UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

INGENIERÍA INDUSTRIAL

"MYBOT: DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC"

PROYECTO FIN DE CARRERA

Junio – 2010

AUTOR: Diego Pedro González González

DIRECTORA: María Asunción Vicente Ripoll

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

MYBOT: DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC

Proyectante: Diego Pedro González González

Directora: María Asunción Vicente Ripoll

V°B° directora del proyecto:

Fdo.: María Asunción Vicente Ripoll

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

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

CAPÍTULO1 INTRODUCCIÓN Y OBJETIVOS

CAPITULO 1. INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

En la actualidad la oferta mundial de robots, tanto para su utilización en líneas de producción, como para el estudio o acercamiento inicial a la robótica es muy elevada. Hoy por hoy existen muchas empresas dedicadas a la construcción de robots . Así mismo existen empresas que integran la robótica en kits de juegos [LEGONTX], preparados para armar, agregando alguna forma de procesamiento, movilidad mediante motores y reconocimiento por medio de sensores.

La mayoría de las empresas involucradas en la construcción y venta de robots son europeas o norteamericanas, y los precios de los robots que producen están de acuerdo con su procedencia. Se hace difícil, para medios académicos, adquirir uno de estos robots así como repuestos, soporte y partes nuevas.

El presente proyecto, indaga la posibilidad de construir un micro robot con una estructura abierta y de bajo coste, enfocado en la utilización de componentes electrónicos comunes, sin perder de vista el enfoque académico que impone el ser un proyecto fin de carrera.

Este proyecto fin de carrera se centra en la creación de un micro robot con una estructura abierta que permita la modificación o ampliación tanto del software como del hardware. La Figura 1.1 muestra el prototipo desarrollado, basado en el sistema descrito en el libro Microcontrolador PIC 16f874.

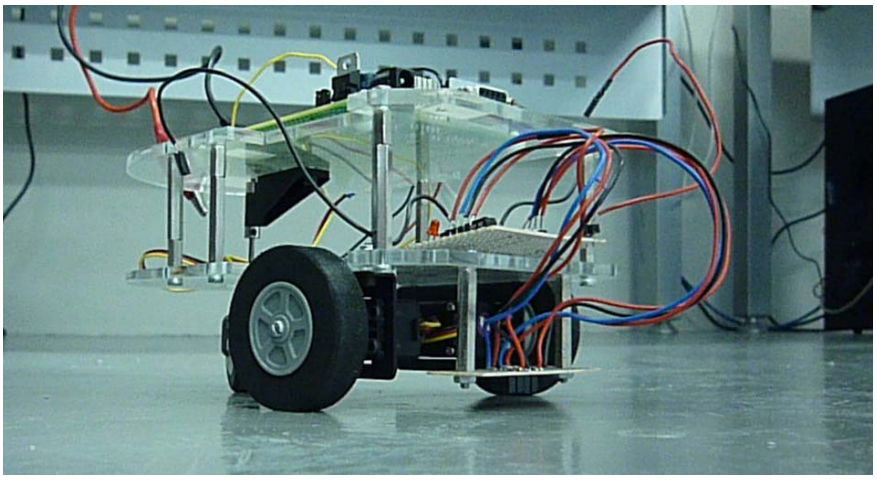


Figura 1. 1 : micro robot MyBot

1.2. Objetivos

El principal objetivo del proyecto es el diseño y la construcción de un micro robot de arquitectura abierta, tanto en software como en hardware, mediante el uso de microcontroladores PIC.

Aparte de este objetivo principal se destacan otros tales como:

- Obtención de un micro robot semejante al MOWAY [1] y al micro robot descrito en el libro microcontroladores PIC 16f874 [2]
- Adquisición de conocimientos en materia de tecnologías de comunicación entre dispositivos, control de servomotores, diseño y montaje de circuitos, uso de sensores y otros componentes.
- Simplificación de la electrónica y llevar los problemas al campo del software siempre que esto sea posible.
- Diseño de un modelo de micro robot flexible y extensible.

Las características del micro robot deberán ser:

- El prototipo debe ser construido con la mayor cantidad de elementos o componentes fácilmente adquiribles.
- El micro robot debe ser autónomo mediante el uso de sensores previamente instalados. También debe tener la posibilidad de la tele operación ya sea mediante el PC o mediante un mando de control inalámbrico.
- Se debe diseñar una librería que contenga las funciones necesarias para el movimiento del micro robot.

Por lo tanto este documento propone un modelo de micro robot multipropósito, orientado a múltiples aplicaciones .

1.3. Trabajos Relacionados

El presente Proyecto Fin de Carrera deriva del Proyecto Final de Carrera **“DESARROLLO DE APLICACIONES Y PUESTA EN MARCHA DEL SISTEMA EASYPIC PARA MICROCONTROLADORES PIC”[3]** centrado básicamente en el desarrollo de aplicaciones prácticas para la docencia mediante la programación de microcontroladores PIC de gama media. Luego el proyecto actual nace como trabajo futuro del anterior proyecto.

El proyecto **"MANEJO DE UNA PANTALLA TÁCTIL CON EL PIC32 PARA EL CONTROL DE DISPOSITIVOS EXTERNOS"[4],** también se ha utilizado como fuente bibliográfica para analizar y estudiar las nuevas posibilidades que nos ofrece el PIC32 respecto al PIC16 en el campo de la micro robótica

1.4. Guía de Capítulos

En el capítulo 2 se realiza una introducción a la arquitectura de los microcontroladores en general y se describe el microcontrolador PIC16F877A y las características principales que se han utilizado del mismo.

El capítulo 3, con motivo de las limitaciones del PIC 16F877A ,que se verán en los capítulos siguientes , se describe con detalle la familia de los microcontroladores PIC de 32 bits.

En el capítulo 4 se narran los distintos entornos de desarrollos utilizados. Es un capitulo que describe en detalle el hardware utilizado en el proyecto.

En el capítulo 5 se presentan los distintos programas que se han utilizado a lo largo del proyecto, tanto para la grabación de los programas en los microcontroladores, como para la depuración y simulación del código fuente.

El capítulo 6 se centra la arquitectura de un micro robot. El tipo de configuración adoptada. Y en general se detalla el Nivel físico de la Torre bot, que posee el micro robot. Explicando el tipo de ruedas, motores, estructura utilizada y su cinemática.

El capítulo 7 se evalúa en el Nivel de reacción. En este capítulo se detalla los sensores utilizados y placas desarrolladas.

El capítulo 8 trata de explicar cómo se controlan los servomotores Hitec para controlar los movimientos del micro robot con un microcontrolador de 8 bits, se narra así la evolución de las formas de programación y algunas justificaciones de decisiones que tuvieron que ver con lenguajes de programación.

En el capítulo 9, a partir de la librería desarrollada en el capítulo 8, se centra en aplicaciones básicas utilizadas por el micro robot con los sensores descritos en el capítulo 7.

El capítulo 10 propone una serie de trabajos futuros para el micro robot y expone una serie de conclusiones.

CAPÍTULO 2
INTRODUCCIÓN A LOS MICROTROLADORES DE 16 BITS

CAPÍTULO 2. EL MICROCONTROLADOR PIC16: HARDWARE Y SOFTWARE

2.1. INTRODUCCIÓN

Hace unos años, los sistemas de control se implementaban usando exclusivamente la 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. DIFERENCIAS 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 de los buses necesarios para comunicarse con los distintos módulos. 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.

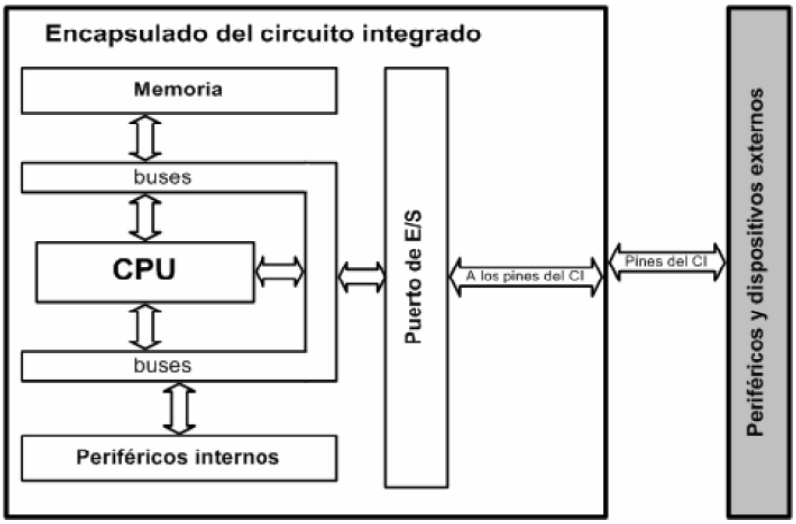


Figura 2. 1: Estructura básica de un microcontrolador.

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.

2.1.2. APLICACIONES DE LOS MICROCONTROLADORES

En la actualidad podemos encontrar 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.

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.

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. 2: Robot humanoide Rovonova

Otro ejemplo de aplicación de los microcontroladores, es la de la utilización para monitorizar y grabar 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.

2.1.3. MODELOS DE MICROCONTROLADORES DE LA MARCA MICROCHIP

Diversos fabricantes ofrecen amplias gamas de microcontroladores para todas las necesidades. Nosotros vamos a utilizar microcontroladores de la marca microchip [5], al ser la marca con un mayor número de modelos de estos y por su mayor utilización tanto profesionalmente como por aficionados.

Dentro de la marca de microchip, nos encontramos con varios modelos de microcontroladores. Estos se clasifican de acuerdo a la longitud de sus instrucciones dando lugar a 3 grandes grupos de PIC.

Clasificación	Longitud de instrucciones	Modelos de PIC
Gama Baja	8 bits	Pic10, Pic12, Pic16, Pic18
Gama Media	16 bits	Pic24F, Pic24H, dsPIC30, dsPIC33
Gama Alta	32 bits	Pic32

Tabla 2. 1: Modelos de microcontroladores de la marca Microchip.

Además, conforme aumentamos la longitud de las instrucciones va a aumentar la funcionalidad, las prestaciones ofrecidas pero también la complejidad de las instrucciones y de su uso.

Figura 2. 3: PIC16F877A, modelo de microcontrolador muy popular en la docencia de PICs

2.2. LOS MICROCONTROLADORES PIC

En los últimos años han tenido un gran auge los microcontroladores PIC fabricados por Microchip Technology Inc. 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.

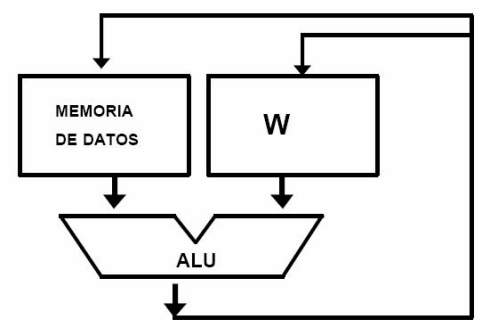


Figura 2. 4: Arquitectura ortogonal utilizada por un PIC

- 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, I2C, 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 2.2 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 PICs

2.3. MICROCONTROLADOR PIC16F877A

2.3.1. 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 PIC16F877A[6] vino influenciada por el uso que se hizo en el proyecto final de carrera **"DESARROLLO DE APLICACIONES Y PUESTA EN MARCHA DEL SISTEMA EASYPIC PARA MICROCONTROLADORES PIC"[3]** .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 micro robot y en base a ello seleccionar el más adecuado, se tomó el PIC16F877A y se verificó si dicho microcontrolador cumplía los requisitos necesarios.

2.3.2. CARACTERÍSTICAS DEL MICROCONTROLADOR PIC16F877A

El PIC16F877A[6] 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.

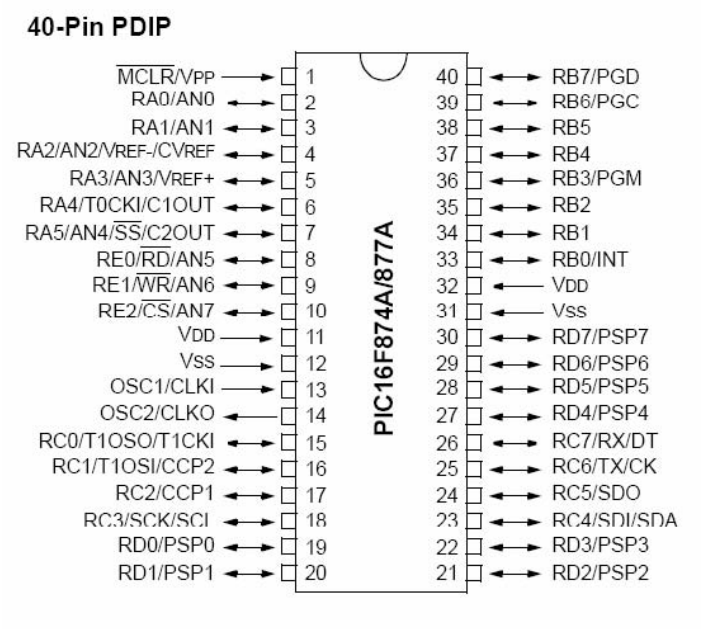


Figura 2. 5 PIC16F877A

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.

(Se adjunta en el anexo A un manual para su configuración.)

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.

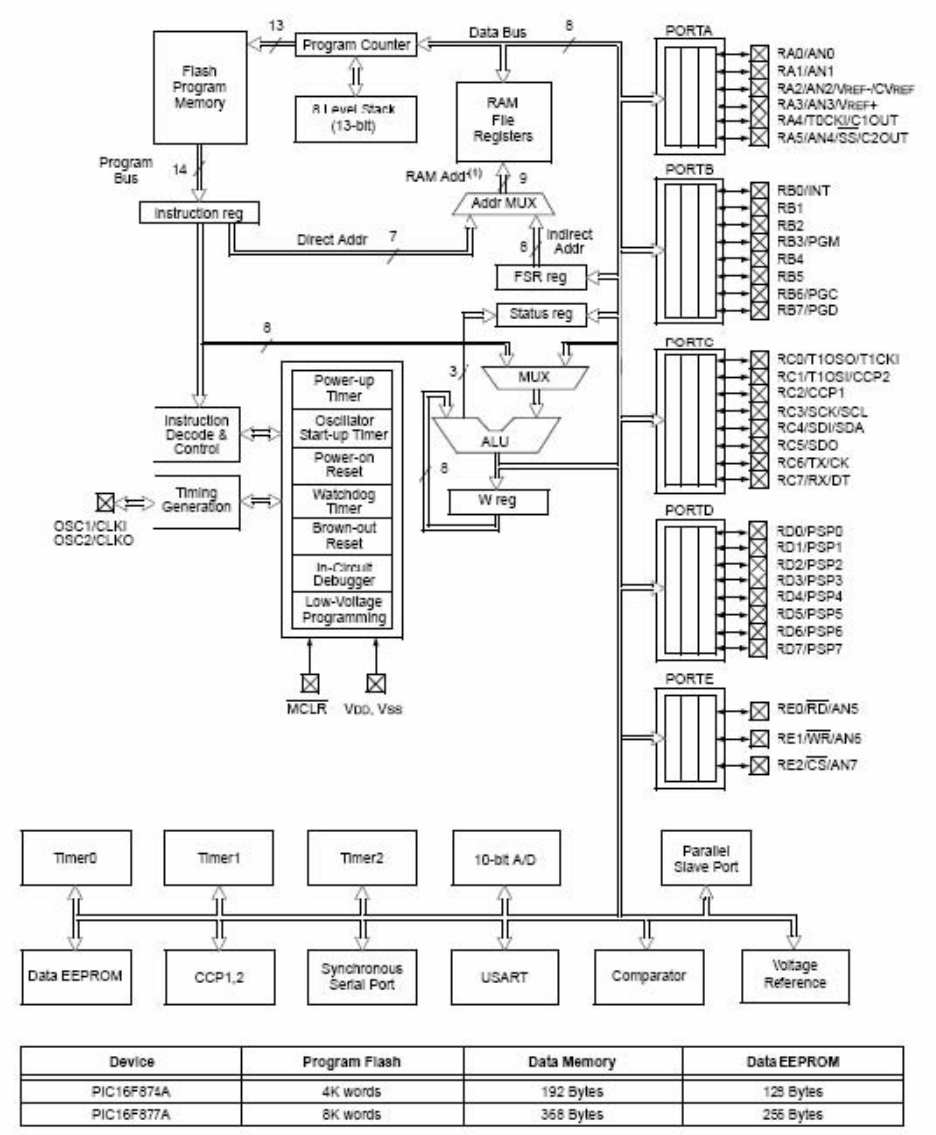
- Timer0: Temporizador-contador de 8 bits con prescaler de 8 bits.
- Timer1: Temporizador-contador de 16 bits con prescaler, puede incrementarse en modo sleep de forma externa por un cristal/clock.
- Timer2: Temporizador-contador de 8 bits con registro de periodo, prescaler y postescaler.
- Dos módulos 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. 6: 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].

