se enciende el LED izquierdo y rota
     if(SEN_LINE_L==1 || SEN_LINE_R==1 ){
        if(SEN_LINE_L==1){                                  //Si se sale por la izquierda
        LED_L_ON();                                         //Enciende el LED izqdo.
        }
        if(SEN_LINE_R==1){
        LED_R_ON();
        }
       MOT_ROT(1,FWD,CENTER,RIGHT,ANGLE,30);   //Gira a la dcha. 360/(100/30)=108º
       while(!input(MOT_END)){}
       MOT_STR(1,FWD,TIME,0);
     }
     else
     {
       LED_L_OFF();
      LED_R_OFF();
     }

     if (SEN_OBS_R==1 && SEN_OBS_L==0)              //Si sensor dcho está ON e izqdo OFF
     {
        LED_R_ON();
        LED_L_OFF();
        MOT_ROT(5,FWD,CENTER,RIGHT,ANGLE,2);   //Velocidad 5% y ángulo 7.2º
        while(!input(MOT_END)){}
        MOT_STR(100,FWD,TIME,0);                        //Se mueve hacia delante al 100%
     }

     if(SEN_OBS_L==1 && SEN_OBS_R==0 )
     {
        LED_L_ON();
        LED_R_OFF();
        MOT_ROT(5,FWD,CENTER,LEFT,ANGLE,2);    //Velocidad 5% y angulo 7.2º
        while(!input(MOT_END)){}
        MOT_STR(100,FWD,TIME,0);                        //Se mueve hacia delante al 100%
     }


     if (SEN_OBS_R==1 && SEN_OBS_L==1)              //Si sensor dcho ON e izqdo ON
     {
        LED_R_ON();
        LED_L_ON();
        MOT_STR(100,FWD,TIME,0);                        //Se mueve hacia delante al 100%
     }
  }
}
```

Como se puede observar tras analizar el código, para la creación tanto del fichero anterior como del siguiente, se ha de incluir las librerías “sumo.h”, "lib_sen_moway_10.h” y "lib_mot_moway_10.h". Estas librerías contienen las funciones necesarias para utilizar los sensores y motores de los robots móviles Moway tales como LED_R_ON, LED_L_ON, MOT_STR, MOT_ROT, etc.

- Fichero “sumo_defensivo.c”

Descripción:

El microbot Moway que contiene el fichero “sumo_defensivo.c” se comporta de la misma manera que el robot con el código “sumo_atacante.c”, sin embargo cuando éste detecta al otro microbot no intenta sacarlo del *ring*, sino que se que realiza un giro de 180º y se aleja lo máximo posible del adversario con el fin de no ser desplazado por éste.

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////          sumo_defensivo.c
////
////PFC Mª Nieves Robles Botella
////Ejemplo de utilización de Moway
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "sumo.h"

#include "lib_sen_moway_10.h"
#include "lib_mot_moway_10.h"
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);

  //Se configuran los sensores y los motores
  SEN_CONFIG();
  MOT_CONFIG();
  //Se espera 1s
  delay_ms(1000);
  //Parpadeo del Led verde
  LED_TOP_GREEN_ON_OFF();
  //Se mueve hacia adelante
  MOT_STR(20,FWD,TIME,0);                    //Se mueve hacia delante al 20%
  while(1){
    //Se comprueba los sensores de obstáculo
    SEN_LINE_DIG();                    //Devuelve si detecta línea negra sobre suelo claro
    SEN_OBS_DIG();
    //Si detecta línea porla izquierda se enciende el LED izquierdo y rota
    if(SEN_LINE_L==1 || SEN_LINE_R==1 ){
          if(SEN_LINE_L==1){                         //Si se sale por la izquierda
           LED_L_ON();                               //Enciende el LED izqdo.
          }
```

```
            if(SEN_LINE_R==1){
            LED_R_ON();
            }
      MOT_ROT(1,FWD,CENTER,RIGHT,ANGLE,20);   //Gira a la dcha.360/(100/20)=72
      while(!input(MOT_END)){}
      MOT_STR(1,FWD,TIME,0);
    }
    else
       {
      LED_L_OFF();
       LED_R_OFF();
       }
    if (SEN_OBS_R==1 && SEN_OBS_L==0)   //Si sensor dcho. está en ON e izqdo. OFF
    {
       LED_R_ON();
       LED_L_OFF();
       MOT_ROT(100,FWD,CENTER,LEFT,ANGLE,50);   //Gira hacia la izqda
    while(!input(MOT_END)){}
       MOT_STR(100,FWD,TIME,0);                 //Se mueve hacia delante al 100%
    }
    if(SEN_OBS_L==1 && SEN_OBS_R==0 )
    {
       LED_L_ON();
       LED_R_OFF();
       MOT_ROT(100,FWD,CENTER,RIGHT,ANGLE,50);         //Velocidad 100% y angulo 180º
       while(!input(MOT_END)){}
       MOT_STR(100,FWD,TIME,0);                 //Se mueve hacia delante al 100%

    }
    if (SEN_OBS_R==1 && SEN_OBS_L==1)   //Si sensor dcho. ON e izqdo. en ON
    {
       LED_R_ON();
       LED_L_ON();
       MOT_STR(100,BACK,TIME,0);                //Se mueve hacia atrás
    }
  }
}
```

CAPÍTULO 9. CONLUSIONES Y TRABAJOS FUTUROS

El presente Proyecto Fin de Carrera se ha centrado básicamente en el desarrollo de aplicaciones prácticas para la docencia mediante la programación de microcontroladores PIC de gama media.

La educación es un proceso interminable, puesto que cada día se aprenden cosas nuevas o se actualizan las ya conocidas o aprendidas, por eso se debe estar en contacto con las tecnologías del momento y preparado para recibir las nuevas.

Por lo expresado anteriormente la elaboración del presente proyecto sirve para comprender un poco más la teoría de los microcontroladores y su aplicación; de ahí la importancia del mismo, donde se puede apreciar lo interesante que es la programación en nuestra vida cotidiana.

Como trabajo futuro o continuación del presente proyecto, se ha pensado en la creación de un microbot con una estructura abierta que permita la modificación o ampliación en la CPU del robot o en sus sensores. La Figura 9.1 muestra el prototipo que se está desarrollando, basado en el sistema descrito en el libro [22].

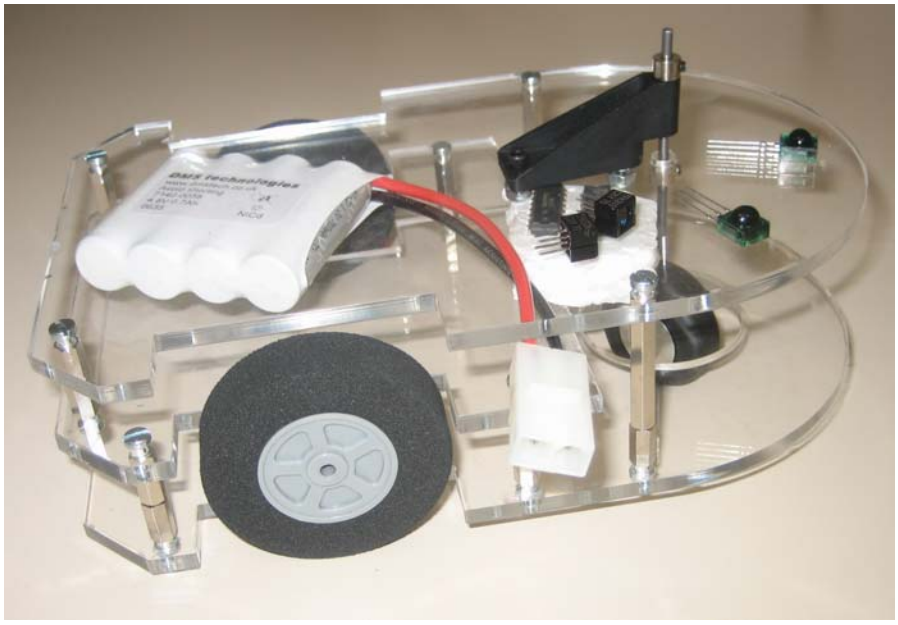


Figura 9.1: Microbot con estructura abierta

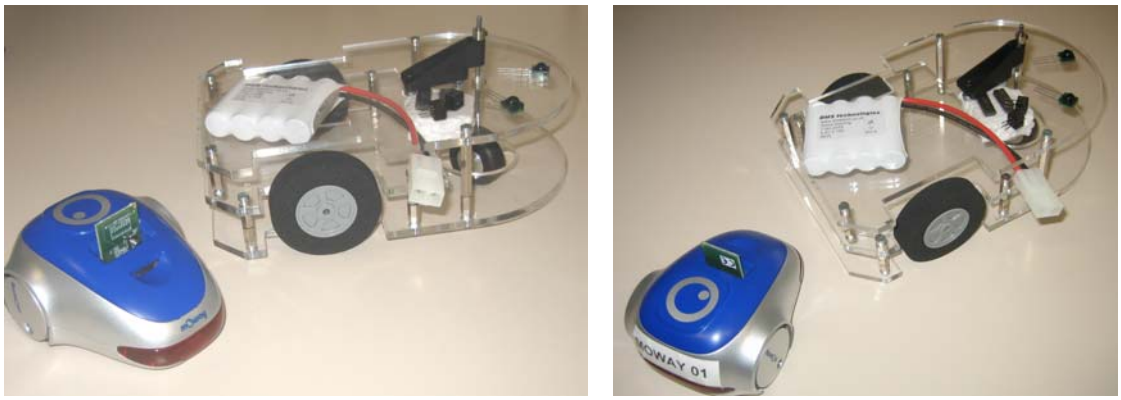


Figura 9.2: Robot móvil Moway y Microbot con estructura abierta

ANEXO A. DRIVERS DESARROLLADOS

En el presente anexo, se describen las librerías desarrolladas para los módulos LCD y GLCD.