2.4. 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.

2.4.1. 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.

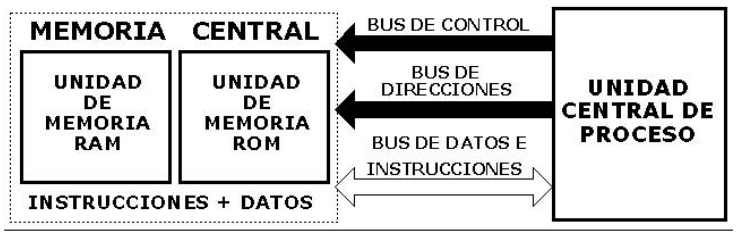


Figura 2. 7 Arquitectura Von Neumann

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.

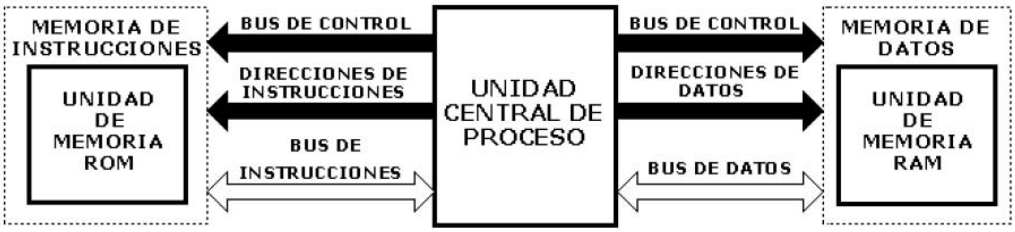


Figura 2. 8 : Arquitectura Harvard

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.

2.4.2. 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).

2.4.3. 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.

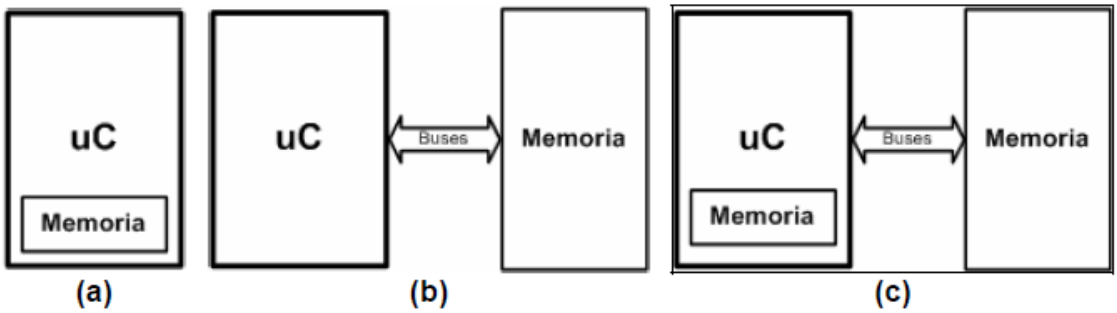


Figura 2.9: Algunas configuraciones de la memoria de programa en microcontroladores

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.

2.4.4. 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.

2.4.5. 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.

2.4.6. RECURSOS ESPECIALES

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

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

CAPÍTULO 3
INTRODUCCIÓN A LOS MICROCONTROLADORES DE 32 BITS

CAPÍTULO 3. EL MICROCONTROLADOR PIC32: HARDWARE Y SOFTWARE

3.1. MICROCONTORLADOR PIC32

3.1.1. ELECCIÓN DEL MICROCONTROLADOR

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

Sin embargo, los modernos microcontroladores de 32 bits (puestos en el mercado a finales del año 2007) 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 de él en el proyecto **"MANEJO DE UNA PANTALLA TÁCTIL CON EL PIC32 PARA EL CONTROL DE DISPOSITIVOS EXTERNOS"[4],** con motivo de analizar y estudiar las nuevas posibilidades que nos ofrecía el PIC32 respecto al PIC16 en el campo de la micro robótica . Dentro de los microcontroladores de gama alta vamos a utilizar dos modelos de 32 bits, el PIC32MX360F512L y el PIC32MX460F512L.

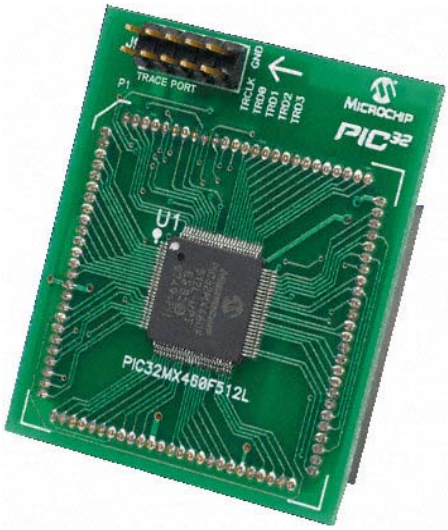


Figura 2.4: PIC32MX460512L.

3.1.2. CARACTERÍSTICAS DEL MICROCONTROLADOR PIC32MXXX

El PIC32X360F512L, introduce una nueva línea de dispositivos de microchip correspondientes a la familia de microcontroladores de 32 bit RISC (Reduced Instruction Set Computer). Además esta familia ofrece la opción de una nueva migración para estas aplicaciones de altas prestaciones las cuales pueden quedarse pequeñas para las plataformas de 16-bit.

A continuación se muestran las características generales del PIC32:

- Frecuencia de operación: Dc-80MHZ.
- Memoria de programa: 512Kbytes
- Memoria de Datos: 32Kbytes
- Recursos de interrupción/vectores: 95/63
- Puertos de E/S: Puertos A, B, C, D, F, G
    - Total pins E/S: 85
- Canales DMA(Direct memory Access): 4
- Timers:
    - Número total (16bit): 5
    - 32-bit (pareja de 16-bit): 2
    - Timer de núcleo de 32 bit: 1

- Canales de captura de entrada: 5
- Canales de salida Comparadores/PWN
- Notificación de cambio de entradas por interrupción: 22
- Comunicaciones en serie:
    - UART: 2
    - SPI(3cables/4cables): 2
    - I^2C: 2
- Comunicaciones paralelas(PMP/PSP): 8bit/16bit
- JTAG boundary SCAN
- JTAG debug and program
- ICSP 2-wire debug and program:
- Instrucción trace
- Hardware break points: 6 instructions, 2 Data
- Modulo de 10-bit analógico-digital (Canales de entrada): 16
- Comparadores analógicos: 2
- Interno LDO
- Resets (y retrasos): POR, BOR, MCLR, WDT, SWT(software reset), CM(configuration Bit Mismatch)
- 100 pin TQFP

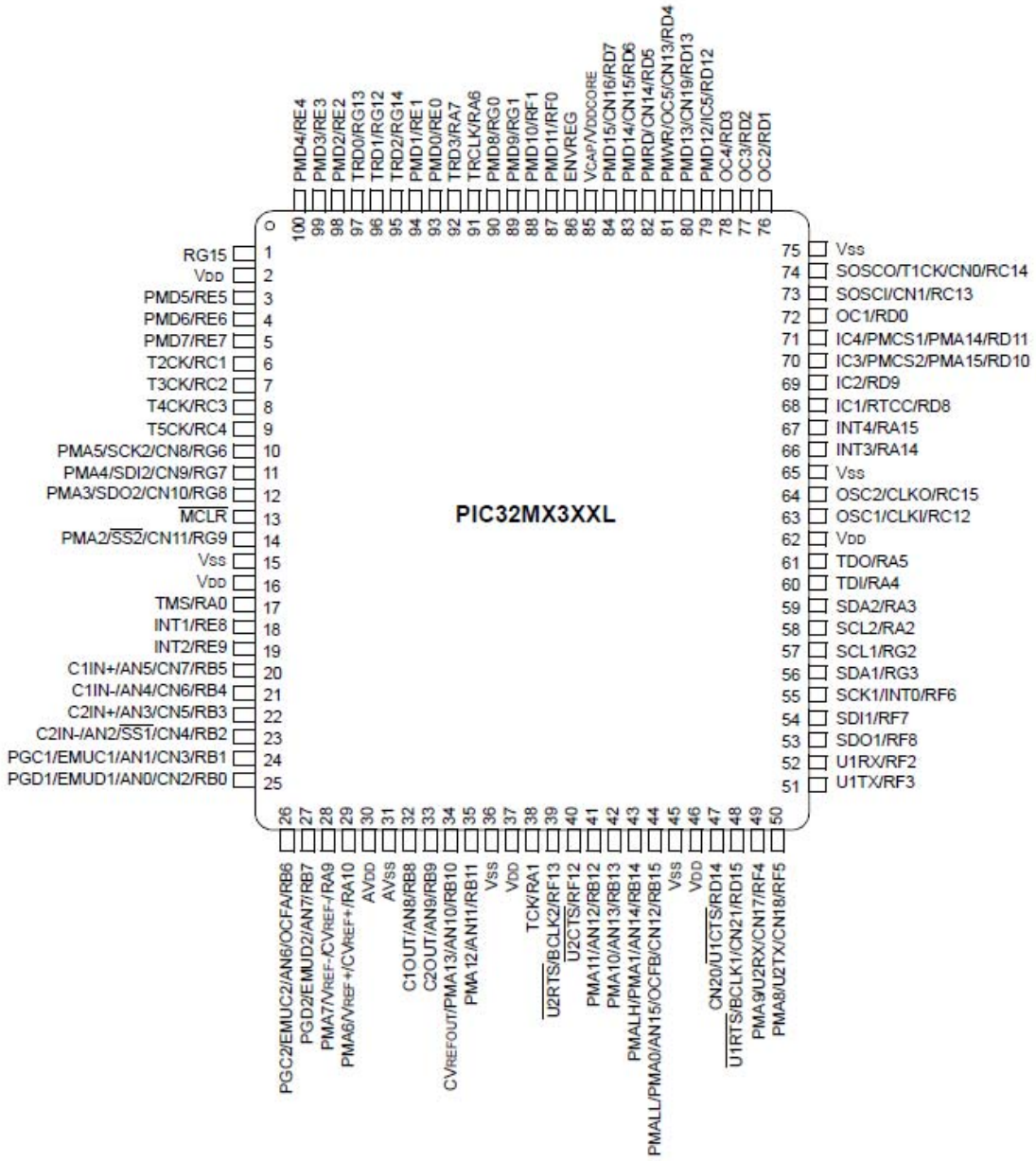


Figura 2.5: 100 pines del PIC32MX3XXL.

Por otra parte, la arquitectura del PIC32 ha sido descompuesta en los siguientes bloques funcionales:

- Núcleo MCU
- Memoria del sistema
- Periféricos
- Integración del sistema

3.1.3. NÚCLEO MCU

El corazón del PIC32 es el núcleo M4K CPU basado en la tecnología MIPS32 [7]. La CPU realiza las operaciones bajo el programa de control. Las instrucciones son leídas y cargadas por la CPU, decodificadas y ejecutadas síncronamente. Estas instrucciones se pueden almacenar en la memoria flash de programa o bien en la memoria de datos RAM.

Esta CPU del PIC32 se basa, por tanto, en una arquitectura de carga-almacenamiento la cual realiza la mayoría de operaciones en base a una serie de registros internos (contiene 32 registros de propósito general de 32 bits). Estas instrucciones específicas de carga y almacenamiento se utilizan para mover datos entre estos registros internos así como fuera del PIC.

Las características principales del núcleo MCU son las siguientes:

- Núcleo RISC MIPS32 M4K de 32 bits.
- ALU de un solo ciclo.
- Unidad de ejecución de carga-almacenamiento.
- 5 estados pipeline.
- Buses de direcciones y datos de 32 bits.
- Archivos de registro de propósito general de 32 bits.
- FMT-Fixed Mapping Translation.
- FMDU-Fast-Multiply-Divide Unit.
- MIPS32 Compatible instruction set.
- MIPS16e Code Compression Instruction Set Architecture Support.
- Instrucciones de 16 y 32 bits, optimizadas para lenguajes de alto nivel como C.
- Puerto debug EJTAG
- Rendimiento hasta 1.5DMIPS/MHz
- Protección del código interno para ayudar a proteger la propiedad intelectual.

A continuación vamos a ir describiendo algunas de estas características así como los conceptos más importantes del Núcleo MCU.

3.1.3.1. Estados pipeline

Los estados del pipeline son 5:

- Estado de instrucción (I)
- Estado de ejecución (E)
- Estado de memoria (M)
- Estado de alinear (A)
- Estado de reescribir (W)

Estado I: Instrucción Fetch

Durante el estado I una instrucción es cargada y leída desde SRAM y las instrucciones MIPS16e son convertidas a instrucciones en MIPS32.

Estado E: Ejecución

Durante el estado E:

- Los operandos son leídos y cargados desde el archivo de registro.
- Operandos de los estados M y A son desviados a este estado.
- La unidad aritmético lógica (ALU) empieza las operaciones aritméticas o lógicas para las instrucciones registro a registro.
- La ALU calcula la dirección virtual de los datos para las instrucciones de cargar y guardar y el MMU (Memory Management Unit) realiza el traslado de la dirección virtual a la física.
- La ALU determina si para una condición de bifurcación esta es verdadera y calcula la dirección objetivo de la ramificación.
- Las instrucciones lógicas seleccionan una dirección de instrucción y el MMU realiza la traslación de la dirección virtual a la física.
- Todas las operaciones de división y multiplicación empiezan en este estado.

Estado M: Memory Fetch

Durante este estado:

- Terminan las operaciones aritméticas o lógicas de la ALU.
- Se realiza el acceso a los datos SRAM para las instrucciones de carga y almacenamiento.
- Los cálculos de multiplicaciones y divisiones continúan en la MDU (Multiply/Divide Unit). Si el cálculo termina antes que el IU (Integer Unit)

mueva la instrucción pasada al estado M, entonces la MDU mantiene el resultado en un registro temporal hasta que el IU mueve la instrucción al Estado A (y consecuentemente sabrá que no será eliminado)

Estado A: Alinear

Durante el estado A:

- Una operación MUL hace que el resultado esté disponible para la reescritura. El registro actual reescrito es ejecutado en el estado W.
- Desde este estado, la carga de datos o el resultado de la MDU están disponibles para su bypassing (comentado a continuación) en el estado E.

Estado W: Reescribir

Durante el registro W: Para registros a registros o cargar instrucciones, el resultado se reescribe en el archivo de registro.

El núcleo M4k implementa un mecanismo de desviación (bypassing) que permite que el resultado de una operación sea enviado directamente a la instrucción que lo necesita sin tener que escribir el resultado en el registro y entonces volver a leerlo.

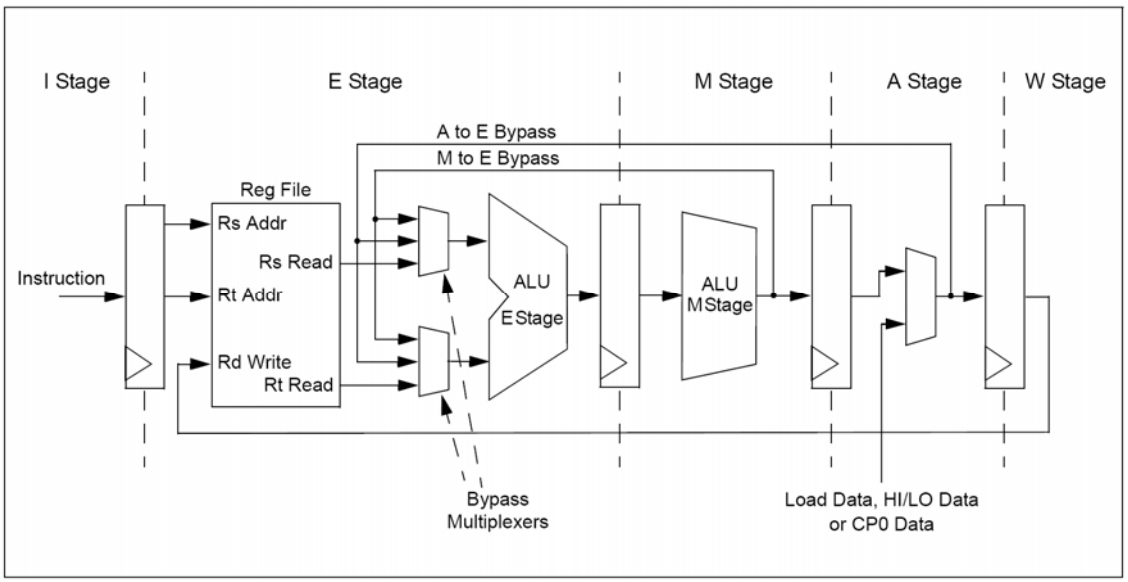


Figura 2.6: Esquema de los estados pipeline de la CPU del PIC32MX.

3.1.3.2. Unidad de Ejecución

La unidad de ejecución del PIC32 es la responsable de llevar a cabo el procesamiento de la mayoría de instrucciones del MIPS. Esta proporciona una ejecución para la mayoría de instrucciones en un solo ciclo, mediante la ejecución en pipeline (pipelining), donde las complejas operaciones son divididas en pequeñas partes llamados estados (explicados anteriormente).

3.1.3.3. MDU: Unidad de Multiplicación y División

La unidad de multiplicación y división realiza como su nombre indica, las operaciones de división y multiplicación. La MDU consiste en un multiplicador de 32x16, registros de resultado de acumulación (HI-LO) y todos los multiplexores y la lógica de control requerida para realizar estas funciones. Esta unidad tiene unas altas prestaciones ya que soporta la ejecución de una operación de multiplicación de 16x16 o 32x16 cada ciclo de reloj, y una operación de multiplicación de 32x32 se realiza cada dos ciclos de reloj. Por otra parte, las operaciones de división se implementan con un solo bit cada ciclo de reloj mediante un algoritmo iterativo y requiere 35 ciclos de reloj en el peor de los casos para que se complete la operación. Tan pronto como el algoritmo detecta el signo del dividendo, si el tamaño actual es 24, 16 o 8 bits, el divisor salta 7, 15 o 23 de las 32 iteraciones.

Además, el M4K implementa una instrucción adicional de multiplicación, MUL, la cual consiste en que los 32 bits más bajos del resultado de la multiplicación se guardan en el archivo de registro en lugar de en el par de registros HI/LO. Para ello, se han diseñado dos instrucciones adicionales que se usan para ejecutar las operaciones de multiplicación y suma así como multiplicación y resta, esta son: multiplicación-suma (MADD/MADDU) y multiplicación-resta (MSUB/MSUBU). La instrucción MADD multiplica dos números y entonces suma el producto al contenido actual de los registros HI y LO. Similarmente, la instrucción MSUB multiplica los dos operandos y entonces resta el producto de los registros HI y LO. Estas operaciones se suelen usar en algoritmos de Digital Signal Processor (DSP).

3.1.3.4. Shadow Register Sets

El procesador del PIC32 implementa una copia de los registros de propósito general para usarlos como interrupciones de alta prioridad. Este banco extra de registro es conocido como shadow register set (controlados por los registros ubicados en la CP0) [8]. Cuando ocurre una interrupción de alta prioridad, el procesador automáticamente conmuta a los shadow register set sin la necesidad de una intervención del software.

Este banco de registros especiales permite reducir eficazmente, tanto la sobrecarga en el manejo de interrupciones como el tiempo de latencia.

3.1.3.5. Register Bypassing

Un Interlock o bloqueo, ocurre cuando una instrucción en el estado pipeline no puede avanzar debido a una dependencia de datos o una condición externa similar, por ejemplo, cuando una instrucción depende del resultado de una instrucción previa. El tipo de bloqueo hardware en el procesador MIPs es Slips. Estos permiten que una parte de la pipeline avance mientras otra parte de la pipeline permanece estática, se muestra un ejemplo en la siguiente figura:

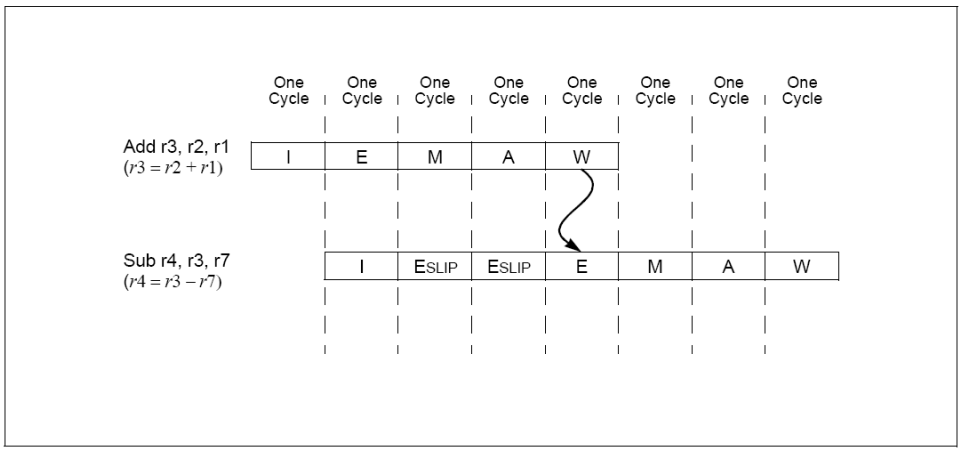


Figura 2.7: Esquema de los estados pipeline de la CPU del PIC32MX ante bloqueo hardware Slip.

Sin embargo, el procesador del PIC32 implementa un mecanismo llamado register bypassing que ayuda a reducir estos slips en la pipeline durante la ejecución. Cuando una instrucción esta en el estado E de la pipeline, el operando debe estar disponible para la siguiente instrucción.

El registro bypassing permite un atajo para conseguir directamente el operando desde la pipeline. De tal forma que una instrucción en el estado E puede recuperar un operando de otra instrucción que se está ejecutando o en el estado M o en el estado A de la pipeline. Mediante la figura que se muestra a continuación podemos ver las interdependencias descritas:

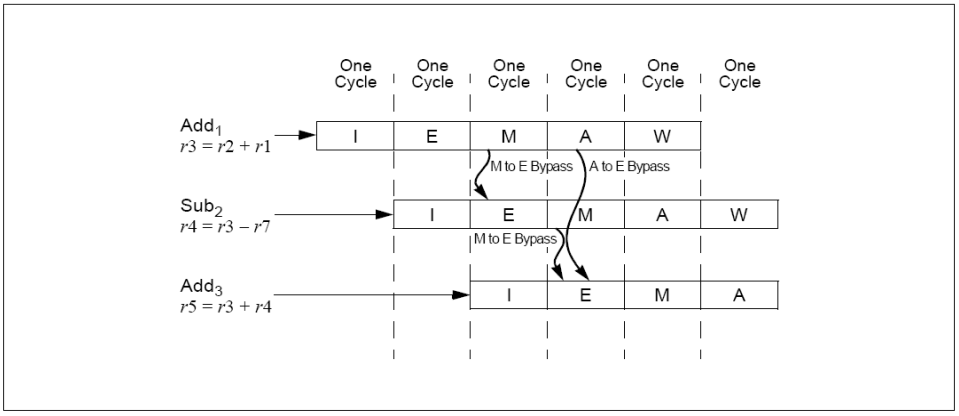


Figura 2.8: Esquema de los estados pipeline de la CPU del PIC32MX usando el mecanismo bypassing.

Evidentemente el uso del bypassing aumenta el rendimiento en el rango de una instrucción por ciclo reloj para las operaciones de la ALU.

3.1.3.6. BITS de estado de la ALU

A diferencia de la mayoría de los otros microcontroladores PIC, el PIC32 no usa un registro de Status con flags. Las banderas o flags se usan en la mayoría de los procesadores a ayudar a tomar una decisión para realizar operaciones durante la ejecución de un programa. Estas banderas se basan en, resultados de comparación de operaciones o bien en operaciones aritméticas. De tal forma que el programa ejecuta instrucciones basadas en estos valores de las banderas.

Sin embargo el PIC32 usa instrucciones que realizan una comparación y almacenan una bandera o valor en un registro de propósito general, ejecutándose entonces una rama condicionada, usando este registro de propósito general que actúa como un operando.

3.1.3.7. Mecanismo de interrupciones y excepciones

La familia de procesadores del PIC32 implementa un mecanismo eficiente y flexible para el manejo de las interrupciones y excepciones. Ambos se comportan de manera similar ya que la instrucción actual cambia temporalmente para ejecutarse un procedimiento especial. La diferencia entre las dos es que las interrupciones son usualmente un comportamiento normal y las excepciones son el resultado de un error en las condiciones, como por ejemplo un error en el bus al enviar o recibir datos.

Cuando ocurre una interrupción o una excepción, el procesador realiza los siguientes pasos:

- El PC (contador de programa) de la siguiente instrucción a ejecutar se guarda en el registro del coprocesador.
- La causa del registro es actualizada para conocer la razón de la excepción o de la interrupción.
- Status EXL o ERL es puesto a 1 como causa del modo de ejecución Kernel.
- Desde los valores de EBASE y SPACING se calcula el PC.
- El procesador comienza la ejecución del programa desde un nuevo PC.

3.1.4. JUEGO DE INSTRUCCIONES

La familia de los microprocesadores PIC32 están diseñados para usarse con un leguaje de programación de alto nivel como es el leguaje de programación C. El PIC32 soporta varios tipos de datos y usa un modo de direccionamiento simple pero necesario para el lenguaje de alto nivel. De manera que dispone de 32 registros de propósito general y 2 registros especiales para realizar las operaciones de multiplicación y división (Hi-Lo).

Existen tres tipos diferentes de formatos para las instrucciones en lenguaje máquina presentes en el procesador.

- Instrucciones Inmediatas o tipo I.
- Instrucciones de Salto o tipo J.
- Instrucciones de Registros o tipo R.

La mayoría de estas instrucciones son ejecutadas en registros. Las instrucciones de Registros tienen tres operandos: 2 fuentes y un destino. Las instrucciones inmediatas tienen una fuente y un destino, mientras que las instrucciones de salto tienen una instrucción relativa de 26 bit, que se usa para calcular el destino del salto.

Field	Description
opcode	6-bit primary operation code
rd	5-bit specifier for the destination register
rs	5-bit specifier for the source register
rt	5-bit specifier for the target (source/destination) register or used to specify functions within the primary opcode REGIMM
immediate	16-bit signed immediate used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement
instr_index	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address
sa	5-bit shift amount
function	6-bit function field used to specify functions within the primary opcode SPECIAL

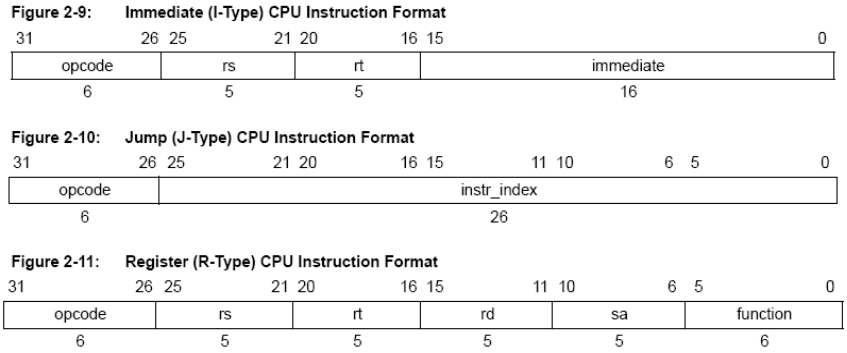


Figura 2.9: Formato de los 3 tipos de instrucciones.

A continuación mostramos una tabla resumen con las instrucciones que están implementadas en los núcleos de las siguientes familias de microcontroladores de 32 bits, PIC32MX3XX/4XX:

Instruction	Description	Function
ADD	Integer Add	Rd = Rs + Rt
ADDI	Integer Add Immediate	Rt = Rs + Immed
ADDIU	Unsigned Integer Add Immediate	Rt = Rs $+_U$ Immed
ADDIUPC	Unsigned Integer Add Immediate to PC (MIPS16e™ only)	Rt = PC $+_u$ Immed
ADDU	Unsigned Integer Add	Rd = Rs $+_U$ Rt
AND	Logical AND	Rd = Rs & Rt
ANDI	Logical AND Immediate	Rt = Rs & (0_{16} || Immed)
B	Unconditional Branch (Assembler idiom for: BEQ r0, r0, offset)	PC += (int)offset
BAL	Branch and Link (Assembler idiom for: BGEZAL r0, offset)	GPR[31> = PC + 8 PC += (int)offset
BEQ	Branch On Equal	if Rs == Rt PC += (int)offset
BEQL	Branch On Equal Likely	if Rs == Rt PC += (int)offset else Ignore Next Instruction
BGEZ	Branch on Greater Than or Equal To Zero	if !Rs[31> PC += (int)offset
BGEZAL	Branch on Greater Than or Equal To Zero And Link	GPR[31> = PC + 8 if !Rs[31> PC += (int)offset
BGEZALL	Branch on Greater Than or Equal To Zero And Link Likely	GPR[31> = PC + 8 if !Rs[31> PC += (int)offset else Ignore Next Instruction
BGEZL	Branch on Greater Than or Equal To Zero Likely	if !Rs[31> PC += (int)offset else Ignore Next Instruction
BGTZ	Branch on Greater Than Zero	if !Rs[31> && Rs != 0 PC += (int)offset
BGTZL	Branch on Greater Than Zero Likely	if !Rs[31> && Rs != 0 PC += (int)offset else Ignore Next Instruction

Instruction	Description	Function
BLEZ	Branch on Less Than or Equal to Zero	if Rs[31> || Rs == 0 PC += (int)offset

BLEZL	Branch on Less Than or Equal to Zero Likely	if Rs[31> || Rs == 0
		  PC += (int)offset
		else
		  Ignore Next Instruction

BLTZ	Branch on Less Than Zero	if Rs[31> PC += (int)offset
BLTZAL	Branch on Less Than Zero And Link	GPR[31> = PC + 8 if Rs[31> PC += (int)offset

BLTZALL	Branch on Less Than Zero And Link Likely	GPR[31> = PC + 8
		if Rs[31>
		  PC += (int)offset
		else
		  Ignore Next Instruction

BLTZL	Branch on Less Than Zero Likely	if Rs[31>
		  PC += (int)offset
		else
		  Ignore Next Instruction

BNE	Branch on Not Equal	if Rs != Rt PC += (int)offset

BNEL	Branch on Not Equal Likely	if Rs != Rt
		  PC += (int)offset
		else
		  Ignore Next Instruction

BREAK	Breakpoint	Break Exception
CLO	Count Leading Ones	Rd = NumLeadingOnes(Rs)
CLZ	Count Leading Zeroes	Rd = NumLeadingZeroes(Rs)
COP0	Coprocessor 0 Operation	See Software User's Manual
DERET	Return from Debug Exception	PC = DEPC Exit Debug Mode
DI	Atomically Disable Interrupts	Rt = Status; $Status_{IE}$ = 0
DIV	Divide	LO = (int)Rs / (int)Rt HI = (int)Rs % (int)Rt
DIVU	Unsigned Divide	LO = (uns)Rs / (uns)Rt HI = (uns)Rs % (uns)Rt
EHB	Execution Hazard Barrier	Stop instruction execution until execution hazards are cleared
EI	Atomically Enable Interrupts	Rt = Status; $Status_{IE}$ = 1

ERET	Return from Exception	if SR[2>
		  PC = ErrorEPC
		else
		  PC = EPC
		  SR[1> = 0
		SR[2> = 0
		LL = 0

EXT	Extract Bit Field	Rt = ExtractField(Rs, pos, size)