A.1. DRIVER “miLCD.c”

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                                miLCD.c
////                                      Driver for LCD module
////
////  lcd_init()   Must be called before any other function.
////
////  lcd_putc(c)  Will display c on the next position of the LCD.
////                     The following have special meaning:
////                      \f  Clear display
////                      \n  Go to start of second line
////                      \b  Move back one position
////
////  lcd_gotoxy(x,y) Set write position on LCD (upper left is 1,1)
////
////  lcd_getc(x,y)   Returns character at position x,y on LCD
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#define LCD_DD4   PIN_D4
#define LCD_DD5   PIN_D5
#define LCD_DD6   PIN_D6
#define LCD_DD7   PIN_D7

#define LCD_E     PIN_D3
#define LCD_RS    PIN_D2


//=======================================

#define lcd_type 2        // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40 // LCD RAM address for the 2nd line


int8 const LCD_INIT_STRING[4] =
{
 0x20 | (lcd_type << 2), // Func set: 4-bit, 2 lines, 5x8 dots
 0xc,                    // Display on
 1,                      // Clear display
```

```
 6                    // Increment cursor
 };


//-----------------------------------
void lcd_send_nibble(int8 nibble)
{
// Note:  !! converts an integer expression
// to a boolean (1 or 0).
 output_bit(LCD_DD4, !!(nibble & 1));
 output_bit(LCD_DD5, !!(nibble & 2));
 output_bit(LCD_DD6, !!(nibble & 4));
 output_bit(LCD_DD7, !!(nibble & 8));

 delay_cycles(1);
 output_high(LCD_E);
 delay_us(2);
 output_low(LCD_E);
}

//-------------------------------------
// This sub-routine is only called by lcd_read_byte().
// It's not a stand-alone routine.  For example, the
// R/W signal is set high by lcd_read_byte() before
// this routine is called.

#ifdef USE_LCD_RW
int8 lcd_read_nibble(void)
{
int8 retval;
// Create bit variables so that we can easily set
// individual bits in the retval variable.
#bit retval_0 = retval.0
#bit retval_1 = retval.1
#bit retval_2 = retval.2
#bit retval_3 = retval.3

retval = 0;

output_high(LCD_E);
delay_cycles(1);

retval_0 = input(LCD_DD4);
retval_1 = input(LCD_DD5);
retval_2 = input(LCD_DD6);
retval_3 = input(LCD_DD7);

output_low(LCD_E);

return(retval);
}
#endif

//---------------------------------------
// Read a byte from the LCD and return it.

#ifdef USE_LCD_RW
int8 lcd_read_byte(void)
{
int8 low;
```

```
int8 high;

output_high(LCD_RW);
delay_cycles(1);

high = lcd_read_nibble();

low = lcd_read_nibble();

return( (high<<4) | low);
}
#endif

//-----------------------------------------
// Send a byte to the LCD.
void lcd_send_byte(int8 address, int8 n)
{
output_low(LCD_RS);

#ifdef USE_LCD_RW
while(bit_test(lcd_read_byte(),7)) ;
#else
delay_us(60);
#endif

if(address)
   output_high(LCD_RS);
else
   output_low(LCD_RS);

 delay_cycles(1);

#ifdef USE_LCD_RW
output_low(LCD_RW);
delay_cycles(1);
#endif

output_low(LCD_E);

lcd_send_nibble(n >> 4);
lcd_send_nibble(n & 0xf);
}

//-----------------------------
void lcd_init(void)
{
int8 i;

output_low(LCD_RS);

#ifdef USE_LCD_RW
output_low(LCD_RW);
#endif

output_low(LCD_E);

delay_ms(15);

for(i=0 ;i < 3; i++)
   {
```

```
    lcd_send_nibble(0x03);
    delay_ms(5);
   }

lcd_send_nibble(0x02);

for(i=0; i < sizeof(LCD_INIT_STRING); i++)
   {
    lcd_send_byte(0, LCD_INIT_STRING[i]);

    // If the R/W signal is not used, then
    // the busy bit can't be polled.  One of
    // the init commands takes longer than
    // the hard-coded delay of 60 us, so in
    // that case, lets just do a 5 ms delay
    // after all four of them.
    #ifndef USE_LCD_RW
    delay_ms(5);
    #endif
   }

}

//-----------------------------

void lcd_gotoxy(int8 x, int8 y)
{
int8 address;

if(y != 1)
   address = lcd_line_two;
else
   address=0;

address += x-1;
lcd_send_byte(0, 0x80 | address);
}

//-----------------------------
void lcd_putc(char c)
{
 switch(c)
   {
    case '\f':
      lcd_send_byte(0,1);
      delay_ms(2);
      break;

    case '\n':
       lcd_gotoxy(1,2);
       break;

    case '\b':
       lcd_send_byte(0,0x10);
       break;

    default:
       lcd_send_byte(1,c);
       break;
   }
```

```
}

//------------------------------
#ifdef USE_LCD_RW
char lcd_getc(int8 x, int8 y)
{
char value;

lcd_gotoxy(x,y);

// Wait until busy flag is low.
while(bit_test(lcd_read_byte(),7));

output_high(LCD_RS);
value = lcd_read_byte();
output_low(lcd_RS);