Instruction	Description	Function
INS	Insert Bit Field	Rt = InsertField(Rs, Rt, pos, size)
J	Unconditional Jump	PC = PC[31:28> || offset<<2
JAL	Jump and Link	GPR[31> = PC + 8 PC = PC[31:28> || offset<<2
JALR	Jump and Link Register	Rd = PC + 8 PC = Rs
JALR.HB	Jump and Link Register with Hazard Barrier	Like JALR, but also clears execution and instruction hazards
JALRC	Jump and Link Register Compact – do not execute instruction in jump delay slot (MIPS16e™ only)	Rd – PC + 2 PC = Rs
JR JR.HB	Jump Register Jump Register with Hazard Barrier	PC = Rs Like JR, but also clears execution and instruction hazards
JRC	Jump Register Compact – do not execute instruction in jump delay slot (MIPS16e only)	PC = Rs
LB	Load Byte	Rt = (byte)Mem[Rs+offset>
LBU	Unsigned Load Byte	Rt = (ubyte))Mem[Rs+offset>
LH	Load Halfword	Rt = (half)Mem[Rs+offset>
LHU	Unsigned Load Halfword	Rt = (uhalf)Mem[Rs+offset>
LL	Load Linked Word	Rt = Mem[Rs+offset> LL = 1 LLAdr = Rs + offset
LUI	Load Upper Immediate	Rt = immediate << 16
LW	Load Word	Rt = Mem[Rs+offset>
LWPC	Load Word, PC relative	Rt = Mem[PC+offset>
LWL	Load Word Left	See Architecture Reference Manual
LWR	Load Word Right	See Architecture Reference Manual
MADD	Multiply-Add	HI | LO += (int)Rs * (int)Rt
MADDU	Multiply-Add Unsigned	HI | LO += (uns)Rs * (uns)Rt
MFC0	Move From Coprocessor 0	Rt – CPR[0, Rd, sel>
MFHI	Move From HI	Rd = HI
MFLO	Move From LO	Rd = LO
MOVN	Move Conditional on Not Zero	if Rt ¼ 0 then Rd = Rs
MOVZ	Move Conditional on Zero	if Rt = 0 then Rd = Rs
MSUB	Multiply-Subtract	HI | LO -= (int)Rs * (int)Rt
MSUBU	Multiply-Subtract Unsigned	HI | LO -= (uns)Rs * (uns)Rt
MTC0	Move To Coprocessor 0	CPR[0, n, Sel> = Rt
MTHI	Move To HI	HI = Rs
MTLO	Move To LO	LO = Rs
MUL	Multiply with register write	HI | LO =Unpredictable Rd = ((int)Rs * (int)Rt)$_{31..0}$
MULT	Integer Multiply	HI | LO = (int)Rs * (int)Rd
MULTU	Unsigned Multiply	HI | LO = (uns)Rs * (uns)Rd

Instruction	Description	Function
NOP	No Operation (Assembler idiom for: SLL r0, r0, r0)	
NOR	Logical NOR	Rd = ~(Rs | Rt)
OR	Logical OR	Rd = Rs | Rt
ORI	Logical OR Immediate	Rt = Rs | Immed
RDHWR	Read Hardware Register	Allows unprivileged access to registers enabled by HWREna register
RDPGPR	Read GPR from Previous Shadow Set	Rt = SGPR[SRSCtl$_{PSS}$, Rd>
RESTORE	Restore registers and deallocate stack frame (MIPS16e™ only)	See Architecture Reference Manual
ROTR	Rotate Word Right	Rd = Rt$_{sa-1..0}$ || Rt$_{31..sa}$
ROTRV	Rotate Word Right Variable	Rd = Rt$_{Rs-1..0}$ || Rt$_{31..Rs}$
SAVE	Save registers and allocate stack frame (MIPS16e only)	See Architecture Reference Manual
SB	Store Byte	(byte)Mem[Rs+offset> = Rt
SC	Store Conditional Word	if LL = 1 mem[Rs+offset> = Rt Rt = LL
SDBBP	Software Debug Break Point	Trap to SW Debug Handler
SEB	Sign-Extend Byte	Rd = (byte)Rs
SEH	Sign-Extend Half	Rd = (half)Rs
SH	Store Half	(half)Mem[Rs+offset> = Rt
SLL	Shift Left Logical	Rd = Rt << sa
SLLV	Shift Left Logical Variable	Rd = Rt << Rs[4:0>

SLT	Set on Less Than	if (int)Rs < (int)Rt
		Rd = 1
		else
		Rd = 0

SLTI	Set on Less Than Immediate	if (int)Rs < (int)Immed
		Rt = 1
		else
		Rt = 0

SLTIU	Set on Less Than Immediate Unsigned	if (uns)Rs < (uns)Immed
		Rt = 1
		else
		Rt = 0

SLTU	Set on Less Than Unsigned	if (uns)Rs < (uns)Immed
		Rd = 1
		else
		Rd = 0

SRA	Shift Right Arithmetic	Rd = (int)Rt >> sa
SRAV	Shift Right Arithmetic Variable	Rd = (int)Rt >> Rs[4:0>
SRL	Shift Right Logical	Rd = (uns)Rt >> sa
SRLV	Shift Right Logical Variable	Rd = (uns)Rt >> Rs[4:0>
SSNOP	Superscalar Inhibit No Operation	NOP
SUB	Integer Subtract	Rt = (int)Rs - (int)Rd
SUBU	Unsigned Subtract	Rt = (uns)Rs - (uns)Rd
SW	Store Word	Mem[Rs+offset> = Rt
SWL	Store Word Left	See Architecture Reference Manual

Instruction	Description	Function
SWR	Store Word Right	See Architecture Reference Manual
SYNC	Synchronize	See Software User's Manual
SYSCALL	System Call	SystemCallException
TEQ	Trap if Equal	if Rs == Rt TrapException
TEQI	Trap if Equal Immediate	if Rs == (int)Immed TrapException
TGE	Trap if Greater Than or Equal	if (int)Rs >= (int)Rt TrapException
TGEI	Trap if Greater Than or Equal Immediate	if (int)Rs >= (int)Immed TrapException
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	if (uns)Rs >= (uns)Immed TrapException
TGEU	Trap if Greater Than or Equal Unsigned	if (uns)Rs >= (uns)Rt TrapException
TLT	Trap if Less Than	if (int)Rs < (int)Rt TrapException
TLTI	Trap if Less Than Immediate	if (int)Rs < (int)Immed TrapException
TLTIU	Trap if Less Than Immediate Unsigned	if (uns)Rs < (uns)Immed TrapException
TLTU	Trap if Less Than Unsigned	if (uns)Rs < (uns)Rt TrapException
TNE	Trap if Not Equal	if Rs != Rt TrapException
TNEI	Trap if Not Equal Immediate	if Rs != (int)Immed TrapException
WAIT	Wait for Interrupts	Stall until interrupt occurs
WRPGPR	Write to GPR in Previous Shadow Set	$SGPR[SRSCtl_{PSS}, Rd> = Rt$
WSBH	Word Swap Bytes Within Halfwords	$Rd = Rt_{23..16} \mid\mid Rt_{31..24} \mid\mid Rt_{7..0} \mid\mid Rt_{15..8}$
XOR	Exclusive OR	Rd = Rs ^ Rt
XORI	Exclusive OR Immediate	Rt = Rs ^ (uns)Immed
ZEB	Zero-extend byte (MIPS16e™ only)	Rt = (ubyte) Rs
ZEH	Zero-extend half (MIPS16e only)	Rt = (uhalf) Rs

Tabla 2.2: Juego de instrucciones completo presente en la familia PIC32MX3XX/4XX.

Como podemos observar existen 124 instrucciones diferentes, por lo que dada su extensión y su dificultad, la familia de los microprocesadores PIC32 suelen programarse en lenguaje de programación de alto nivel, accediendo a los registros específicos usando las funciones proporcionadas por el compilador o asignado directamente los unos y los ceros en las posiciones deseadas.

3.1.4.1. Registros de la CPU

Tal y como hemos comentando antes los registros de la CPU son los siguientes:

- 32 registros de propósito general de 32 bits.
- 2 registros de propósito general para almacenar el resultado de las multiplicaciones, divisiones y operaciones de multiplicación acumulativa (HI y LO).
- Y un registro de propósito especial, contador de programa (PC), al cual solo le afectan indirectamente ciertas instrucciones, tal y como hemos visto anteriormente, las interrupciones y excepciones. Este registro no es un registro visible en la arquitectura.

En la siguiente figura se muestra la distribución de los registros de la CPU:

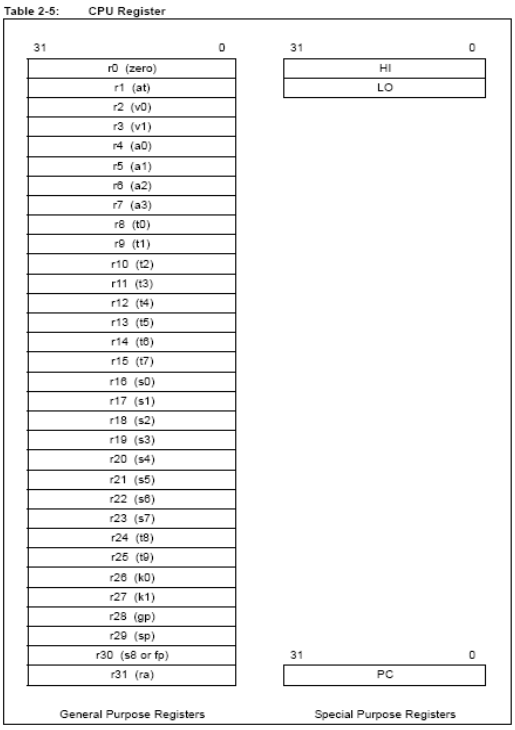


Figura 2.10: Registros de la CPU.

3.1.4.2. Modos del procesador:

Hay tres maneras distintas de ejecución de la CPU del PIC32: dos modos de operación y un modo especial: modo User, modo Kernel y modo Debug. El procesador comienza la ejecución en el modo Kernel y si se quiere se puede permanecer en este modo durante la operación normal. El modo usuario es un modo opcional que permite

al diseñador partir el código entre software privilegiado y no privilegiado. Por otra parte, el modo DEBUG solo se usa solo mediante un depurador.

Una de las principales diferencias entre los distintos modos de operación es el direccionamiento de la memora, de tal forma que el software solo permite el acceso a determinadas regiones de esta. Por ejemplo los periféricos no son accesibles en el modo usuario. La siguiente figura muestra la organización de la memoria en cada modo:

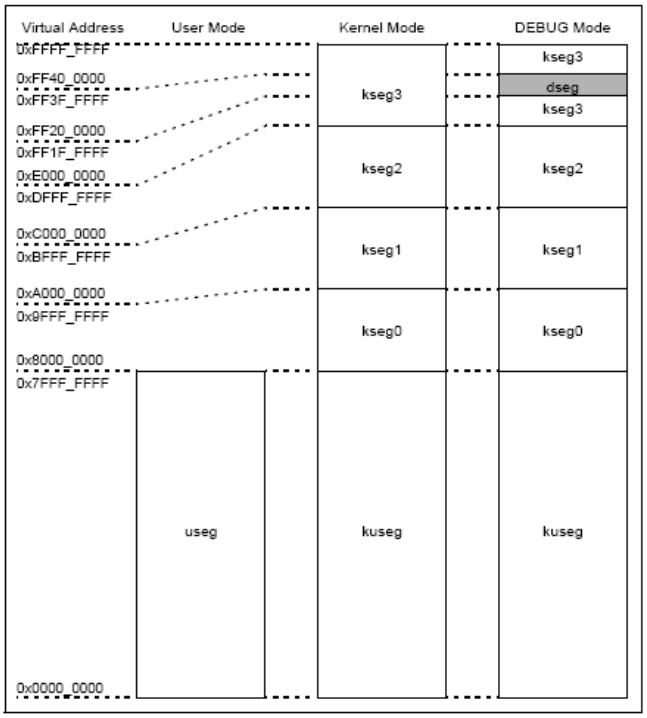


Figura 2.11: Modos de operación de la CPU.

Modo Kernel:

Para poder acceder a todos los recursos hardware, el procesador debe estar en este estado. Tal y como podemos ver en la figura anterior, este modo permite el acceso software a todo el espacio de direcciones del procesador así como también a las instrucciones privilegiadas.

De tal forma que el procesador operará en este modo cuando el bit DM del registro DEBUG sea “0” y el registro STATUS contenga alguno de los siguientes valores: UM=0; ERL=1; EXL=1.

Modo User:

Para poder operar en este modo, el registro STATUS debe contener los siguientes valores: UM=1; EXL=0; ERL=0.

Mientras se está ejecutando el procesador en este modo, el software se restringe a una serie de recursos del procesador. De tal forma, que en este modo se tiene únicamente acceso al área de memoria USEG.

Modo Debug:

Este modo especial del procesador, requiere que para su funcionamiento se ejecute una excepción debug. De tal forma que se accederá a todos los recursos como en el modo kernel así como a los recursos hardwares especiales usados en las aplicaciones de debug. Cuando se entra en este modo el bit DM del registro DEBUG es 1. Para salir de este modo basta con ejecutar la instrucción DERET.

3.1.4.3. Registros CP0:

El PIC32 usa un registro especial como interface para comunicar el estatus y la información de control entre el software del sistema y la CPU. Esta interfaz se llama coprocesador 0. El software del sistema accede a los registros del CP0 usando las instrucciones del coprocesador como MFC0 y MTC0. Los registros CP0 en el MCU del PIC32 son los siguientes:

Table 2-8: CP0 Registers

Register Number	Register Name	Function
0-6	Reserved	Reserved in the PIC32MX core
7	HWREna	Enables access via the RDHWR instruction to selected hardware registers in Non-privileged mode
8	BadVAddr	Reports the address for the most recent address-related exception
9	Count	Processor cycle count
10	Reserved	Reserved in the PIC32MX core
11	Compare	Timer interrupt control
12	Status/ IntCtl/ SRSCtl/ SRSMap	Processor status and control; interrupt control; and shadow set control
13	Cause	Cause of last exception
14	EPC	Program counter at last exception
15	PRId/ EBASE/	Processor identification and revision; exception base address
16	Config/ Config1/ Config2/ Config3	Configuration registers
17-22	Reserved	Reserved in the PIC32MX core
23	Debug/ Debug2/	Debug control/exception status and EJTAG trace control
24	DEPC	Program counter at last debug exception
25-29	Reserved	Reserved in the PIC32MX core
30	ErrorEPC	Program counter at last error
31	DeSAVE	Debug handler scratchpad register

Figura 2.12: Registros CP0.

Entre los registros más importantes del coprocesador destaca el registro STATUS (registro 12 del CP0, selección 0). Este registro STATUS es de lectura y escritura y contiene el modo de operaciones, la habilitación de interrupciones y el estado del diagnostico del procesador.

- Interrupciones habilitadas cuando:
  - IE=1 EXL=0 ERL=0 DM=0
- Modos de operación:
  - DEBUG cuando el bit DM es "1" en cualquier otro caso Kernel o user
  - Modo user cuando UM=1 EXL=0 ERL=0
  - Modo kernel cuando UM=0 EXL=1 ERL=1

La posición que ocupan estos bits son:

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	UM	—	ERL	EXL	IE
bit 7							bit 0

Figura 2.13: Primeros 8 bits del registro STATUS.

3.1.5. MEMORIA DEL SISTEMA

El PIC32 proporciona 4GB (2^{32}) de espacio de direcciones de memoria virtual unificada. Todas las regiones de memoria, incluidas la memoria de programa, memoria de datos, SFRs y registros de configuración residen en este espacio de direcciones con sus respectivas direcciones únicas. Opcionalmente, la memoria de datos y de programa se pueden partir en memoria de usuario y kernel. Además, la memoria de datos puede ponerse como ejecutable, permitiendo al PIC32 ejecutarse desde esta.

Características principales de la organización de la memoria:

- Ancho de datos de 32 bits.
- Separación del espacio de direcciones del modo User y kernel
- Flexibilidad para poder realizar una partición de la memoria flash de programa.
- Flexibilidad para poder realizar una partición de la memoria RAM para datos y otro espacio para programa.
- Separación de la memoria Flash boot para proteger el código.
- Manejo robusto de las excepciones del bus para interceptar el código fuera de control.

- Mapeo de la memoria mediante la unidad FMT(Fixed Mapping Translation)
- Regiones de direcciones cacheables y no cacheables.

En el proyecto **"MANEJO DE UNA PANTALLA TÁCTIL CON EL PIC32 PARA EL CONTROL DE DISPOSITIVOS EXTERNOS"[4]**, se detalla con más precisión la forma en la que se separan los espacios de memoria en modo user y kernel, los registros usados para configurar la memoria del microcontrolador, así como la manera de realizar una partición tanto de la memoria flash de programa como de la memoria RAM y las consideraciones que hay que tener en cuenta a la hora de llevarlo a cabo.

3.1.6. RECURSOS ESPECIALES

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

3.1.6.1. Perro guardián o Watchdog:

El perro guardián está 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.

3.1.6.2. Tecnología de ahorro energético:

Todos los dispositivos de la familia del PIC32 incorporan un rango de características que pueden significativamente reducir el consumo eléctrico durante su funcionamiento. Se debe básicamente a dos características:

- Cambio de reloj sobre la marcha: El reloj del dispositivo puede ser cambiado bajo un software que controle alguno de los 4 relojes durante la operación.
- Instrucciones basadas en modos de ahorro de energía. El microcontrolador puede suspender todas las operaciones, o selectivamente apagar el núcleo mientras deja activo los periféricos, utilizando una instrucción del software.

3.1.6.3. Osciladores:

Toda la familia del PIC32 nos ofrece 4 osciladores diferentes con sus correspondientes opciones, lo que le permite al usuario un gran rango de posibles elecciones en el desarrollo de las aplicaciones hardware. Estas incluyen:

- **FRC:** Internal oscillator, para velocidades altas de operación con un consumo bajo, salida nominal de 8MHz.

- **LPRC:** Internal low-frequency and low-power oscillator. Designado para velocidades de operación pequeñas con bajo consumo, baja precisión, 32KHZ.

- **POSC:** External primary oscillator. Hasta 20MHZ (XT-hasta 10MHz; HS para más de 10MHz), dependiendo si usamos cristal de cuarzo o un resonador cerámico.

- **SOSC:** External low-frequency and low-power, designado para bajas velocidades con un cristal externo de 32,768Khz. Utilizado para temporizar tiempos exactos. Además es necesario si se quiere usar el módulo RTCC (Real-Time Clock and Calendar).

- **EC:** External clock, permite conectar un circuito externo y reemplazarlo por el oscilador, proporcionando al microcontrolador una onda de la frecuencia que queramos.

La señal producida por cada reloj puede ser multiplicada o dividida para ofrecer un ancho rango de frecuencias a través del circuito PLL. Este circuito proporciona un rango de frecuencias desde 32KHZ hasta la frecuencia máxima especificada por el PIC32 80MHZ, comentado en mayor profundidad en el Anexo B del presente proyecto.

3.1.6.4. 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. En las aplicaciones de los PIC, estas se reducen a entender los protocolos. En general en el PIC32 existen disponibles dos tipos, la interfaz de comunicación serie asíncrona (UART) y la síncrona (SPI y I2C).

El PIC32 proporciona 7 formas de comunicación periférica de las cuales 6 son en serie. Estas últimas son:

- 2 Universal Asynchronous Receiver and Transmitters (UARTs).
- 2 SPI y 2 I2C (síncrono).

La diferencia principal entre una comunicación síncrona y otra asíncrona es la necesidad de pasar información temporal desde el que transmite al receptor. Es decir, la comunicación síncrona necesita de una línea dedicada para una señal de reloj que proporcione una sincronización entre ambos dispositivos. Así, el que origina la señal de reloj se le denomina maestro y al otro esclavo.

Interfaz I2C:

Utiliza dos cables, dos pines del microcontrolador: 1 para el reloj (SCL) y otro para trasmitir los datos de forma bidireccional (SDA).

Requiere 10-bits para configurar el dispositivo antes de que se envíe cualquier dato. Esto permite que con el mismo cable se pueda usar hasta 1000 dispositivos.

Interfaz SPI:

Esta interfaz separa la transmisión de datos en dos líneas: una para los datos de entrada (SDI) y otra para los de salida (SDO), por tanto requiere otro cable pero permite transferir datos simultáneamente en ambas direcciones.

Sin embargo, esta interfaz requiere una línea física adicional (selección del esclavo, SS) para conectar cada dispositivo. La principal ventaja de la interfaz SPI es que es muy simple y la velocidad puede ser mucho más alta que con la mayor velocidad del bus I2C.

Interfaz UART:

No requiere una línea de reloj. Existen 2 líneas de datos, TX y RX, las cuales se usan para la entrada y salida de datos, y se pueden colocar adicionalmente dos líneas que pueden ser usadas para proporcionar un hardware handshake (líneas CTS y RTS). La sincronización se obtiene añadiendo un bit de start y otro de stop a los datos.

Generalmente se usa esta interfaz cuando la distancia física es grande.

Interfaz pararela:

El Parallel Master Port (PMP) se ha añadido a la arquitectura del PIC32 para dotar de un bus de E/S flexible para realizar tareas cotidianas del control de periféricos externos. El PMP proporciona la habilidad de transferir datos de 8 o 16 bits de una manera bidireccional y hasta 64K de espacio de direcciones. Además se puede ajustar el tiempo para adecuar el PMP a la velocidad de los periféricos con los que queremos interactuar.

3.1.6.5. 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.

El PIC32 ofrece la posibilidad de convertir la información analógica a digital concretamente a través de un modulo ADC de 10 bits.

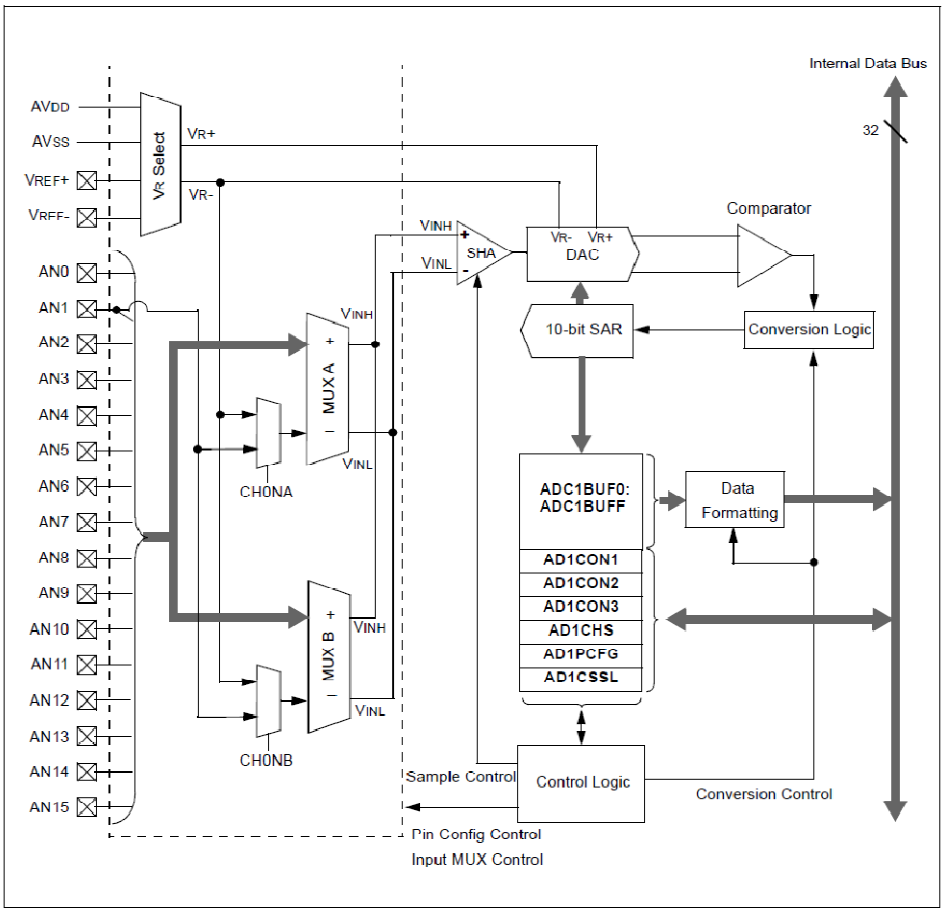


Figura 2.14: Diagrama de bloques del módulo ADC de 10 bits.

Como se puede ver en el esquema de bloques anterior, el módulo A/D pose hasta 16 pines de entradas que se pueden usar para recibir entradas analógicas. Estos pines están conectados a 2 multiplexores para seleccionar los distintos canales y las diferentes referencias para cada uno. La salida del convertidor de 10bits se puede pasar a enteros con signo o sin signo de 16 o 32 bits.

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 , el documento "*PIC32MX Family Reference Manual*" [9] o bien la dirección web del fabricante Microchip [5].

CAPÍTULO 4 ENTORNO DE DESARROLLO

CAPÍTULO 4. ENTORNOS DE DESARROLLO

4.1. INTRODUCCIÓN

En el siguiente capítulo detallaremos el hardware y software utilizado en el proyecto. Comenzaremos describiendo, en el apartado de hardware, la tarjeta de desarrollo EASY PIC 4 en donde se comenzaron a desarrollar los primeros programas para controlar los servomotores y sensores que componen el micro robot. Más tarde se detallarán las tarjetas de desarrollo empleadas en el proyecto : MSE-F87X Y PIC-P40.

4.2. TARJETA DE DESARROLLO EASY PIC 4

4.1.1. INTRODUCCIÓN

El sistema de desarrollo EasyPIC4 de Mikroelektronika [10], 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.

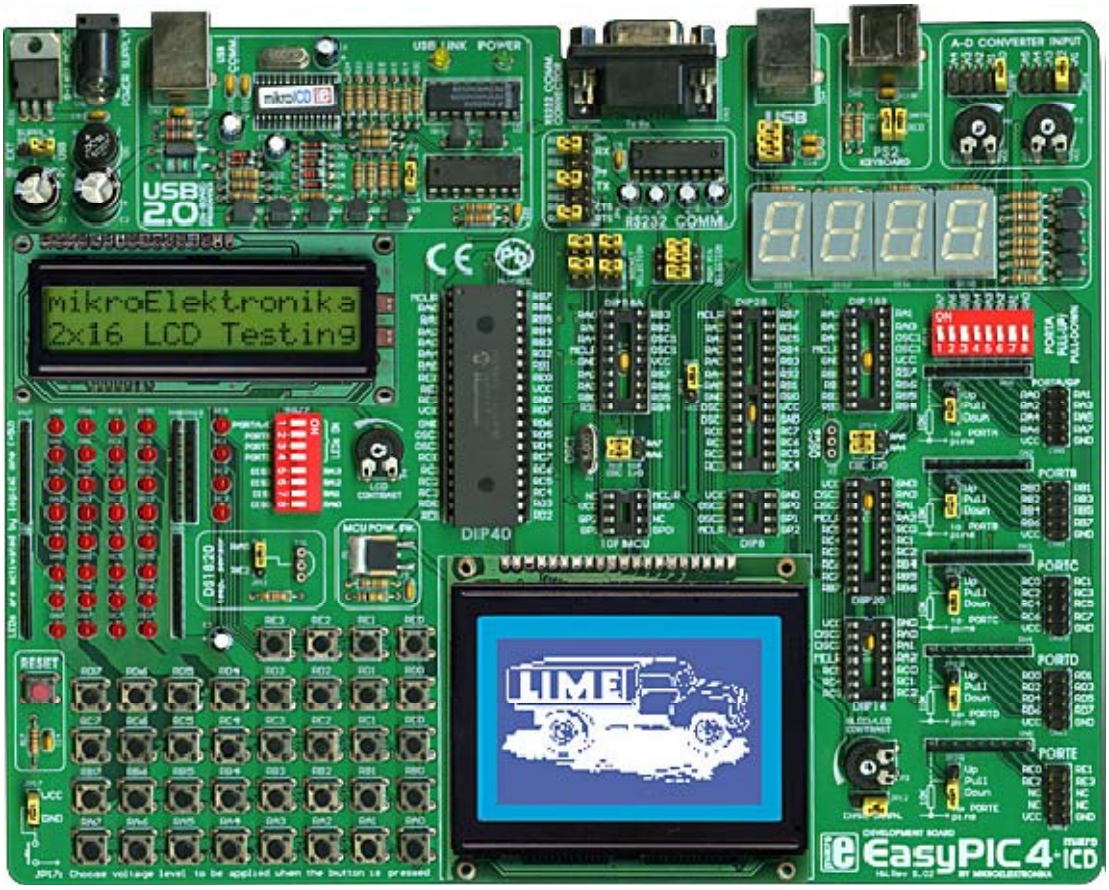


Figura 4. 1 : EASYPIC4 de Mikroelektronika

4.1.2. 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 (ver Figura 4.2):

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.

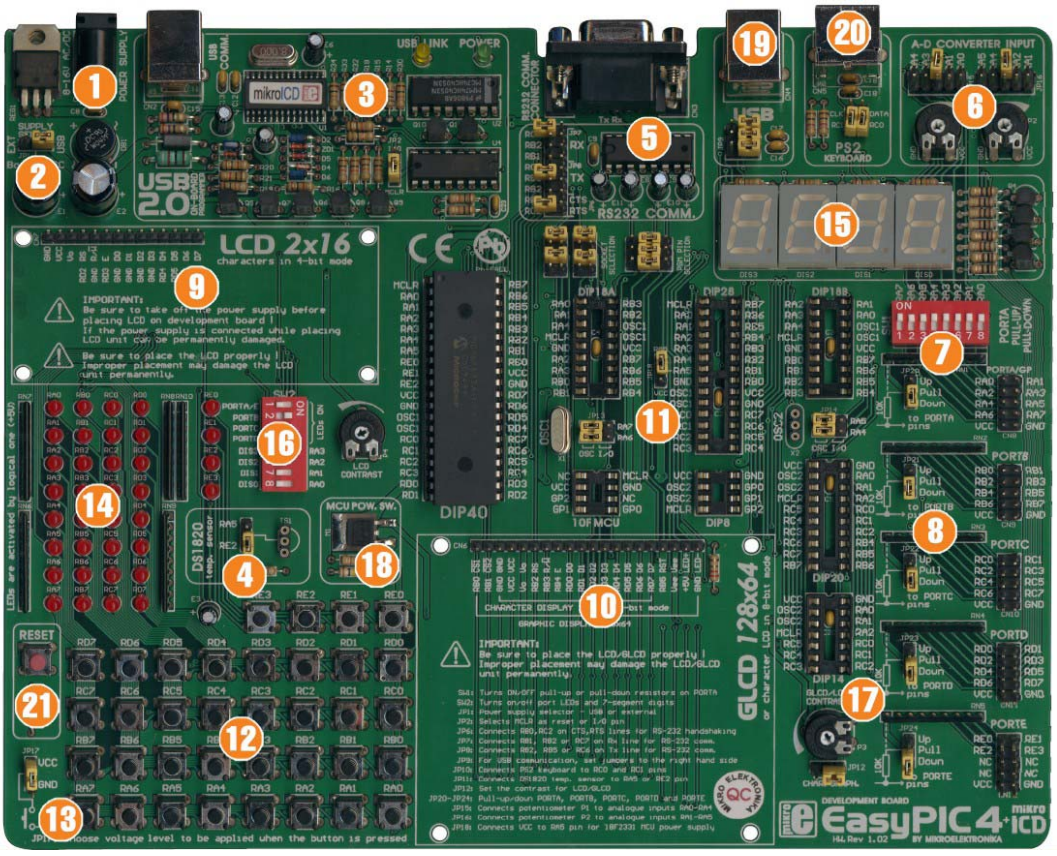


Figura 4. 2.

4.1.3. ARQUITECTURA DEL ENTRENADOR EASYPIC4

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

1. 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 4. 3 : Interruptores

2. 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.

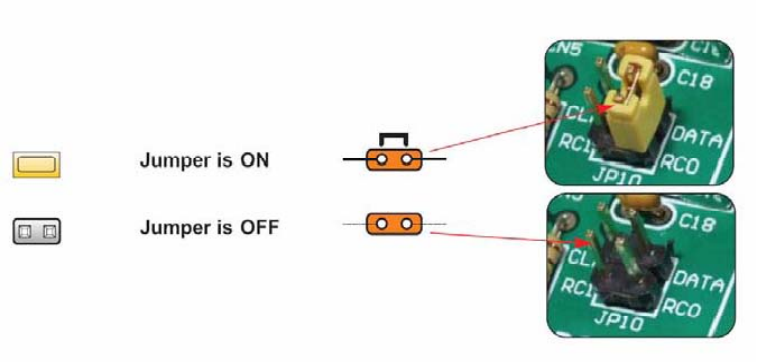


Figura 4. 4 : Jumpers

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.

3. 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.