return(value);
}
#endif
```

A.2. DRIVER “miGLCD.c”

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                                  miGLCD.c
////                                             Driver for  GLCD module
////
//// This file contains drivers for using a LGM12864B-FL-YBW with
//// a KS0108 display controller. This GLCD is 128 by 64 pixels.
//// The driver treats the upper left pixel as (0,0).
////
//// Use #define FAST_GLCD if the target chip has at least 1k of RAM
//// to decrease the time it takes to update the display.
//// glcd_update() must then be called to update the display after
//// changing the pixel information.
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
//// LCD Pin connections:
//// (These can be changed as needed in the following defines).
////  * 1: Chip Select 1 is connected to B1
////  * 2: Chip Select 2 is connected to B0
////  * 3: VSS is connected to GND
////  * 4: VDD is connected to +5V
////  * 5: V0  - LCD operating voltage (Constrast adjustment)
////  * 6: R/S - Register select is connected to B2
////  * 7: R/W - Read or Write is connected to B3
////  * 8: Enable is connected to B4
////  *9-16: Data Bus 0 to 7 is connected to port D
////  *17: Reset is connected to B5
////  *18: Negative voltage is also connected to the 20k Ohm POT
////  *19: Positive voltage for LED backlight is connected to +5V
////  *20: Negavtive voltage for LED backlight is connected to GND
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////
////  glcd_init(mode)
////    * Must be called before any other function.
////      - mode can be ON or OFF to turn the GLCD on or off
////
////  glcd_update()
////    * Write the display data stored in RAM to the GLCD
////    * Only available if FAST_GLCD is defined
////
////  glcd_pixel(x,y,color)
////    * Sets the pixel to the given color.
////      - color can be ON or OFF
////
////  glcd_fillScreen(color)
////    * Fills the entire GLCD with the given color.
////      - color can be ON or OFF
////
////  glcd_line(x1,y1,x2,y2,color)
////    * Draws a line from the first point to the second point
////      with the given color.
////      - color can be ON or OFF
////
////  glcd_rect(x1,y1,x2,y2,fill,color)
////    * Draws a rectangle with upper left point (x1,y1) and lower
////      right point (x2,y2).
////      - fill can be ON or OFF
////      - color can be ON or OFF
////
////  glcd_bar(x1,y1,x2,y2,width,color)
////    * Draws a bar (wide line) from the first point to the
////      second point.
////      - width is the number of pixels wide
////      - color is ON or OFF
////
////  glcd_circle(x,y,radius,fill,color)
////    * Draws a circle with center at (x,y)
////      - fill can be YES or NO
////      - color can be ON or OFF
////
////  glcd_text57(x,y,textptr,size,color)
////    * Write the null terminated text pointed to by textptr with
////      the upper left coordinate of the first character at (x,y).
////      Characters are 5 pixels wide and 7 pixels tall.
////      - size is an integer that scales the size of the text
////      - color is ON or OFF
////    * Note - The given text is character wrapped. If this
////      function is used on a different size display, then change
////      the GLCD_WIDTH define appropriately.
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef HDM64GS12
#define HDM64GS12

#ifndef GLCD_WIDTH
#define GLCD_WIDTH   128
#endif

#ifndef GLCD_CS1
#define GLCD_CS1     PIN_B1   // Chip Selection 1
```

```
#endif

#ifndef GLCD_CS2
#define GLCD_CS2     PIN_B0   // Chip Selection 2
#endif

#ifndef GLCD_DI
#define GLCD_DI      PIN_B2   // Data or Instruction input
#endif

#ifndef GLCD_RW
#define GLCD_RW      PIN_B3   // Read/Write
#endif

#ifndef GLCD_E
#define GLCD_E       PIN_B4   // Enable
#endif

#ifndef GLCD_RST
#define GLCD_RST     PIN_B5   // Reset
#endif

#define GLCD_LEFT    0
#define GLCD_RIGHT   1

#ifndef ON
#define ON           1
#endif

#ifndef OFF
#define OFF          0
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function Prototypes
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void glcd_init(int1 mode);
void glcd_update();
void glcd_pixel(int8 x, int8 y, int1 color);
void glcd_fillScreen(int1 color);
void glcd_writeByte(int1 side, BYTE data);
BYTE glcd_readByte(int1 side);
void glcd_line(int x1, int y1, int x2, int y2, int1 color);
void glcd_rect(int x1, int y1, int x2, int y2, int fill, int1 color);
void glcd_bar(int x1, int y1, int x2, int y2, int width, int1 color);
void glcd_circle(int x, int y, int radius, int1 fill, int1 color);
void glcd_text57(int x, int y, char* textptr, int size, int1 color);
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


const BYTE TEXT[51][5] ={0x00, 0x00, 0x00, 0x00, 0x00, // SPACE
                         0x00, 0x00, 0x5F, 0x00, 0x00, // !
                         0x00, 0x03, 0x00, 0x03, 0x00, // "
                         0x14, 0x3E, 0x14, 0x3E, 0x14, // #
                         0x24, 0x2A, 0x7F, 0x2A, 0x12, // $
                         0x43, 0x33, 0x08, 0x66, 0x61, // %
                         0x36, 0x49, 0x55, 0x22, 0x50, // &
                         0x00, 0x05, 0x03, 0x00, 0x00, // '
                         0x00, 0x1C, 0x22, 0x41, 0x00, // (
                         0x00, 0x41, 0x22, 0x1C, 0x00, // )
```

```
                    0x14, 0x08, 0x3E, 0x08, 0x14, // *
                    0x08, 0x08, 0x3E, 0x08, 0x08, // +
                    0x00, 0x50, 0x30, 0x00, 0x00, // ,
                    0x08, 0x08, 0x08, 0x08, 0x08, // -
                    0x00, 0x60, 0x60, 0x00, 0x00, // .
                    0x20, 0x10, 0x08, 0x04, 0x02, // /
                    0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
                    0x04, 0x02, 0x7F, 0x00, 0x00, // 1
                    0x42, 0x61, 0x51, 0x49, 0x46, // 2
                    0x22, 0x41, 0x49, 0x49, 0x36, // 3
                    0x18, 0x14, 0x12, 0x7F, 0x10, // 4
                    0x27, 0x45, 0x45, 0x45, 0x39, // 5
                    0x3E, 0x49, 0x49, 0x49, 0x32, // 6
                    0x01, 0x01, 0x71, 0x09, 0x07, // 7
                    0x36, 0x49, 0x49, 0x49, 0x36, // 8
                    0x26, 0x49, 0x49, 0x49, 0x3E, // 9
                    0x00, 0x36, 0x36, 0x00, 0x00, // :
                    0x00, 0x56, 0x36, 0x00, 0x00, // ;
                    0x08, 0x14, 0x22, 0x41, 0x00, // <
                    0x14, 0x14, 0x14, 0x14, 0x14, // =
                    0x00, 0x41, 0x22, 0x14, 0x08, // >
                    0x02, 0x01, 0x51, 0x09, 0x06, // ?
                    0x3E, 0x41, 0x59, 0x55, 0x5E, // @
                    0x7E, 0x09, 0x09, 0x09, 0x7E, // A
                    0x7F, 0x49, 0x49, 0x49, 0x36, // B
                    0x3E, 0x41, 0x41, 0x41, 0x22, // C
                    0x7F, 0x41, 0x41, 0x41, 0x3E, // D
                    0x7F, 0x49, 0x49, 0x49, 0x41, // E
                    0x7F, 0x09, 0x09, 0x09, 0x01, // F
                    0x3E, 0x41, 0x41, 0x49, 0x3A, // G
                    0x7F, 0x08, 0x08, 0x08, 0x7F, // H
                    0x00, 0x41, 0x7F, 0x41, 0x00, // I
                    0x30, 0x40, 0x40, 0x40, 0x3F, // J
                    0x7F, 0x08, 0x14, 0x22, 0x41, // K
                    0x7F, 0x40, 0x40, 0x40, 0x40, // L
                    0x7F, 0x02, 0x0C, 0x02, 0x7F, // M
                    0x7F, 0x02, 0x04, 0x08, 0x7F, // N
                    0x3E, 0x41, 0x41, 0x41, 0x3E, // O
                    0x7F, 0x09, 0x09, 0x09, 0x06, // P
                    0x1E, 0x21, 0x21, 0x21, 0x5E, // Q
                    0x7F, 0x09, 0x09, 0x09, 0x76};// R


const BYTE TEXT2[44][5]={0x26, 0x49, 0x49, 0x49, 0x32, // S
                    0x01, 0x01, 0x7F, 0x01, 0x01, // T
                    0x3F, 0x40, 0x40, 0x40, 0x3F, // U
                    0x1F, 0x20, 0x40, 0x20, 0x1F, // V
                    0x7F, 0x20, 0x10, 0x20, 0x7F, // W
                    0x41, 0x22, 0x1C, 0x22, 0x41, // X
                    0x07, 0x08, 0x70, 0x08, 0x07, // Y
                    0x61, 0x51, 0x49, 0x45, 0x43, // Z
                    0x00, 0x7F, 0x41, 0x00, 0x00, // [
                    0x02, 0x04, 0x08, 0x10, 0x20, // \
                    0x00, 0x00, 0x41, 0x7F, 0x00, // ]
                    0x04, 0x02, 0x01, 0x02, 0x04, // ^
                    0x40, 0x40, 0x40, 0x40, 0x40, // _
                    0x00, 0x01, 0x02, 0x04, 0x00, // `
                    0x20, 0x54, 0x54, 0x54, 0x78, // a
                    0x7F, 0x44, 0x44, 0x44, 0x38, // b
                    0x38, 0x44, 0x44, 0x44, 0x44, // c
```

```
                    0x38, 0x44, 0x44, 0x44, 0x7F, // d
                    0x38, 0x54, 0x54, 0x54, 0x18, // e
                    0x04, 0x04, 0x7E, 0x05, 0x05, // f
                    0x08, 0x54, 0x54, 0x54, 0x3C, // g
                    0x7F, 0x08, 0x04, 0x04, 0x78, // h
                    0x00, 0x44, 0x7D, 0x40, 0x00, // i
                    0x20, 0x40, 0x44, 0x3D, 0x00, // j
                    0x7F, 0x10, 0x28, 0x44, 0x00, // k
                    0x00, 0x41, 0x7F, 0x40, 0x00, // l
                    0x7C, 0x04, 0x78, 0x04, 0x78, // m
                    0x7C, 0x08, 0x04, 0x04, 0x78, // n
                    0x38, 0x44, 0x44, 0x44, 0x38, // o
                    0x7C, 0x14, 0x14, 0x14, 0x08, // p
                    0x08, 0x14, 0x14, 0x14, 0x7C, // q
                    0x00, 0x7C, 0x08, 0x04, 0x04, // r
                    0x48, 0x54, 0x54, 0x54, 0x20, // s
                    0x04, 0x04, 0x3F, 0x44, 0x44, // t
                    0x3C, 0x40, 0x40, 0x20, 0x7C, // u
                    0x1C, 0x20, 0x40, 0x20, 0x1C, // v
                    0x3C, 0x40, 0x30, 0x40, 0x3C, // w
                    0x44, 0x28, 0x10, 0x28, 0x44, // x
                    0x0C, 0x50, 0x50, 0x50, 0x3C, // y
                    0x44, 0x64, 0x54, 0x4C, 0x44, // z
                    0x00, 0x08, 0x36, 0x41, 0x41, // {
                    0x00, 0x00, 0x7F, 0x00, 0x00, // |
                    0x41, 0x41, 0x36, 0x08, 0x00, // }
                    0x02, 0x01, 0x02, 0x04, 0x02};// ~


#ifdef FAST_GLCD
struct
{
  int8 left[512];
  int8 right[512];
} displayData;
#endif


// Purpose:      Initialize the LCD.
//               Call before using any other LCD function.
// Inputs:       OFF - Turns the LCD off
//               ON  - Turns the LCD on
void glcd_init(int1 mode)
{
  // Initialze some pins
  output_high(GLCD_RST);
  output_low(GLCD_E);
  output_low(GLCD_CS1);
  output_low(GLCD_CS2);

  output_low(GLCD_DI);                // Set for instruction
  glcd_writeByte(GLCD_LEFT,  0xC0);   // Specify first RAM line at the top
  glcd_writeByte(GLCD_RIGHT, 0xC0);   //   of the screen
  glcd_writeByte(GLCD_LEFT,  0x40);   // Set the column address to 0
  glcd_writeByte(GLCD_RIGHT, 0x40);
  glcd_writeByte(GLCD_LEFT,  0xB8);   // Set the page address to 0
  glcd_writeByte(GLCD_RIGHT, 0xB8);

  if(mode == ON)
  {
```

```
    glcd_writeByte(GLCD_LEFT,  0x3F); // Turn the display on
    glcd_writeByte(GLCD_RIGHT, 0x3F);
  }
  else
  {
    glcd_writeByte(GLCD_LEFT,  0x3E); // Turn the display off
    glcd_writeByte(GLCD_RIGHT, 0x3E);
  }

  glcd_fillScreen(OFF);              // Clear the display

  #ifdef FAST_GLCD
  glcd_update();
  #endif
}


// Purpose:    Update the LCD with data from the display arrays
#ifdef FAST_GLCD
void glcd_update()
{
  int8 i, j;
  int8 *p1, *p2;

  p1 = displayData.left;
  p2 = displayData.right;

  // Loop through the vertical pages
  for(i = 0; i < 8; ++i)
  {
    output_low(GLCD_DI);                  // Set for instruction
    glcd_writeByte(GLCD_LEFT, 0x40);          // Set horizontal address to 0
    glcd_writeByte(GLCD_RIGHT, 0x40);
    glcd_writeByte(GLCD_LEFT, i | 0xB8);      // Set page address
    glcd_writeByte(GLCD_RIGHT, i | 0xB8);
    output_high(GLCD_DI);                  // Set for data

    // Loop through the horizontal sections
    for(j = 0; j < 64; ++j)
    {
      glcd_writeByte(GLCD_LEFT, *p1++);      // Turn pixels on or off
      glcd_writeByte(GLCD_RIGHT, *p2++);     // Turn pixels on or off
    }
  }
}
#endif


// Purpose:    Turn a pixel on a graphic LCD on or off
// Inputs:     1) x - the x coordinate of the pixel
//             2) y - the y coordinate of the pixel
//             3) color - ON or OFF
void glcd_pixel(int8 x, int8 y, int1 color)
#ifdef FAST_GLCD
{
  int8* p;
  int16 temp;
  temp =  y/8;
  temp *= 64;
  temp += x;
```

```
  if(x > 63)
  {
    p = displayData.right + temp - 64;
  }
  else
  {
    p = displayData.left + temp;
  }

  if(color)
  {
    bit_set(*p, y%8);
  }
  else
  {
    bit_clear(*p, y%8);
  }
}
#else
{
  BYTE data;
  int1 side = GLCD_LEFT;  // Stores which chip to use on the LCD

  if(x > 63)             // Check for first or second display area
  {
    x -= 64;
    side = GLCD_RIGHT;
  }

  output_low(GLCD_DI);                      // Set for instruction
  bit_clear(x,7);                           // Clear the MSB. Part of an instruction code
  bit_set(x,6);                             // Set bit 6. Also part of an instruction code
  glcd_writeByte(side, x);                  // Set the horizontal address
  glcd_writeByte(side, (y/8 & 0xBF) | 0xB8);   // Set the vertical page address
  output_high(GLCD_DI);                     // Set for data
  glcd_readByte(side);                      // Need two reads to get data
  data = glcd_readByte(side);               //  at new address

  if(color == ON)
    bit_set(data, y%8);       // Turn the pixel on
  else                        // or
    bit_clear(data, y%8);     // turn the pixel off

  output_low(GLCD_DI);         // Set for instruction
  glcd_writeByte(side, x);     // Set the horizontal address
  output_high(GLCD_DI);        // Set for data
  glcd_writeByte(side, data);  // Write the pixel data
}
#endif


// Purpose:    Fill the LCD screen with the passed in color
// Inputs:     ON  - turn all the pixels on
//             OFF - turn all the pixels off
void glcd_fillScreen(int1 color)
#ifdef FAST_GLCD
{
  int8  data;
  int8  *p1, *p2;
```

```
  int16 i;

  p1 = displayData.left;
  p2 = displayData.right;
  data = 0xFF * color;

  for(i=0; i<512; ++i)
  {
    *p1++ = data;
    *p2++ = data;
  }
}
#else
{
  int8 i, j;

  // Loop through the vertical pages
  for(i = 0; i < 8; ++i)
  {
    output_low(GLCD_DI);                    // Set for instruction
    glcd_writeByte(GLCD_LEFT, 0b01000000);    // Set horizontal address to 0
    glcd_writeByte(GLCD_RIGHT, 0b01000000);
    glcd_writeByte(GLCD_LEFT, i | 0b10111000);// Set page address
    glcd_writeByte(GLCD_RIGHT, i | 0b10111000);
    output_high(GLCD_DI);                   // Set for data

    // Loop through the horizontal sections
    for(j = 0; j < 64; ++j)
    {
      glcd_writeByte(GLCD_LEFT, 0xFF*color);  // Turn pixels on or off
      glcd_writeByte(GLCD_RIGHT, 0xFF*color); // Turn pixels on or off
    }
  }
}
#endif


// Purpose:    Write a byte of data to the specified chip
// Inputs:     1) chipSelect - which chip to write the data to
//             2) data - the byte of data to write
void glcd_writeByte(int1 side, BYTE data)
{
  if(side)                  // Choose which side to write to
    output_high(GLCD_CS2);
  else
    output_high(GLCD_CS1);

  output_low(GLCD_RW);       // Set for writing
  output_d(data);            // Put the data on the port
  delay_cycles(1);
  output_high(GLCD_E);       // Pulse the enable pin
  delay_cycles(5);
  output_low(GLCD_E);

  output_low(GLCD_CS1);      // Reset the chip select lines
  output_low(GLCD_CS2);
}


// Purpose:    Reads a byte of data from the specified chip
```

```
// Ouputs:     A byte of data read from the chip
BYTE glcd_readByte(int1 side)
{
   BYTE data;                // Stores the data read from the LCD

   set_tris_d(0xFF);         // Set port d to input
   output_high(GLCD_RW);     // Set for reading

   if(side)                  // Choose which side to write to
      output_high(GLCD_CS2);
   else
      output_high(GLCD_CS1);

   delay_cycles(1);
   output_high(GLCD_E);      // Pulse the enable pin
   delay_cycles(4);
   data = input_d();         // Get the data from the display's output register
   output_low(GLCD_E);

   output_low(GLCD_CS1);     // Reset the chip select lines
   output_low(GLCD_CS2);
   return data;              // Return the read data
}

#endif



// Purpose:       Draw a line on a graphic LCD using Bresenham's
//                line drawing algorithm
// Inputs:        (x1, y1) - the start coordinate
//                (x2, y2) - the end coordinate
//                color - ON or OFF
// Dependencies:  glcd_pixel()
void glcd_line(int x1, int y1, int x2, int y2, int1 color)
{
   signed int  x, y, addx, addy, dx, dy;
   signed long P;
   int i;
   dx = abs((signed int)(x2 - x1));
   dy = abs((signed int)(y2 - y1));
   x = x1;
   y = y1;

   if(x1 > x2)
      addx = -1;
   else
      addx = 1;
   if(y1 > y2)
      addy = -1;
   else
      addy = 1;

   if(dx >= dy)
   {
      P = 2*dy - dx;

      for(i=0; i<=dx; ++i)
      {
         glcd_pixel(x, y, color);
```

```
      if(P < 0)
      {
        P += 2*dy;
        x += addx;
      }
      else
      {
        P += 2*dy - 2*dx;
        x += addx;
        y += addy;
      }
    }
  }
  else
  {
    P = 2*dx - dy;

    for(i=0; i<=dy; ++i)
    {
      glcd_pixel(x, y, color);

      if(P < 0)
      {
        P += 2*dx;
        y += addy;
      }
      else
      {
        P += 2*dx - 2*dy;
        x += addx;
        y += addy;
      }
    }
  }
}



// Purpose:      Draw a rectangle on a graphic LCD
// Inputs:       (x1, y1) - the start coordinate
//               (x2, y2) - the end coordinate
//               fill  - YES or NO
//               color - ON or OFF
// Dependencies:  glcd_pixel(), glcd_line()
void glcd_rect(int x1, int y1, int x2, int y2, int fill, int1 color)
{
  if(fill)
  {
    int y, ymax;                        // Find the y min and max
    if(y1 < y2)
    {
      y = y1;
      ymax = y2;
    }
    else
    {
      y = y2;
      ymax = y1;
    }
```

```
    for(; y<=ymax; ++y)                  // Draw lines to fill the rectangle
      glcd_line(x1, y, x2, y, color);
  }
  else
  {
    glcd_line(x1, y1, x2, y1, color);      // Draw the 4 sides
    glcd_line(x1, y2, x2, y2, color);
    glcd_line(x1, y1, x1, y2, color);
    glcd_line(x2, y1, x2, y2, color);
  }
}



// Purpose:       Draw a bar (wide line) on a graphic LCD
// Inputs:        (x1, y1) - the start coordinate
//                (x2, y2) - the end coordinate
//                width  - The number of pixels wide
//                color - ON or OFF
void glcd_bar(int x1, int y1, int x2, int y2, int width, int1 color)
{
  signed int  x, y, addx, addy, j;
  signed long P, dx, dy, c1, c2;
  int i;
  dx = abs((signed int)(x2 - x1));
  dy = abs((signed int)(y2 - y1));
  x = x1;
  y = y1;
  c1 = -dx*x1 - dy*y1;
  c2 = -dx*x2 - dy*y2;

  if(x1 > x2)
  {
    addx = -1;
    c1 = -dx*x2 - dy*y2;
    c2 = -dx*x1 - dy*y1;
  }
  else
    addx = 1;
  if(y1 > y2)
  {
    addy = -1;
    c1 = -dx*x2 - dy*y2;
    c2 = -dx*x1 - dy*y1;
  }
  else
    addy = 1;

  if(dx >= dy)
  {
    P = 2*dy - dx;

    for(i=0; i<=dx; ++i)
    {
      for(j=-(width/2); j<width/2+width%2; ++j)
      {
        if(dx*x+dy*(y+j)+c1 >= 0 && dx*x+dy*(y+j)+c2 <=0)
          glcd_pixel(x, y+j, color);
      }
```

```
        if(P < 0)
        {
          P += 2*dy;
          x += addx;
        }
        else
        {
          P += 2*dy - 2*dx;
          x += addx;
          y += addy;
        }
      }
    }
    else
    {
      P = 2*dx - dy;

      for(i=0; i<=dy; ++i)
      {
        if(P < 0)
        {
          P += 2*dx;
          y += addy;
        }
        else
        {
          P += 2*dx - 2*dy;
          x += addx;
          y += addy;
        }
        for(j=-(width/2); j<width/2+width%2; ++j)
        {
          if(dx*x+dy*(y+j)+c1 >= 0 && dx*x+dy*(y+j)+c2 <=0)
            glcd_pixel(x+j, y, color);
        }
      }
    }
}


// Purpose:       Draw a circle on a graphic LCD
// Inputs:        (x,y) - the center of the circle
//                radius - the radius of the circle
//                fill - ON or OFF
//                color - ON or OFF
void glcd_circle(int x, int y, int radius, int1 fill, int1 color)
{
  signed int a, b, P;
  a = 0;
  b = radius;
  P = 1 - radius;

  do
  {
    if(fill)
    {
      glcd_line(x-a, y+b, x+a, y+b, color);
      glcd_line(x-a, y-b, x+a, y-b, color);
      glcd_line(x-b, y+a, x+b, y+a, color);
      glcd_line(x-b, y-a, x+b, y-a, color);
```

```
    }
    else
    {
      glcd_pixel(a+x, b+y, color);
      glcd_pixel(b+x, a+y, color);
      glcd_pixel(x-a, b+y, color);
      glcd_pixel(x-b, a+y, color);
      glcd_pixel(b+x, y-a, color);
      glcd_pixel(a+x, y-b, color);
      glcd_pixel(x-a, y-b, color);
      glcd_pixel(x-b, y-a, color);
    }

    if(P < 0)
      P+= 3 + 2*a++;
    else
      P+= 5 + 2*(a++ - b--);
  } while(a <= b);
}



// Purpose:      Write text on a graphic LCD
// Inputs:       (x,y) - The upper left coordinate of the first letter
//               textptr - A pointer to an array of text to display
//               size - The size of the text: 1 = 5x7, 2 = 10x14, ...
//               color - ON or OFF
void glcd_text57(int x, int y, char* textptr, int size, int1 color)
{
  int i, j, k, l, m;                  // Loop counters
  BYTE pixelData[5];                  // Stores character data

  for(i=0; textptr[i] != '\0'; ++i, ++x) // Loop through the passed string
  {
    if(textptr[i] < 'S') // Checks if the letter is in the first text array
      memcpy(pixelData, TEXT[textptr[i]-' '], 5);
    else if(textptr[i] <= '~') // Check if the letter is in the second array
      memcpy(pixelData, TEXT2[textptr[i]-'S'], 5);
    else
      memcpy(pixelData, TEXT[0], 5);   // Default to space

    if(x+5*size >= GLCD_WIDTH)          // Performs character wrapping
    {
      x = 0;                            // Set x at far left position
      y += 7*size + 1;                  // Set y at next position down
    }
    for(j=0; j<5; ++j, x+=size)         // Loop through character byte data
    {
      for(k=0; k<7*size; ++k)           // Loop through the vertical pixels
      {
        if(bit_test(pixelData[j], k)) // Check if the pixel should be set
        {
          for(l=0; l<size; ++l)      // The next two loops change the
          {                          // character's size
            for(m=0; m<size; ++m)
            {
              glcd_pixel(x+m, y+k*size+l, color); // Draws the pixel
            }
          }
        }
```

```
      }
    }
  }
}
```