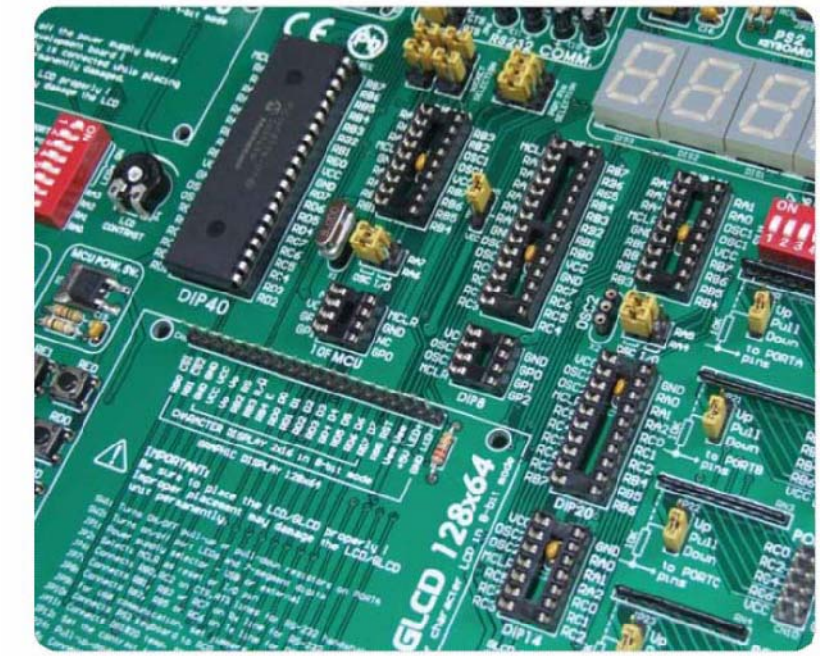


Figura 4. 5: Zócalos MCU

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.

4. 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.

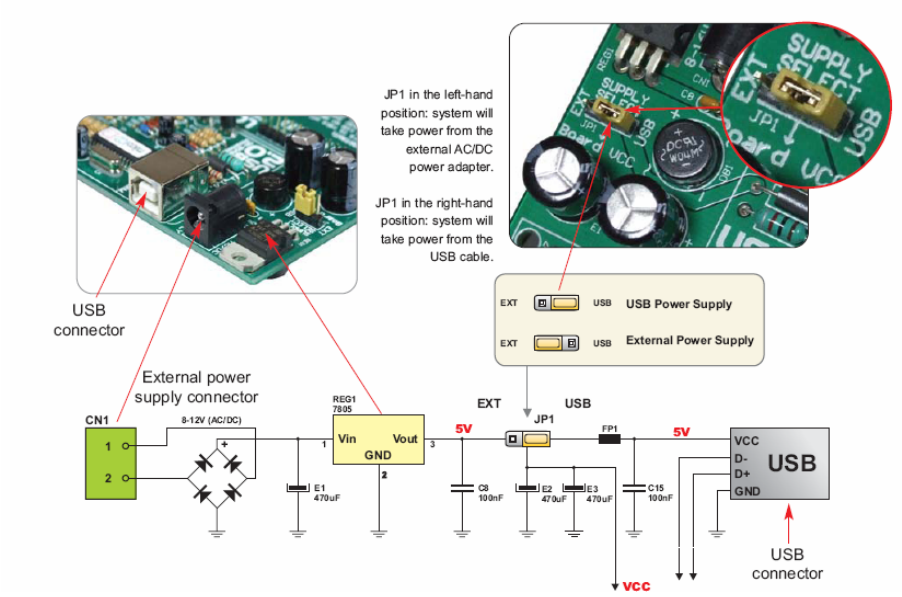


Figura 4. 6 : Fuente de Alimentación

5. 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 4. 7 : **Programador USB**

6. 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.

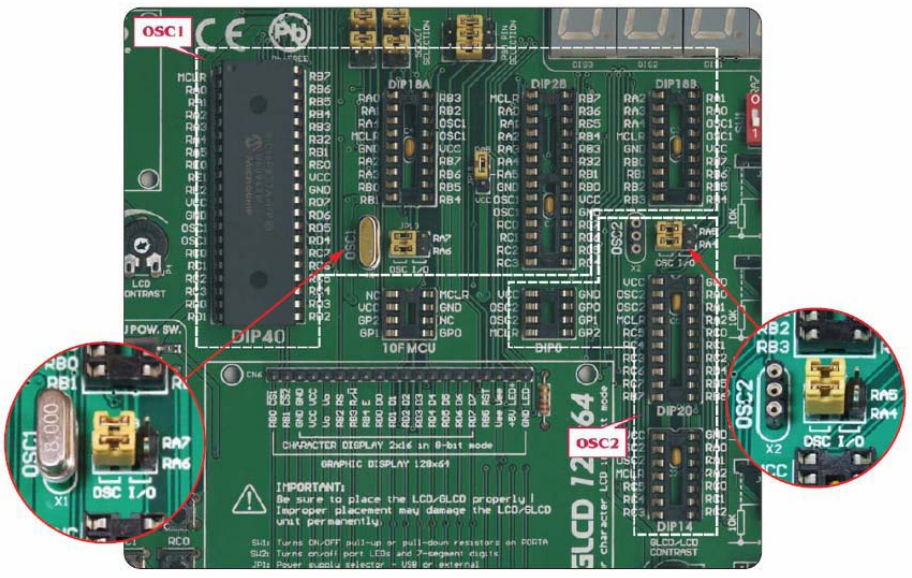


Figura 4. 8 : **Oscilador**

Como se puede apreciar en la Figura 4.9, 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.

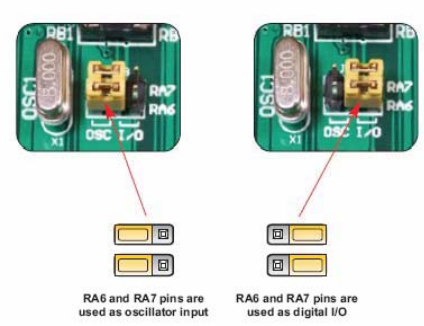


Figura 4. 9 : Pines de entrada y salida

7. 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.

Figura 4. 10 : mikroICD

8. 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.

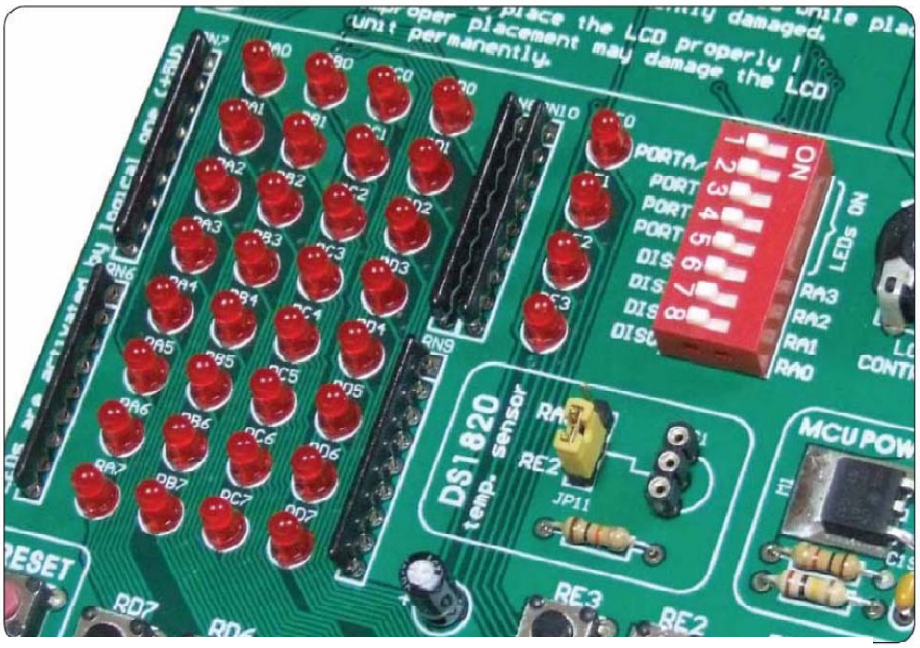


Figura 4. 11: LEDS

9. 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.

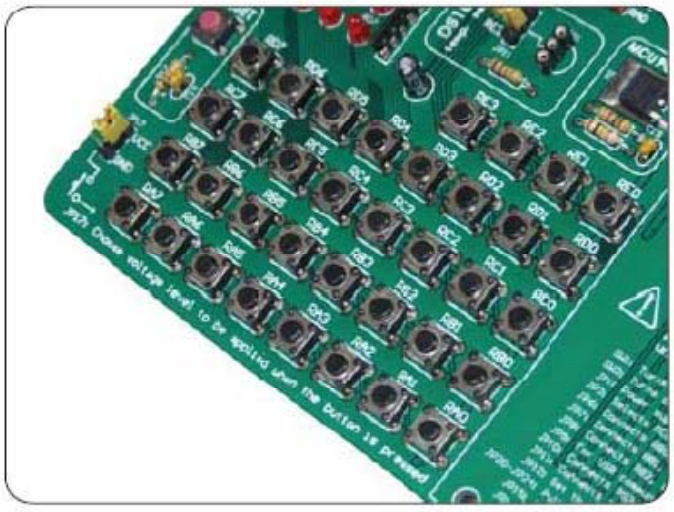


Figura 4. 12 : **Pulsadores**

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 4 .13, 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.

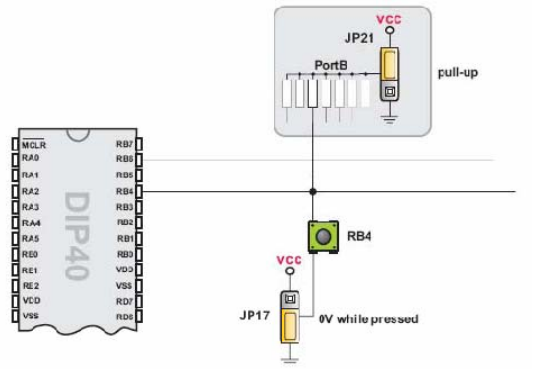


Figura 4. 13

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.

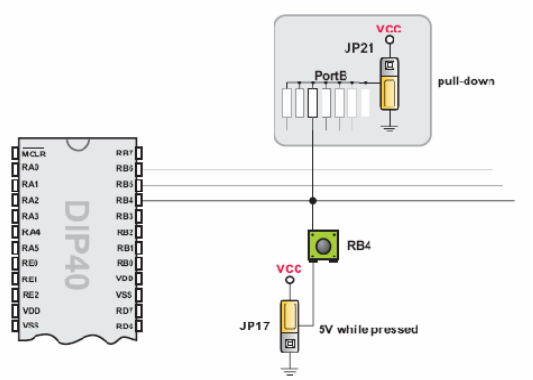


Figura 4. 14

10. 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.

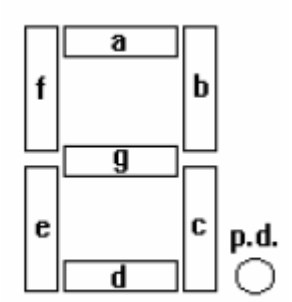


Figura 4. 15: Displays de 7 segmentos

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

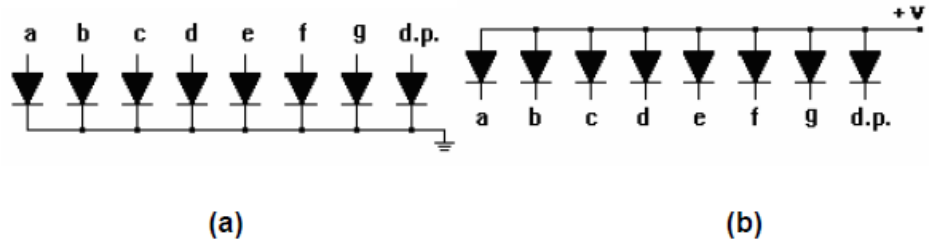


Figura 4. 16 : Estructura de cátodo común (a) y ánodo común (b)

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 4. 17

11. 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 carácter, debe recibir por el bus de datos el código correspondiente.

Figura 4. 18

Además de los caracteres mostrados en la Figura 4.18 , 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.

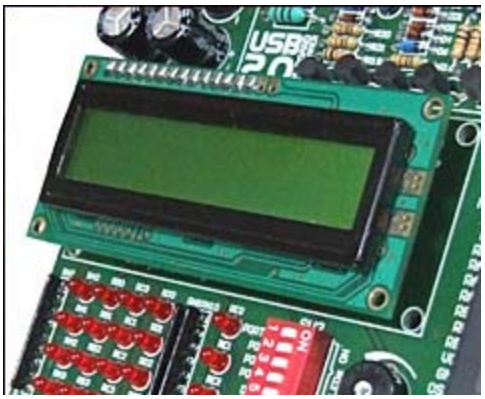


Figura 4. 19 : LCD 2X16 en modo 4 bits

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

Figura 4. 20 : LCD 2X16 en modo 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.

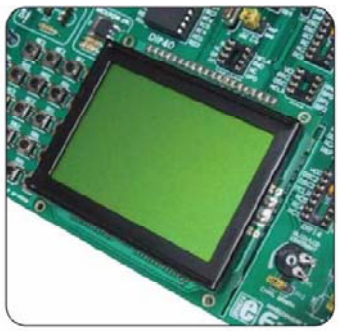


Figura 4. 21 : LCD gráfico

12. 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.

13. 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.

14. 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.

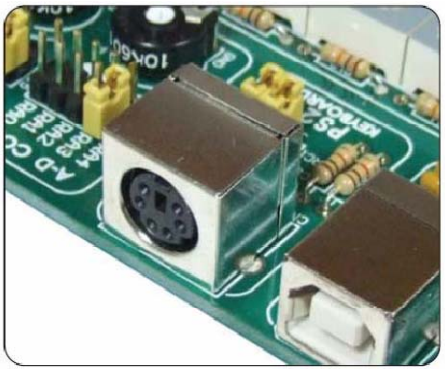


Figura 4. 22 : Comunicación PS/2

15. 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.

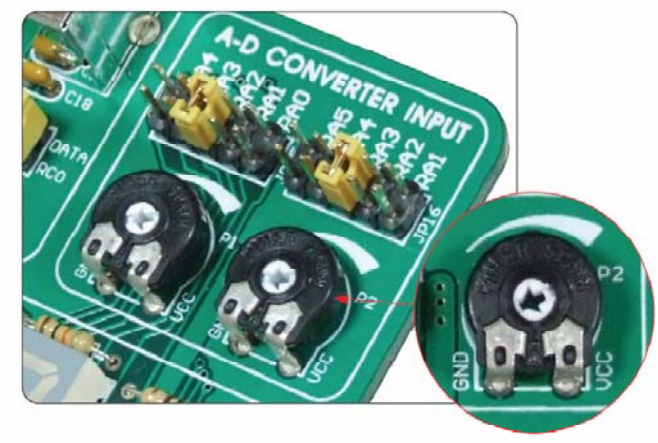


Figura 4. 23 : Conversor A/D

16. 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.

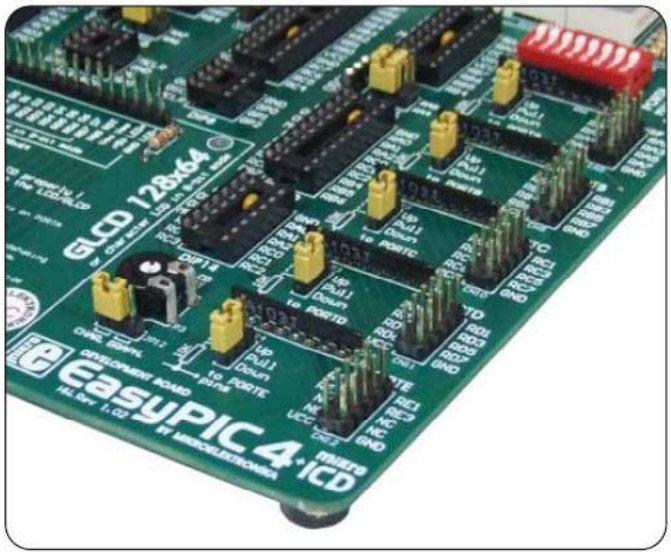


Figura 4. 24 : Accesos director a puertos

4.3. TARJETA DE DESARROLLO MSE-F87X

4.3.1. INTRODUCCIÓN

Al principio de la realización del micro robot se hizo uso de la tarjeta de control MSE-F87X de Ingeniería de Microsistemas Programados S.L[11].

La tarjeta de control MSE-F87X de Ingeniería de Microsistemas Programados S.L. es una tarjeta autónoma, de propósito general, de bajo coste y reducido tamaño, destinada al control directo de diferentes tipos de periféricos según el programa de aplicación del usuario. Se muestra en la Figura 4.25.