ANEXO B. ANÁLISIS TEMPORAL DE LA APLICACIÓN CRONOMETRO.C

Con el fin de profundizar más en la aplicación desarrollada cronometro.c, se considera interesante realizar un análisis temporal de las funciones principales implicadas en la escritura del módulo GLCD.

En el presente anexo se muestra un estudio pormenorizado, en el cual, se explica detalladamente el tiempo invertido en la duración de dichas funciones.

B.1. FUNCIONES

A continuación se realiza una breve descripción de las funciones empleadas en el desarrollo de la aplicación cronometro.c. Parte de las funciones descritas a continuación se encuentran en el manual de CCS C Compiler [14], mientras que el resto se encuentran en el driver “miGLCD.c” presente en el Anexo A del presente Proyecto Fin de Carrera.

- delay_ms

void delay_ms (time)

La función *delay_ms* realiza retardos del valor especificado en *time*.

time Valor de tiempo en milisegundos.

- delay_us

```
void delay_us (time)
```

La función *delay_us* realiza retardos del valor especificado en *time*.

time Valor de tiempo en microsegundos.

- glcd_bar

```
void glcd_bar (int x1, int y1, int x2, int y2, int width, int1 color)
```

La función *glcd_bar* dibuja una barra (línea con grosor) en el módulo GLCD.

(x1, y1) Coordenadas de comienzo de la línea.

(x2, y2) Coordenadas de fin de la línea.

width Número de píxeles, grosor de la línea.

color

ON Visualizar píxel.

OFF No visualizar píxel.

- glcd_init

```
void glcd_init (int 1 mode)
```

La función *glcd_init* inicializa el módulo gráfico LCD.

mode

OFF Apaga el GLCD.

ON Enciende el GLCD.

- glcd_line

```
void glcd_line(int x1, int y1, int x2, int y2, int1 color)
```

La función *glcd_line* dibuja una línea en el módulo GLCD.

(x1, y1) Coordenadas de comienzo de la línea.

(x2, y2) Coordenadas de fin de la línea.

color

ON Visualizar píxel.

OFF No visualizar píxel.

- glcd_pixel

```
void glcd_pixel(int8 x, int8 y, int1 color)
```

La función *glcd_pixel* dibuja o borra un pixel del modulo GLCD.

x Coordenada x del píxel.

y Coordenada y del píxel.

color

ON Visualizar píxel.

OFF No visualizar píxel.

- glcd_rect

```
void glcd_rect (int x1, int y1, int x2, int y2, int fill, int1 color)
```

La función *glcd_rect* dibuja un rectángulo en el display GLCD.

(x1, y1) Coordenadas de comienzo de la línea.

(x2, y2) Coordenadas de fin de la línea.

fill Relleno

ON Con relleno.

OFF Sin relleno.

color

ON Visualizar píxel.

OFF No visualizar píxel.

- glcd_text57

void glcd_text57 (int x, int y, char* textptr, int size, int1 color)

La función *glcd_text57* escribe texto en el módulo GLCD.

(x, y) Coordenada superior izquierda de la primera letra.

textptr Texto a mostrar en el display GLCD.

size Tamaño del texto a mostrar.

1 = 5 x 7 píxeles

2 = 10 x 14 píxeles

…

color

ON Visualizar píxel.

OFF No visualizar píxel.

- sprintf

void sprintf (string, cstring, value)

La función *sprintf* almacena la variable *value* en *string*. El formato está relacionado con el argumento que se pone dentro de la cadena *cstring.*

El caracter % se pone dentro de *cstring* para indicar un valor variable, seguido de uno o más caracteres que dan formato al tipo de información a representar.

string Array de caracteres.

cstring Cadena constante.

value Variable.

B.2. ANÁLISIS TEMPORAL

Una vez descritas las funciones empleadas en el desarrollo de la aplicación cronometro.c, se procede al análisis de la duración de las funciones que se encuentran en dicha aplicación.

A continuación se muestra el fragmento de código desarrollado para la visualización de las décimas de segundo en el módulo GLCD.

```
…
    for(i=0;i<10;i++)                //Décimas
       {
            sprintf(a, "%d",i );              // Convierte el entero i en texto
            glcd_rect(76,25, 87, 35, ON, OFF);              // Borra el numero anterior
            glcd_text57(76,25,a,1,ON);          //Muestra las décimas
            delay_us(19321);

            if (i==0)
            {
                borrarMu();                     //Borra la imagen anterior
                displayMu01();                  //Muestra la primera imágen
            }
            if (i==3)
            {
                borrarMu();                     //Borra la imagen anterior
```

```
                displayMu02();                  //Muestra la segunda imágen
            }
            if (i==6)
            {
                borrarMu();                     //Borra la imagen anterior
                displayMu03();                  //Muestra la tercera imágen
            }
            if (i==9)
            {
                borrarMu();                     //Borra la imagen anterior
                displayMu04();                  //Muestra la cuarta imagen
            }
            if(input (ONOFF1))              //Si se pulsa RC1
            {
                while (!input (ONOFF0))   //Mientras no se haya pulsado RC0 se espera
                {
                    if(input (ONOFF2))     //Si se pulsa RC2 reinicia la cuenta
                    {
                        glcd_rect(40,25, 87, 35, ON, OFF);
                        glcd_text57(40,25,segT,1,ON);
                        glcd_rect(10,21, 32, 41, ON, OFF);
                        glcd_text57(10,21,cer00,2,ON);
                        i=0;
                        j=1;
                        k=1;
                        h=0;
                    }
                }
            }

        }          //Cierra el for con i
…
```

Para el análisis de la duración de cada décima, se utiliza la herramienta “stopwatch” de MPLAB.

Dicha herramienta es muy útil a la hora de realizar una medición de tiempos de manera rápida y sencilla, puesto que solamente es necesario indicar el fragmento de código que se desea medir y la frecuencia de reloj del microcontrolador, obteniendo el tiempo invertido en ello.

A continuación se muestra en la Tabla B.1, la duración en milisegundos de las décimas mostradas en el display GLCD.

Mostrar décima	Duración (ms.)
0	152,088
1	55,8835
2	57,1385
3	151,1645

4	57,131
5	57,8535
6	156,196
7	56,356
8	57,871
9	153,834
Salto al terminar el bucle	0,004

Total	**955,52**

Tabla B.1: Mostrar décimas

La duración de cada décima de segundo también se puede calcular manualmente, sabiendo la duración de las funciones presentes en dicho fragmento. A continuación se muestran las distintas funciones y el tiempo empleado.

Función	**Duración (ms.)**
Volver al for	0.001
for(i=0;i<10;i++)	0.0025
sprintf(a, “%d”,i);	0,044
glcd_rect(76,25, 87, 35, ON, OFF);	32,6425
glcd_text57(76,25,a,1,ON); * cuando a es	
0	6,344
1	3,8615
2	5,1165
3	5,1065
4	5,109
5	5,8315
6	5,8515
7	4,334
8	5,849
9	5,8465
delay_us(19321);	19,3185

if(i==0)	
entra	0,0015
salta	0,002
if(i==3) o if(i==6) o if(i==9)	
entra	0,002
salta	0,0025
borrarMu	82,9165
displayMu01	10,807
displayMu02	11,12
displayMu03	15,4065
displayMu04	13,0495
if(input(ONOFF1))	
salta	0,004

Tabla B.2: Duración de las funciones

* Nota: Como se ha comentado en el Capítulo 6, la duración de la función de escritura glcd_text57, depende del número (dígito) a representar en el GLCD, y por tanto la escritura de las décimas es distinto, ya que cada dígito se representa con un número de píxeles diferente.

Por ejemplo, el tiempo empleado para mostrar la décima “0” es el que resulta de sumar la duración de las distintas funciones que se ejecutan:

Analizando el código y sabiendo que i=0, se observa que lo primero que se ejecuta es el *for (i=0;i<10;i++),* entrando en el bucle y convirtiendo el entero en texto (mediante la función *sprintf*). Posteriormente, se dibuja un rectángulo para borrar las décimas anteriores, se muestra las décimas utilizando la función *glcd_text57* y se realiza un retardo mediante la función *delay_us*. Como i=0, se ejecuta (entra) el primer *if*, se borra la imagen anterior y se muestra la simulación de la persona en otra posición mediante *displayMu0*1. Al terminar el *if*, se comprueba si i= 3, 6 ó 9, y puesto que i=0, no se entra en ninguno de ellos y los salta. El siguiente *if*, es el correspondiente a la comprobación de si ha sido pulsado RC1, y por tanto también es saltado.

Función	Duración
for(i=0;i<10;i++)	0.0025 ms
sprintf(a, “%d”,i);	0,044 ms
glcd_rect(76,25, 87, 35, ON, OFF);	32,6425 ms
glcd_text57(76,25,a,1,ON);	6,344 ms
delay_us(19321);	19,3185 ms
if(i==0)	
entra	0,0015 ms
if(i==3) o if(i==6) o if(i==9)	
salta	3*0,0025 ms
borrarMu	82,9165 ms
displayMu01	10,807 ms
if(input(ONOFF1))	
salta	0,004 ms

Suma	152,088 ms

Tabla B.3: Duración de las funciones cuando i=0

Procediendo del mismo modo, se puede comprobar la duración de cada una de las restantes décimas de segundo mostradas en la Tabla B.1.

Si se detuviese el estudio temporal aquí, se supondría que en cada segundo el cronómetro se adelantaría 1000 - 955.52 = 44.48ms. Esto no es cierto, ya que se ha de tener en cuenta que la escritura de los segundos en el GLCD consume tiempo y por tanto compensa el adelanto que se produce en las décimas.

Para justificar lo explicado en el párrafo anterior, se realiza un estudio del proceso de escritura de los segundos. A continuación, se muestra el fragmento de código correspondiente a dicho proceso.

```
…
        for(j=1;j<=60;j++)              //Segundos
          {
              …
              …

              if(j<=9)                                  //Si el número es de una unidad
```

```
                {
                    sprintf(b, "%d",j );                    // Convierte el entero j en texto
                    glcd_rect(58,25, 69, 35, ON, OFF);  // Borra el número anterior
                    glcd_text57(58,25,cer0,1,ON);       //Muestra el 0 en la decena del número
                    glcd_text57(64,25,b,1,ON);          //Muestra los segundos

                } else{

                    sprintf(b, "%d",j );                            // Convierte el entero j en texto
                    glcd_rect(58,25, 69, 35, ON, OFF);          // Borra el número anterior
                    glcd_text57(58,25,b,1,ON);                  //Muestra los segundos
                }
            } //Cierra el for de j
…
```

Como se puede observar en el código anterior, para la escritura de los segundos se ha utilizado un *if*, donde se diferencia entre la escritura de un número con un solo dígito (unidades) y los números con dos dígitos (decenas y unidades). En un primer momento se pensó en la escritura de los números menores de 10 de la siguiente manera:

```
sprintf(b, "%d",j );                    // Convierte el entero j en texto
glcd_rect(64,25, 69, 35, ON, OFF);      // Borra el número anterior
glcd_text57(64,25,b,1,ON);              //Muestra los segundos
```

Mediante este código, únicamente era borrado el número de las unidades, puesto que el de las decenas se mantenía con el dígito 0. Sin embargo, esta forma de escribir los números menores de 10 (de un dígito) era mucho más rápida que la de escribir el resto de números (de dos dígitos), puesto que estos últimos requerían de un número mayor de píxeles para ser representados en el módulo GLCD.

Esta es la razón por la cual se modificó el programa de la siguiente forma, donde la escritura de los números de un solo dígito y los de dos presentasen una duración similar.

```
sprintf(b, "%d",j );                    // Convierte el entero j en texto
glcd_rect(58,25, 69, 35, ON, OFF);      // Borra el número anterior
glcd_text57(58,25,cer0,1,ON);           //Muestra el 0 en la decena del número
glcd_text57(64,25,b,1,ON);              //Muestra los segundos
```

Nota: Cada vez que se procede a la escritura de un número de una unidad, se borra el número entero (0x) y se vuelve a escribir el dígito 0 y el dígito correspondiente.

A continuación se muestra la duración de la escritura de los segundos en el display GLCD.

Mostrar segundo	Duración (ms.)
01	42,555
02	43,81
03	43,8
04	43,8025
05	44,525
06	44,545
07	43,0275
08	44,5425
09	44,54
10	42,6285
11	40,146
12	41,401
13	41,391
14	41,3935
15	42,116
16	42,136
17	40,6185
18	42,1335
19	42,131
20	43,8635
21	41,381
22	42,636
23	42,626
24	42,6285
25	43,351
26	43,371
27	41,8535
28	43,3685
29	43,366

30	43,8535
…	…

Tabla B.4: Duración de la escritura de segundos

La duración de la escritura de los distintos segundos se ha obtenido con la herramienta “stopwatch”, pero también se puede calcular sumando el tiempo que dura cada función implicada en la escritura de dicho segundo. Los datos correspondientes a las funciones de este fragmento de código, se muestran en la Tabla B.5.

Función	**Duración (ms.)**
if(j<=9)	
entra	0,002
salta	0,0025
sprintf(b, "%d",j);	0,044
glcd_rect(58,25, 69, 35, ON, OFF);	32,3785
glcd_text57(58,25,cer0,1,ON);	6,268
glcd_text57(64,25,b,1,ON); cuando b es	
1	3,8615
2	5,1165
3	5,1065
4	5,109
5	5,8315
6	5,8515
7	4,334
8	5,849
9	5,8465
10	10,2035
11	7,721
12	8,976
13	8,966
14	8,9685
15	9,691
…	…

else	
salta	0,001

Tabla B.5: Duración de las funciones

Como se puede observar en la Tabla B.5, la duración de la escritura de los números de dos dígitos (10, 11, 12, etc.), corresponde a la suma del tiempo empleado en la escritura de los dígitos por separado. Por ejemplo, la escritura del número 11 es la equivalente a representar el número 1 dos veces (menos 2us. puesto que sólo llama a la función *glcd_text57* una vez).

Una vez obtenido la duración de las décimas y de la escritura de los segundos, se calcula el tiempo de duración de los segundos. Al igual que en los casos anteriores, estos datos calculados, pueden ser obtenidos directamente, sin necesidad de realizar ningún cálculo, mediante la herramienta “stopwatch”.

Segundo	Duración mostrar segundo (ms.)	Duración décimas en ms.	Duración de la función for(j=1;j<=60;j++) y vuelta (ms.)	Duración total de segundos en ms.
01	42,555	955,52	0.0025+0.001=0.0035	998,0785
02	43,81	955,52	0.0035	999,3335
03	43,8	955,52	0.0035	999,3235
04	43,8025	955,52	0.0035	999,326
05	44,525	955,52	0.0035	1000,0485
06	44,545	955,52	0.0035	1000,0685
07	43,0275	955,52	0.0035	998,551
08	44,5425	955,52	0.0035	1000,066
09	44,54	955,52	0.0035	1000,0635
10	42,6285	955,52	0.0035	998,152
11	40,146	955,52	0.0035	995,6695
12	41,401	955,52	0.0035	996,9245
13	41,391	955,52	0.0035	996,9145
14	41,3935	955,52	0.0035	996,917
15	42,116	955,52	0.0035	997,6395

16	42,136	955,52	0.0035	997,6595
17	40,6185	955,52	0.0035	996,142
18	42,1335	955,52	0.0035	997,657
19	42,131	955,52	0.0035	997,6545
20	43,8635	955,52	0.0035	999,387
21	41,381	955,52	0.0035	996,9045
22	42,636	955,52	0.0035	998,1595
23	42,626	955,52	0.0035	998,1495
24	42,6285	955,52	0.0035	998,152
25	43,351	955,52	0.0035	998,8745
26	43,371	955,52	0.0035	998,8945
27	41,8535	955,52	0.0035	997,377
28	43,3685	955,52	0.0035	998,892
29	43,366	955,52	0.0035	998,8895
30	43,8535	955,52	0.0035	999,377
…	…	…	…	…

Tabla B.6: Duración de los segundos

A continuación se muestra en la Tabla B.7, la duración de cada uno de los segundos que compone un minuto, así como su error.

Segundos	Duración (ms.)	Error (ms.)	% Error	Segundos acumulados en ms.	Segundos acumulados teóricos en ms.	Error acumulado en ms.
1	998,0785	1,9215	0,19215	998,0785	1000	1,9215
2	999,3335	0,6665	0,06665	1997,412	2000	2,588
3	999,3235	0,6765	0,06765	2996,7355	3000	3,2645
4	999,326	0,674	0,0674	3996,0615	4000	3,9385
5	1000,048	-0,048	-0,0048	4996,1095	5000	3,8905
6	1000,069	-0,069	-0,0069	5996,1785	6000	3,8215
7	998,551	1,449	0,1449	6994,7295	7000	5,2705
8	1000,066	-0,066	-0,0066	7994,7955	8000	5,2045
9	1000,064	-0,064	-0,0064	8994,8595	9000	5,1405

10	998,152	1,848	0,1848	9993,0115	10000	6,9885
11	995,6695	4,3305	0,43305	10988,681	11000	11,319
12	996,9245	3,0755	0,30755	11985,6055	12000	14,3945
13	996,9145	3,0855	0,30855	12982,52	13000	17,48
14	996,917	3,083	0,3083	13979,437	14000	20,563
15	997,6395	2,3605	0,23605	14977,0765	15000	22,9235
16	997,6595	2,3405	0,23405	15974,736	16000	25,264
17	996,142	3,858	0,3858	16970,878	17000	29,122
18	997,657	2,343	0,2343	17968,535	18000	31,465
19	997,6545	2,3455	0,23455	18966,1895	19000	33,8105
20	999,387	0,613	0,0613	19965,5765	20000	34,4235
21	996,9045	3,0955	0,30955	20962,481	21000	37,519
22	998,1595	1,8405	0,18405	21960,6405	22000	39,3595
23	998,1495	1,8505	0,18505	22958,79	23000	41,21
24	998,152	1,848	0,1848	23956,942	24000	43,058
25	998,8745	1,1255	0,11255	24955,8165	25000	44,1835
26	998,8945	1,1055	0,11055	25954,711	26000	45,289
27	997,377	2,623	0,2623	26952,088	27000	47,912
28	998,892	1,108	0,1108	27950,98	28000	49,02
29	998,8895	1,1105	0,11105	28949,8695	29000	50,1305
30	999,377	0,623	0,0623	29949,2465	30000	50,7535
31	996,8945	3,1055	0,31055	30946,141	31000	53,859
32	998,1495	1,8505	0,18505	31944,2905	32000	55,7095
33	998,1395	1,8605	0,18605	32942,43	33000	57,57
34	998,142	1,858	0,1858	33940,572	34000	59,428
35	998,8645	1,1355	0,11355	34939,4365	35000	60,5635
36	998,8845	1,1155	0,11155	35938,321	36000	61,679
37	997,367	2,633	0,2633	36935,688	37000	64,312
38	998,882	1,118	0,1118	37934,57	38000	65,43
39	998,8795	1,1205	0,11205	38933,4495	39000	66,5505
40	999,3795	0,6205	0,06205	39932,829	40000	67,171
41	996,897	3,103	0,3103	40929,726	41000	70,274
42	998,152	1,848	0,1848	41927,878	42000	72,122