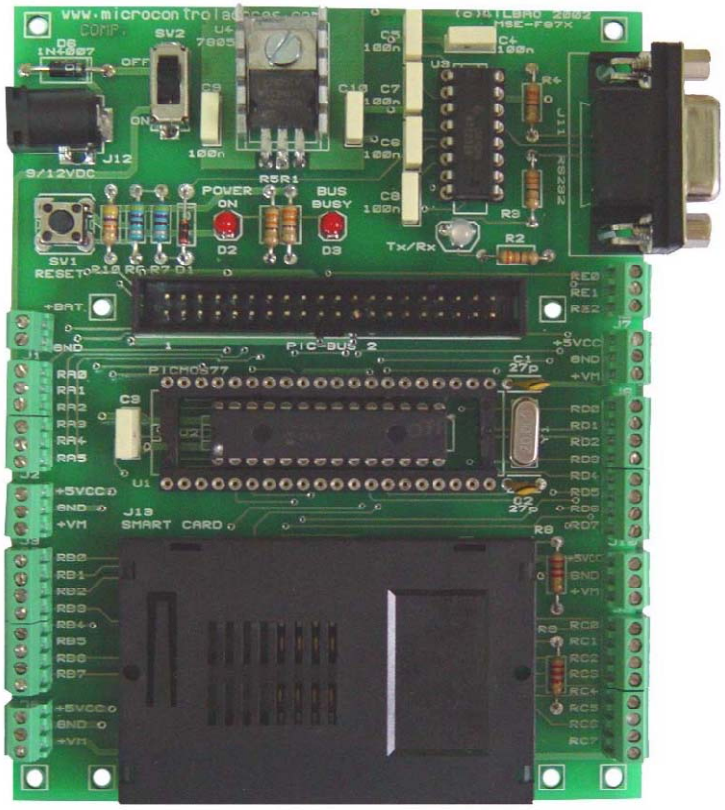


Figura 4. 25 : Tarjeta de control MSE-F87X

Es capaz de soportar los microcontroladores PIC16F873/F876 de 28 patillas o el PIC16F874/F877 de 40 patillas. Dichos microcontroladores pueden venir pre grabados de fábrica con el programa monitor PICMOS desarrollado por Ingeniería de Microsistemas Programados S.L. que, junto con el programa Real_PIC para PC, facilita la edición, el ensamblado, la grabación y depuración de los programas de aplicación del usuario.

La tarjeta de control MSE-F87X incorpora de serie el PICMOS'76 que se basa en el PIC16F876 con el programa monitor PICMOS pre grabado y que dispone de 22 líneas de E/S. Opcionalmente se puede incorporar el PICMOS'77 basado en el PIC16F877 y que dispone de 33 líneas de E/S.

Un sencillo sistema de conexión basado en bornas o clemas con las distintas señales de E/S del PIC, permite una rápida y fácil conexión con cualquier tipo de periférico, entre los que cabe destacar los diversos módulo de sensores y actuadores de la serie MSE-XXXX desarrollados por Ingeniería de Microsistemas Programados S.L.

La tarjeta de control MSE-F87X es totalmente programable por parte del usuario. Según el programa contenido en el PIC que la controla, así como los periféricos conectados a la misma, se determina la aplicación final. Algunas de estas se resumen a continuación:

- Control de aplicaciones industriales y automatismos donde en función de unas señales de entrada procedente de sensores, captadores, transductores, etc. Se activan diferentes mecanismos actuadores como relés, electroválvulas, motores, etc.

- Alarmas, control de presencia, de acceso, seguridad, etc.

- Control de motores de corriente continua (DC) mediante accionamiento digital(ON/OFF), técnicas de PWM para la regulación de velocidad, control de sentido de giro, realimentación, posicionamiento, etc.Control de motores paso a paso (PAP), regulación de velocidad, sentido de giro, posicionamiento, etc.

- Micro robótica. Control de una estructura móvil accionada mediante distintos tipos de motores en función de diferentes sensores que proporcionan información del entorno del robot. Control de trayectorias, posicionamiento, trazado de

rutas predefinidas ,detección de obstáculos, seguimiento de objetos, determinar perímetros o áreas, etc.

4.3.2. CARACTERISTICAS

Las características más relevantes de la tarjeta de control MSE-F87X se resumen a continuación:

- Tarjeta impresa de reducidas dimensiones de 120 x 95 mm
- Alimentación mediante F.A. externa de 6-15VDC o bien mediante packs de baterías de 6 a 15VDC
- Incorpora diodo de protección para polarización inversa así como circuito de filtrado y estabilización.
- Admite microcontroladores PIC de 28 y 40 patillas de la serie PIC16F87X. Estos van insertados en sus correspondientes zócalos.
- Velocidad de trabajo a 20MHz, lo que supone un tiempo de ejecución de 200ns/instrucción.
- Interruptor de ON/OFF y pulsador para Reset manual.
- Conector DB9 hembra estándar para acceso al canal serie incorporado en la tarjeta.
- Leds indicadores de alimentación ON, transferencia de datos I2C y transferencia de datos serie.
- Conector Smart Card que admite las tarjetas Memory Card de *Ingeniería de Microsistemas Programados S.L.* para salvar/recuperar los programas de aplicación. Este conector transporta las señales estándar de los buses I2C y SPI,por lo que es posible conectar otro tipo de tarjetas o periféricos con este tipo de interface.
- Conector de expansión PICBUS-2 que transporta todas las señales del PIC y realizar todo tipo de expansiones externas. Este conector es compatible con la Microsistemas Programados S.L.

- Todas las líneas de E/S están accesibles mediante conjuntos de bornas o clemas que permiten una rápida y fácil conexión con cualquier tipo de periféricos.

4.3.3. ARQUITECTURA DE LA TARJETA DE DESARROLLO MSE-F87X

Se describe a continuación la descripción funcional del hardware de la tarjeta de control MSE-F87X. En la figura 4.26 se muestra la serigrafía con la disposición de los distintos componentes.

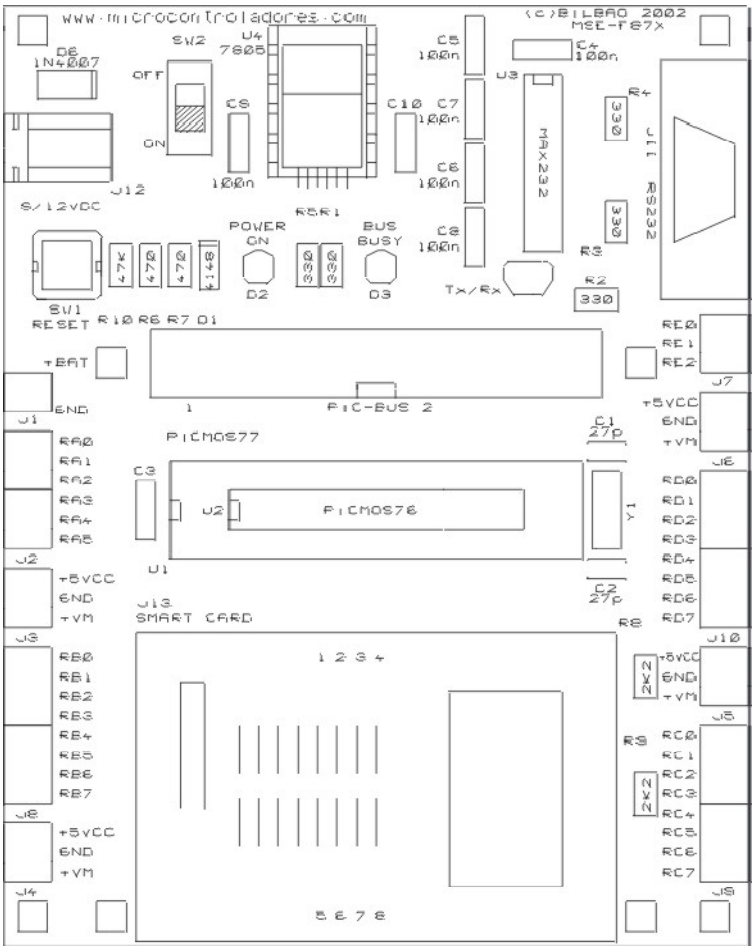


Figura 4. 26 : Serigrafía de la tarjeta de control MSE-F87X

1. El circuito de alimentación

Se muestra en la fotografía de la figura 4.27 se encarga de obtener la tensión de +5Vcc con la que se alimenta toda la electrónica de control del sistema.

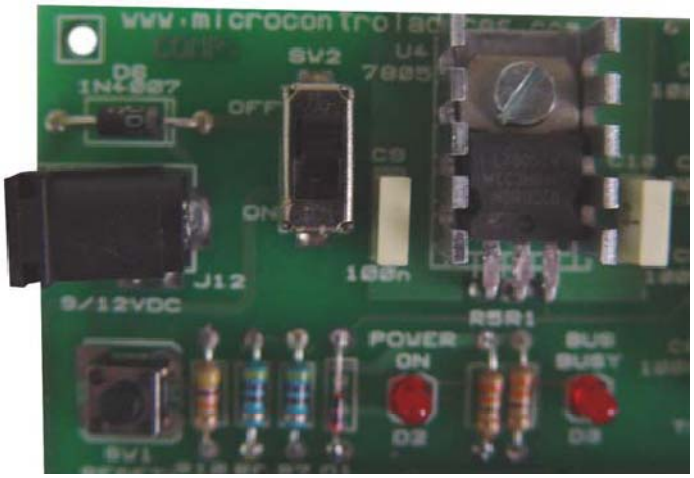


Figura 4. 27

Su esquema eléctrico se muestra en la figura 4.28. Por la borna J1 se conecta la tensión de entrada procedente de una batería o pack de baterías. De este modo la tarjeta de control MSE-F87X se alimenta de forma autónoma. Esta tensión debe ser de 6 a 15 VDC. También es posible alimentarla desde una fuente de alimentación externa, de 6 a 9VDC, aplicada a través de la clavija J12 y con el + al centro de la misma. En este caso la tensión procedente de la batería se desconecta automáticamente.

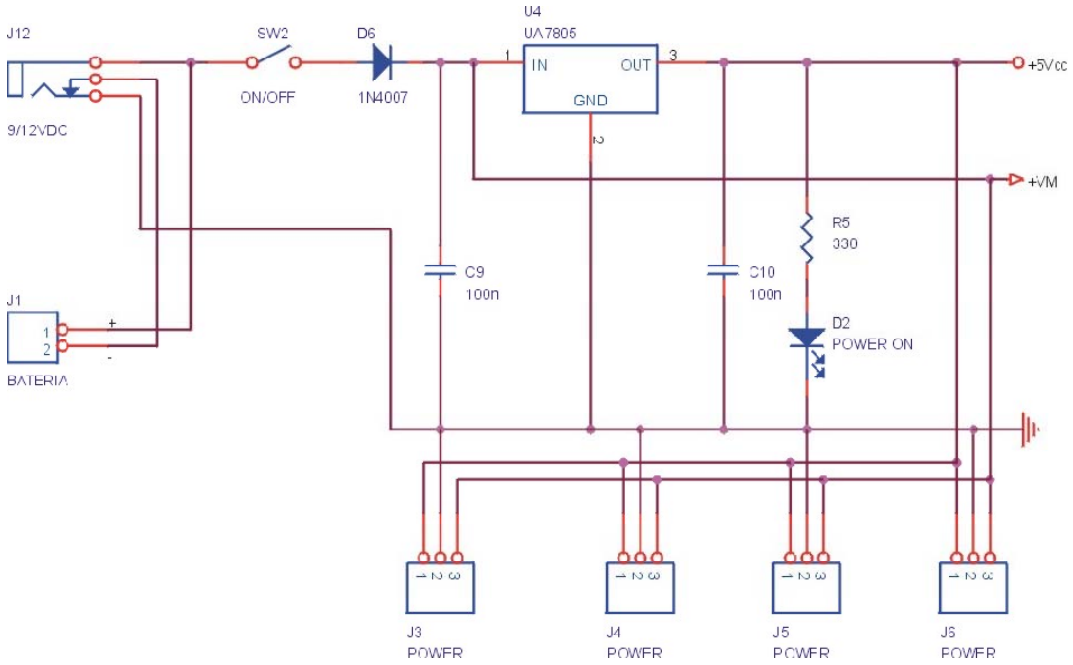


Figura 4. 28

El interruptor SW2 conecta o desconecta la entrada de alimentación al circuito de estabilización. El diodo led D2 monitoriza si la tarjeta de control está o no alimentada. El diodo D6 evita que la alimentación externa se aplique de forma inversa, evitando así daños en el circuito.

La tensión continua de entrada se aplica al estabilizador UA7805 (U4) que se encarga de obtener los +5Vcc con los que se alimenta toda la electrónica de control. La intensidad máxima que soporta el estabilizador es de 1 A. Tanto la tensión de entrada (+VM), la estabilizada (+5Vcc), así como la tierra de alimentación (GND), están disponibles a través de las bornas J3, J4, J5 y J6. De esta forma es posible alimentar directamente los periféricos que sean necesarios sin necesidad de añadir nuevas fuentes de alimentación.

2. Comunicación RS-232

Se muestra en la figura 4.29 y proporciona una canal serie RS-232 estándar para comunicaciones.

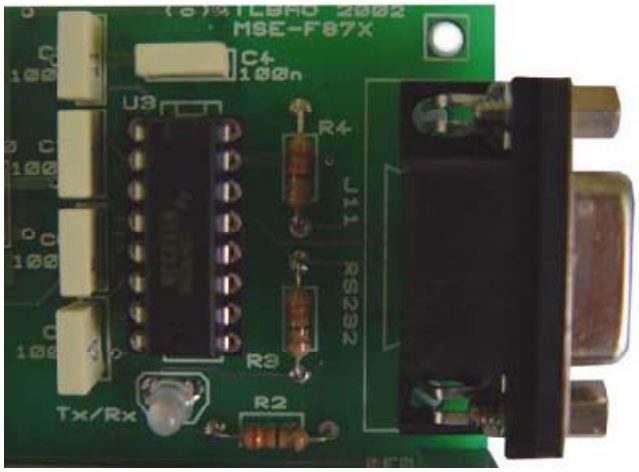


Figura 4. 29

La conexión con el canal serie del PC se realiza mediante el conector estándar DB9 hembra (J11). Las señales de recepción (RxD) y transmisión de datos (TxD) se representan mediante los leds D4 y D5 que pilotan la transferencia de datos. Van a parar al adaptador MAX232 (U3). Este se encarga de convertir los niveles lógicosRS232 a niveles TTL que se aplican a las líneas RC6 (TxD) y RC7 (RxD) del PIC respectivamente. La señal RTS procedente del canal serie se convierte en una señal TTL gracias al MAX232 y se aplica al circuito de reset del PIC (MCLR).

3. Las líneas de E/S

Tal y como se muestra en la fotografía de la figura 4.30 todas las líneas de E/S del PIC están disponibles mediante un conjunto de clemas o bornas que permiten una muy rápida y fácil conexión con los diferentes periféricos que se deseen gobernar.

Figura 4. 30

Las líneas de E/S así como sus correspondientes bornas se agrupan en bloques que representan a las puertas de E/S del PIC. Así, la Puerta A, consta de 6 líneas denominadas RA0-RA5 (J2). Las puertas B, C y D constan de 8 líneas cada una y se denominan RB0-RB7 (J8), RC0-RC7 (J9) y RD0-RD7 (J10) respectivamente. Finalmente la puerta E consta tan sólo de 3 líneas que se denominan RE0-RE2 (J7).

El esquema de la figura 4.31 representa las conexiones eléctricas de todas las líneas de E/S. También se puede apreciar las conexiones del conector de expansión PIC-BUS 2 (J14). Se trata de un conector estándar de 2x 20 con paso 2.54 para cable plano. Contiene todas las señales del PIC y además están distribuidas en el mismo orden que en el propio chip. El PIC-BUS 2 permite conectar cualquier tipo de tarjetas o circuitos de expansión.

Por último cabe destacar las bornas de 3 contactos J3, J4, J5 y J6. Estas transportan la propia tensión de alimentación del sistema: tensión CC de entrada procedente de las baterías o de la F.A. externa (+VM), la tensión de referencia o tierra (GND) y la tensión estabilizada de +5V (+5Vcc). De esta forma resulta muy sencillo alimentar a los diferentes circuitos o periféricos que se están gobernando sin necesidad de añadir distintas tomas de alimentación para los mismos.