43	998,142	1,858	0,1858	42926,02	43000	73,98
44	998,1445	1,8555	0,18555	43924,1645	44000	75,8355
45	998,867	1,133	0,1133	44923,0315	45000	76,9685
46	998,887	1,113	0,1113	45921,9185	46000	78,0815
47	997,3695	2,6305	0,26305	46919,288	47000	80,712
48	998,8845	1,1155	0,11155	47918,1725	48000	81,8275
49	998,882	1,118	0,1118	48917,0545	49000	82,9455
50	1000,09	-0,09	-0,009	49917,1445	50000	82,8555
51	997,6075	2,3925	0,23925	50914,752	51000	85,248
52	998,8625	1,1375	0,11375	51913,6145	52000	86,3855
53	998,8525	1,1475	0,11475	52912,467	53000	87,533
54	998,855	1,145	0,1145	53911,322	54000	88,678
55	999,5775	0,4225	0,04225	54910,8995	55000	89,1005
56	999,5975	0,4025	0,04025	55910,497	56000	89,503
57	998,08	1,92	0,192	56908,577	57000	91,423
58	999,595	0,405	0,0405	57908,172	58000	91,828
59	999,5925	0,4075	0,04075	58907,7645	59000	92,2355
60	1000,11	-0,11	-0,011	59907,8745	60000	92,1255
Escribir el minuto 01	87,24			59995,1145	60000	4,8855

Tabla B.7: Duración y error en un minuto

Nota: El valor de error positivo indica que la duración es menor de un segundo, mientras que el valor de error negativo, indica que dicha duración es mayor.

Según la Tabla B.7, en el segundo 60 se tiene un error de 92.1255 ms. de adelanto. Este error se ve compensado, inmediatamente, con el tiempo consumido en la escritura del minuto 01, obteniendo un error de solamente 4.8855 ms. en un minuto.

A continuación se detalla la parte del código correspondiente a la escritura de los minutos, con el fin de justificar la duración de la escritura del minuto 01.

```
…
            glcd_rect(58,25, 69, 35, ON, OFF);          //Borra los segundos
            glcd_text57(58,25,cer00,1,ON);
```

```
            if(k<=9)
            {
                sprintf(c, "%d",k );                    // Convierte el entero k en texto
                glcd_rect(40,25, 51, 35, ON, OFF);
                glcd_text57(40,25,cer0,1,ON);
                glcd_text57(46,25,c,1,ON);              //Muestra los minutos

            } else{
                sprintf(c, "%d",k );                    // Convierte el entero k en texto
                glcd_rect(40,25, 51, 35, ON, OFF);      // Borra el numero anterior
                glcd_text57(40,25,c,1,ON);              //Muestra los minutos

            }//Cierra el else

        }//Cierra el for de k
…
```

Como se ha visto en la Tabla B.7, la escritura del minuto 01 presenta una duración de 87.24 ms., dicho tiempo puede ser calculado mediante los datos que se muestran en la Tabla B.8.

Función	**Duración (ms.)**
glcd_rect(58,25, 69, 35, ON, OFF);	32,3785
glcd_text57(58,25,cer00,1,ON);	12,6
if (k<=9)	0,002
sprintf(c, "%d",k);	0,044
glcd_rect(40,25, 51, 35, ON, OFF);	32,1145
glcd_text57(40,25,cer0,1,ON);	6,268
glcd_text57(46,25,c,1,ON); con c=1	3,8255
Vuelta al bucle	0,0075
suma	87,24

Tabla B.8: Duración de la escritura del minuto 01.

Una vez analizada la precisión del cronómetro, se llega a la conclusión de que el adelanto producido tanto en décimas de segundo, como en segundos, minutos y horas, es compensado con el tiempo consumido en la escritura de los números (dígitos) en el display GLCD.

ANEXO C. PROGRAMAS DE LECTURA DE TENSIÓN

En el presente anexo, se describen las aplicaciones desarrolladas para la visualización del valor de tensión presente en el pin RA2.

- Fichero “Potenciometro.c”

Descripción:

A continuación se muestra la aplicación desarrollada para visualizar el valor de tensión presente en el pin RA2 tanto en valor numérico como en representación gráfica. En la Figura C.1 se puede apreciar la imagen mostrada en el display GLCD, cuando la tensión es de 2.07V. La línea horizontal se desplaza hacia arriba o hacia abajo, dependiendo del valor de tensión existente en RA2.

Figura C.1: Imagen del módulo GLCD

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                  Potenciometro.c
////
//// PFC Mª Nieves Robles Botella
////
//// Utilización del display GLCD
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#include <16F877A.h>
#include <miGLCD.c>


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)


void displayVoltage(int adc) {
        char voltage[9];
        float tn;
        tn=(float)adc * .01960784;
        sprintf(voltage, "%f", tn );
        // Convierte el entero adc en texto
        voltage[4] = '\0';                              // Muestra sólo 3 dígitos
        glcd_line(9,(int)(58-28/(5/tn)),120,(int)(58-28/(5/tn)), ON);          //Dibuja la recta
correspondiente
        glcd_rect(81, 19, 105, 26, ON, OFF);                        // Borra la tensión anterior
        glcd_text57(81, 19, voltage, 1, ON);                        // Muestra la tensión actual
        glcd_line(9,(int)(58-28/(5/tn)),120,(int)(58-28/(5/tn)), OFF);         // Borra la recta
}


void main() {
        char text[]= "TeNsIoN eN RA2";
        int adc=0;
        char  voltText[] = "V";


  setup_adc_ports(ALL_ANALOG);      //Establece todos los pines adc como analógicos
  setup_adc(ADC_CLOCK_INTERNAL);          //Habilita el módulo A/D
  set_adc_channel(2);                       //En la instruccion de read_adc leerá el canal 2 RA2

  glcd_init(ON);                    //Inicializa el GLCD

        glcd_rect(0,0,127,63,OFF,ON);           //Dibuja dos rectángulos
        delay_ms(500);
        glcd_rect(2,2,125,61,OFF,ON);

        delay_ms(1000);
        glcd_text57(10,10,text,1,ON);
        glcd_text57(20,10,text,1,ON);
        delay_ms(1000);
        //Muestra los ejes
        glcd_line(8,58,8,30, ON);
        glcd_line(8,58,120,58, ON);
        delay_ms(500);
        glcd_text57(106, 19, voltText, 1, ON);  // Muestra la palabra "Voltios"
```

```
        for(;;) {
                adc = read_adc();       // Lee el valor ADC
        displayVoltage(adc);            // Muestra el valor
  }
}
```

- Fichero “PotenciometroBar.c”

Descripción:

La siguiente aplicación desarrollada muestra el valor de tensión presente en el pin RA2. La barra horizontal así como el valor numérico que aparece en la Figura C.2, se ven modificados al cambiar el valor de tensión.

La longitud de la barra mostrada es proporcional al valor de tensión presente en RA2. Para la representación de dicha barra se ha utilizado la función *glcd_bar*.

Figura C.2: Imagen del módulo GLCD

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                        Potenciometro2.c
////
//// PFC Mª Nieves Robles Botella
////
//// Utilización del display GLCD
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
#include <16F877A.h>
#include <miGLCD.c>


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)


void displayVoltage(int adc) {
          char voltage[9];
          float tn;
          tn=(float)adc * .01960784;
          sprintf(voltage, "%f", tn );                    // Convierte el entero adc en texto
          voltage[4] = '\0';                              // Muestra sólo 3 dígitos
          glcd_bar(1,45, adc/2,45,5,ON);
          glcd_rect(81, 19, 105, 26, ON, OFF);    // Borra la tensión anterior
          glcd_text57(81, 19, voltage, 1, ON);    // Muestra la tensión actual
          glcd_bar(1,45,adc/2,45,5,OFF);

}

void main() {
          char text[]= "TeNsIoN eN RA2";
          int adc=0;
          char  voltText[] = "V";

          setup_adc_ports(ALL_ANALOG);//Establece todos los pines adc como analógicos
          setup_adc(ADC_CLOCK_INTERNAL);  //Habilita el módulo A/D
          set_adc_channel(2);                 //En la instruccion de read_adc leerá el canal 2 RA2

          glcd_init(ON);                    //Inicializa el GLCD

          glcd_rect(1,1,126,33,OFF,ON);             //Dibuja dos rectángulos
          delay_ms(500);
          glcd_rect(3,3,124,31,OFF,ON);
          delay_ms (500);

          delay_ms(1000);

          glcd_text57(10,10,text,1,ON);
          glcd_text57(20,10,text,1,ON);
          glcd_text57(106, 19, voltText, 1, ON);  // Muestra la palabra "Voltios"
  for(;;) {
          adc = read_adc();                   // Lee el valor ADC
          displayVoltage(adc);                 // Muestra el valor

    }

  }
```

- Fichero “PotenciometroBar2.c”

Descripción:

La aplicación desarrollada es idéntica a la mostrada anteriormente. La única diferencia existente, es que en esta aplicación se ha utilizado la función *glcd_rect* en lugar de *glcd_bar* para representar la barra horizontal, produciendo un efecto visual en el cual, la actualización de la barra se realiza verticalmente, mientras que con la función *glcd_bar* se realizaba horizontalmente.

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                  PotenciometroBar2.c
////
//// PFC Mª Nieves Robles Botella
////
//// Utilización del display GLCD
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#include <16F877A.h>
#include <miGLCD.c>


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)


void displayVoltage(int adc) {
        char voltage[9];
        float tn;
        tn=(float)adc * .01960784;
        sprintf(voltage, "%f", tn );
        // Convierte el entero adc en texto
        voltage[4] = '\0';                              // Muestra sólo 3 dígitos
        glcd_rect(0,45, adc/2,50,ON,ON);

        glcd_rect(81, 19, 105, 26, ON, OFF);            // Borra la tensión anterior
        glcd_text57(81, 19, voltage, 1, ON);            // Muestra la tensión actual
        glcd_rect(0,45,adc/2,50,ON,OFF);

}


void main() {
        char text[]= "TeNsIoN eN RA2";
        int adc=0, adcAnt=0;
        char  voltText[] = "V";
```

```
  setup_adc_ports(ALL_ANALOG);       //Establece todos los pines adc como analógicos
  setup_adc(ADC_CLOCK_INTERNAL);                    //Habilita el módulo A/D
  set_adc_channel(2);                    //En la instruccion de read_adc leerá el canal 2 RA2

  glcd_init(ON);                    //Inicializa el GLCD

          glcd_rect(1,1,126,33,OFF,ON);          //Dibuja dos rectángulos
          delay_ms(500);
          glcd_rect(3,3,124,31,OFF,ON);
          delay_ms (500);

          delay_ms(1000);


  for(;;) {
          glcd_text57(10,10,text,1,ON);                    //Para estar centrado
glcd_text57(20,10,text,1,ON);
          glcd_text57(106, 19, voltText, 1, ON);           // Muestra la palabra "Voltios"
    adc = read_adc();                                      // Lee el valor ADC
    displayVoltage(adc);                                   // Muestra el valor



     }

  }
```

- Fichero “PotenciometroBar3.c”

Descripción:

Mediante la siguiente aplicación, se representa el valor de tensión existente en el pin RA2 en formato *barra horizontal*. En este caso, la actualización de la barra se realiza únicamente cuando el microcontrolador ha detectado un cambio de tensión en el valor de RA2. Sin embargo, como se puede apreciar en la Figura C.3, la actualización del valor numérico es continuo, por tanto, la intensidad del número representado es menor.

Figura C.3: Imagen del GLCD

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                   PotenciometroBar3.c
////
//// PFC Mª Nieves Robles Botella
////
//// Utilización del display GLCD
////sobreescribiendo la barra
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#include <16F877A.h>
#include <miGLCD.c>


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)


void main() {
        char text[]= "TeNsIoN eN RA2";
        char voltage[9];
        float tn;
        int adc=0, adcAnt=0;
        char  voltText[] = "V";

        setup_adc_ports(ALL_ANALOG);//Establece todos los pines adc como analógicos
        setup_adc(ADC_CLOCK_INTERNAL);          //Habilita el módulo A/D
        set_adc_channel(2);             //En la instruccion de read_adc leerá el canal 2 RA2

        glcd_init(ON);                  //Inicializa el GLCD

        glcd_rect(1,1,126,33,OFF,ON);           //Dibuja dos rectángulos
        delay_ms(500);
        glcd_rect(3,3,124,31,OFF,ON);
        delay_ms (1500);
```

```
        glcd_text57(10,10,text,1,ON);    //Para estar centrado glcd_text57(20,10,text,1,ON);
        glcd_text57(106, 19, voltText, 1, ON);  // Muestra la letra "V"

        for(;;) {
           adc = read_adc();                    // Lee el valor ADC
                tn=(float)adc * .01960784;
                sprintf(voltage, "%f", tn );          // Convierte el entero adc en texto
                voltage[4] = '\0';                    // Muestra sólo 3 dígitos
                glcd_rect(81, 19, 105, 26, ON, OFF);     //Borra la tensión anterior
                glcd_text57(81, 19, voltage, 1, ON);    // Muestra la tensión actual

                if (adc!=adcAnt)
                {
                        glcd_rect(adc/2, 45, adcAnt/2, 50, ON, OFF);  // Borra la barra anterior
                        glcd_rect(0, 45, adc/2, 50, ON, ON);          // Muestra la barra actual
                        adcAnt = adc;
                }

  }
}
```

- Fichero “PotenciometroBar4.c”

Descripción:

Mediante la presente aplicación, se muestra el valor de la tensión de RA2 en formato numérico y formato *barra horizontal*. La aplicación desarrollada es una mejora del código anterior, ya que tanto el valor de tensión como su representación en formato barra no son actualizados hasta que no se produce cambio alguno en el pin RA2.

Figura C.4: Imagen del GLCD

Código en C:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                    PotenciometroBar4.c
////
//// PFC Mª Nieves Robles Botella
////
//// Utilización del display GLCD
//// sin sobreescribir en la barra
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#include <16F877A.h>
#include <miGLCD.c>


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=8000000)


void main() {
        char text[]= "TeNsIoN eN RA2";
        char voltage[9];
        float tn;
        int adc=0, adcAnt=0;
        char  voltText[] = "V";


  setup_adc_ports(ALL_ANALOG);       //Establece todos los pines adc como analógicos
  setup_adc(ADC_CLOCK_INTERNAL);                    //Habilita el módulo A/D
  set_adc_channel(2);             //En la instruccion de read_adc leerá el canal 2 RA2

  glcd_init(ON);                    //Inicializa el GLCD

        glcd_rect(1,1,126,33,OFF,ON);             //Dibuja dos rectángulos
        delay_ms(500);
        glcd_rect(3,3,124,31,OFF,ON);
        delay_ms (1500);
        glcd_text57(10,10,text,1,ON);    //Para estar centrado glcd_text57(20,10,text,1,ON);
        glcd_text57(106, 19, voltText, 1, ON);  // Muestra la letra "V"

  for(;;) {

   adc = read_adc();                        // Lee el valor ADC
   tn=(float)adc * .01960784;
        sprintf(voltage, "%f", tn );          // Convierte el entero adc en texto
        voltage[4] = '\0';                    // Muestra sólo 3 dígitos


        if (adc!=adcAnt)
        {
        glcd_rect(81, 19, 105, 26, ON, OFF);     //Borra la tensión anterior
        glcd_text57(81, 19, voltage, 1, ON);    // Muestra la tensión actual

                if(adc<adcAnt)
                {
                  glcd_rect(adc/2, 45, adcAnt/2, 50, ON, OFF);  // Borra el trozo de barra que
sobra
                }
```

```
                    else
                    {
                      glcd_rect(adc/2, 45, adcAnt/2, 50, ON, ON);    //Añade el trozo que falta
                    }

      adcAnt = adc;
                    }

  }
}
```

Como se puede apreciar, en el código anterior se realiza una comprobación que en el fichero “PotenciometroBar3.c” no existía, con el fin de no rescribir los píxeles que ya están dibujados o borrar los necesarios.

Se comprueba si el valor de tensión actual es mayor que el anterior y se realiza la diferencia para representar los píxeles que son necesarios para dibujar la barra. De la misma forma, cuando la tensión actual es menor se borran los píxeles que no han de estar representados. Consiguiendo así, una aplicación más eficiente.

ACRÓNIMOS

A/D	Analogic/Digital
ARES	Advanced Routing Modelling
Bit	Binary Digit
BOR	Brown-Out Reset
CAD	Computer Aided Design
CAN	Controller Area Network
CCS	Custom Computer Services
CGRAM	Character Generator RAM
CGROM	Character Generator ROM
CISC	Complex Instruction Set Computer.
CPU	Central Processing Unit
CS	Chip Select
DDRAM	Data Display RAM
EEPROM	Electrical Erasable Programmable Read Only Memory
EPROM	Erasable Programmable Read Only Memory
E/S	Entrada/Salida
GLCD	Graphic Liquid Crystal Display
IDE	Integrated Development Environment
ISIS	Intelligent Schematic Input System

LCD	Liquid Crystal Display
LED	Light Emitting Dioses
MCU	MicroController Unit
OP	Opcode, código de operación
OTP	One Time Programmable
PCB	Printed Circuit Board
PGC	Program Clock
PGD	Program Data
PIC	Peripheral Interface Controller
POR	Power On Reset
PSP	Parallel Slave Port
PWM	Pulse Width Modulator
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
SISC	Computadora de Juego de Instrucciones Específico
SFR	Special File Regisers
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous Asynchronous Receiver Transmitter También se conoce como SCI (Serial Communications Interface)
USB	Universal Serial Bus
VSM	Virtual System Modelling
WDT	Watchdog Timer

BIBLIOGRAFÍA

[1] http://www.atmel.com/, Atmel.

[2] http://www.freescale.com/, Freescale.

[3] http://www.hitachi.com/, Hitachi.

[4] http://www.intel.com/products/embedded/micro.htm?iid=embed_body+micro, Intel.

[5] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=64, Microchip Technology Inc.

[6] http://www.national.com/, National Semiconductor.

[7] http://www.standardics.nxp.com/, Philips Electronics.

[8] http://www.ti.com/, Texas Instruments.

[9] http://www.westerndesigncenter.com/wdc/, Western Design Center.

[10] http://www.zilog.com/, Zilog.

[11] http://www.mikroe.com/, Mikroelektronika.

[12] http://www.htsoft.com, Hi-tech.

[13] http://www.iar.com, IAR.

[14] http://www.ccsinfo.com/, CCS Inc.

[15] http://www.sourceboost.com/Products/Products.html, C2C Compiler.

[16] http://www.labcenter.co.uk/index.cfm, Labcenter Electronics.

[17] http://www.ieeproteus.com/, Proteus.

[18] http://www.microengineeringlabs.com/, Microengineering Labs.

[19] http://www.bytecraft.com/, Byte Craft.

[20] http://www.bizintekinnova.com/, Bizintek Innova.

[21] http://www.moway-robot.com/, Microbot Moway.

[22] "Microcontrolador PIC16F84, Desarrollo de proyectos", E. Palacios, F. Remiro, L.J. López. Ra-Ma, 2004

[23] “Microcontroladores PIC. La clave del diseño”, E. Martín, J.M. Angulo Usategui, I. Angulo Martínez. Thomson, 2003

[24] “Los PIC de la gama media. Arquitectura y técnicas de programación”, S. Salamanca Miño, D. Arroyo Muñoz. Escuela de Ingenierías Industriales de la Universidad de Extremadura