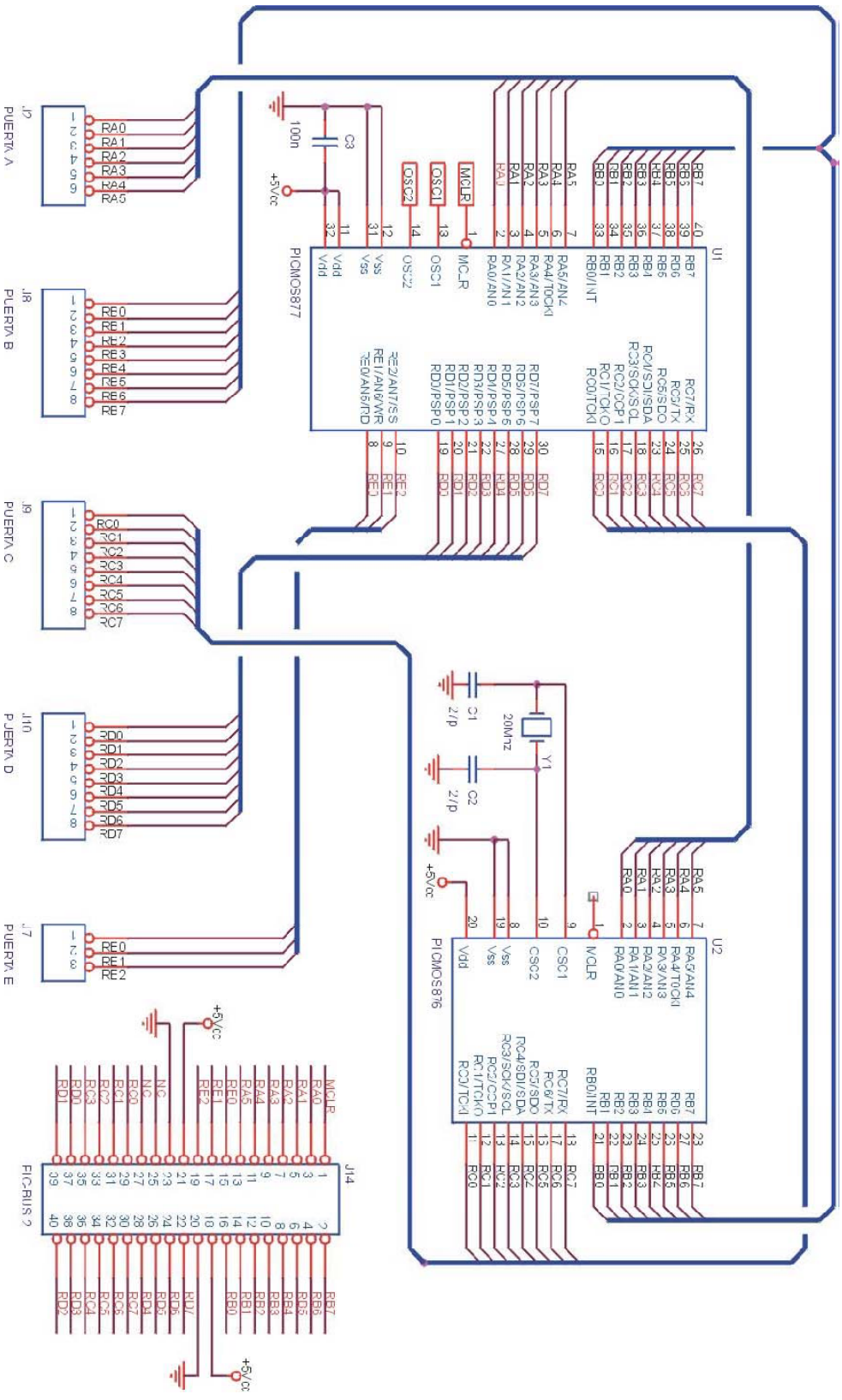

PUERTA A
PUERTA B
PUERTA C
PUERTA D
PUERTA E
PICMOS877
PICMOS876
FIC-RIS 2
100n
20Mhz
+5Vcc

Figura 4. 31

4. El interface SMART-CARD

Se muestra en la figura 4.32. Se trata de un conector Smart-Card estándar que permite insertar diferentes tipos de tarjetas chip: tarjetas de memoria, control de acceso, tarjetas monedero, etc. La disponibilidad de este interface permite que la tarjeta de control MSE-F87X se vea potenciada y apta para un gran número de aplicaciones adicionales

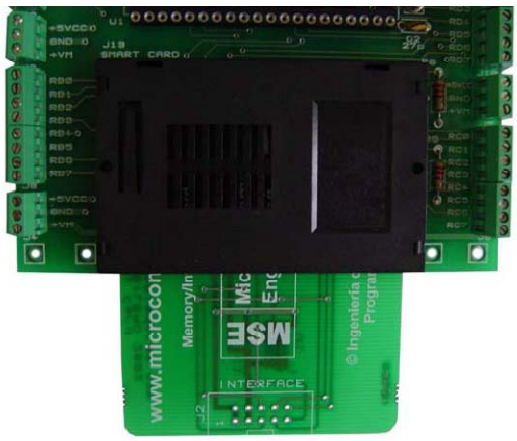


Figura 4. 32

El esquema de la figura 1-10 muestra las conexiones eléctricas de este interface. En el conector Smart-Card están disponibles las señales RC0-RC5 del PIC. Dichas señales componen los buses I2C o SPI estándar. Controlándolas adecuadamente es posible gestionar las múltiples tarjetas chip disponibles en el mercado como son las tarjetas de memoria, tarjetas monedero, tarjetas de control de accesos, tarjetas de identificación, etc.

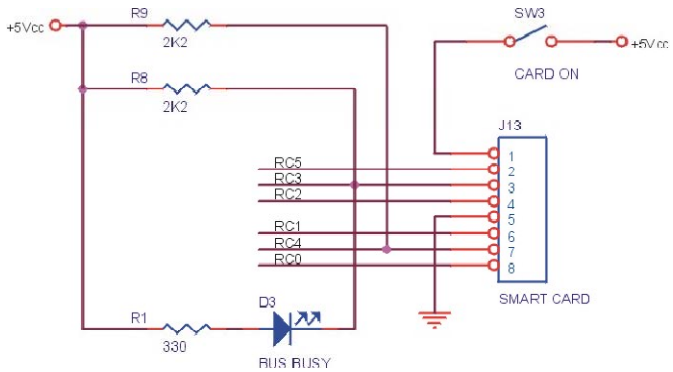


Figura 4. 33

Cada vez que se inserta una tarjeta en el zócalo, el interruptor SW3 integrado en el propio conector, se cierra, aplicando +5Vcc por la patilla 1. Esta tensión permite alimentar a la propia tarjeta insertada. El diodo led D3 indica la transferencia de datos entre el PIC y la tarjeta insertada.

4.4. TARJETA DE DESARROLLO PIC-P40

4.4.1. INTRODUCCIÓN

La tarjeta MSE-F87X fue sustituida por la tarjeta de control PIC-P40 de OLIMEX, puesto que tenía un problema de fabricación. Esta tarjeta será la encargada de controlar al micro robot mediante el uso del microchip PIC16F877A.

La tarjeta de control PIC-P40 de OLIMEX [12] es una tarjeta tipo protoboard , de propósito general, de muy bajo coste y presenta un reducido tamaño, destinada, en este caso , al control directo de los diferentes tipos de periféricos que compondrán nuestro micro robot. Se muestra en la figura 4.34.

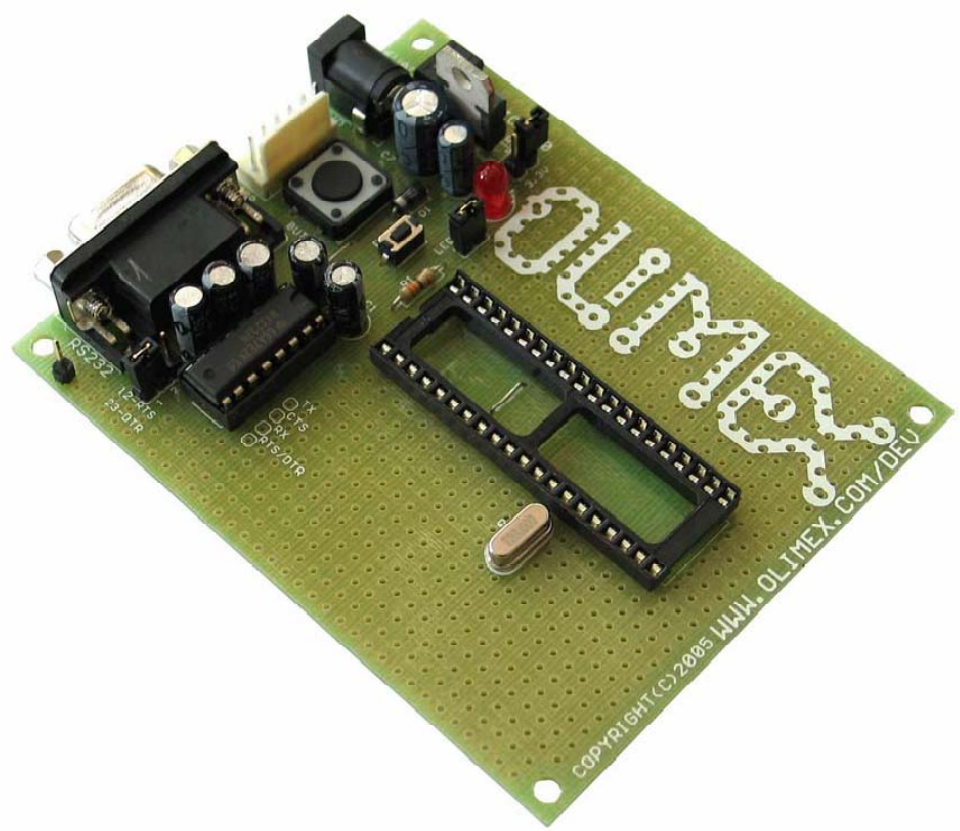


Figura 4. 34: Tarjeta de control PIC-P40 de OLIMEX.

Esta tarjeta no presenta un sistema de conexión basado en bornas o clemas para gestionar distintas señales de E/S del PIC, por lo que se tuvo que realizar diversas modificaciones para conseguir tal comodidad. Se observa que es una estructura más sencilla y presenta menos periféricos.

Como podemos comprobar en la parte inferior de la tarjeta, figura 4.35, está preparada para realizar soldado.

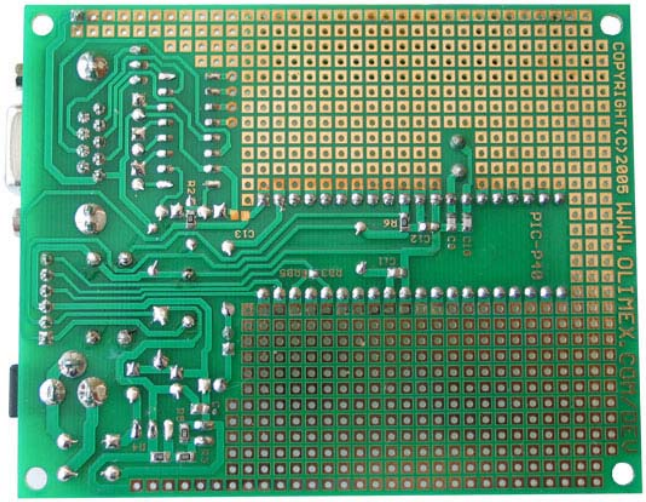


Figura 4. 35

4.4.2. CARACTERISTICAS

Las características más relevantes de la tarjeta de control PIC-P40 se resumen a continuación:

- Conectores ICSP/ICD para programar y debugging
- Interfaz RS232 con MAX232 IC on socket
- Admite microcontroladores PIC de 40 patillas de la serie PIC16F87X. Estos van insertados en sus correspondientes zócalos.
- Velocidad de trabajo a 20MHz, lo que supone un tiempo de ejecución de 200ns/instrucción LED to RA0 through jumper
- Botón para uso cualquiera.

- Botón de reset
- Power plug-in jack, accept AC and DC input
- Voltaje ajustable +3.3V / +5V mediante un jumper
- GND bus
- Vcc bus
- Perforaciones en la placa para realizar las conexiones que desee el usuario
- Tarjeta impresa de reducidas dimensiones de 100 x 80 mm
- Alimentación mediante F.A. externa de 6-15VDC o bien mediante packs de baterías de 5V

4.4.3. ARQUITECTURA DE LA TARJETA DE DESARROLLO PIC-P40

Se describe a continuación la descripción funcional del hardware de la tarjeta de control PIC-P40. En la figura 4.36 se muestra los distintos componentes que forman la placa.

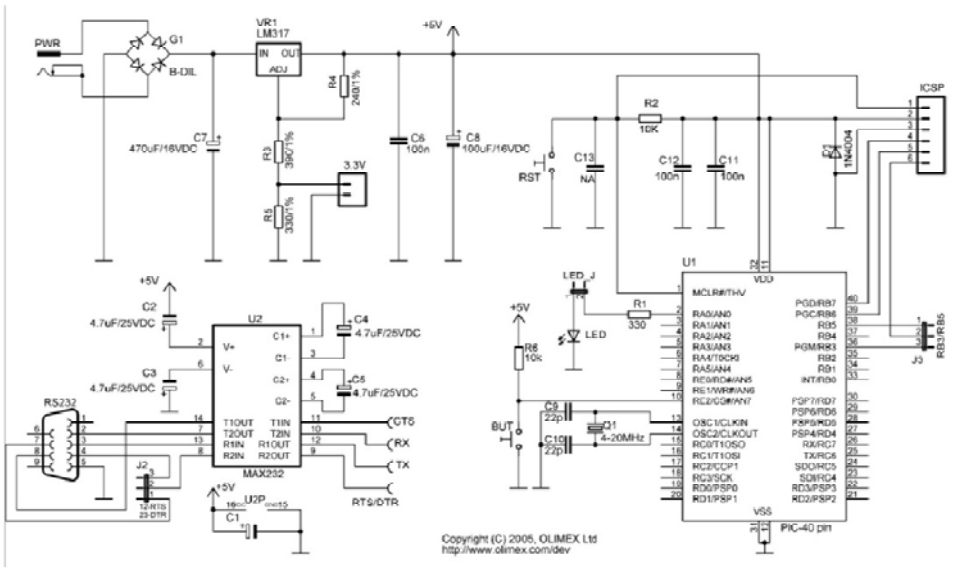


Figura 4. 36

1. El circuito de alimentación

Como Se muestra en figura 4.37 se encarga de obtener la tensión de +5Vcc con la que se alimenta toda la electrónica de control del sistema. Podemos observar los componentes que lo forman.

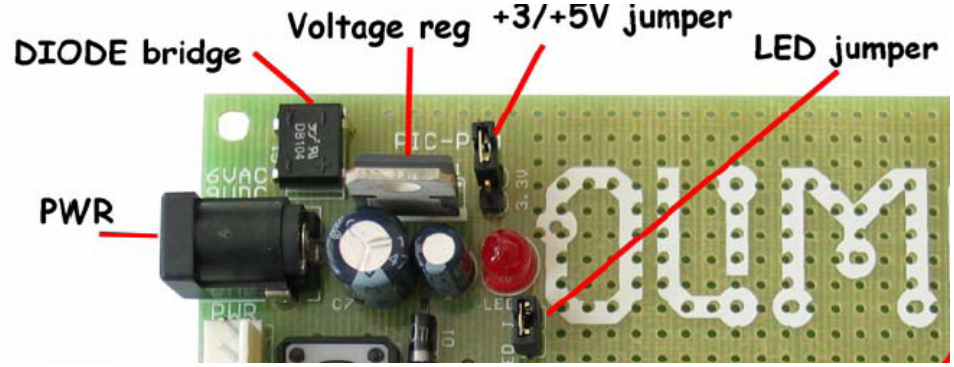


Figura 4. 37

Su esquema eléctrico se muestra en la figura 4.38. Por la borna PWR se conecta la tensión de entrada procedente de una batería o pack de baterías. De este modo la tarjeta de control PIC-PI40 se alimenta de forma autónoma. Esta tensión debe ser de 7.5 a 15 VDC. También es posible alimentarla desde una fuente de alimentación externa, de 6 a 12VAC, aplicada a través de la clavija PWR y con el + al centro de la misma.

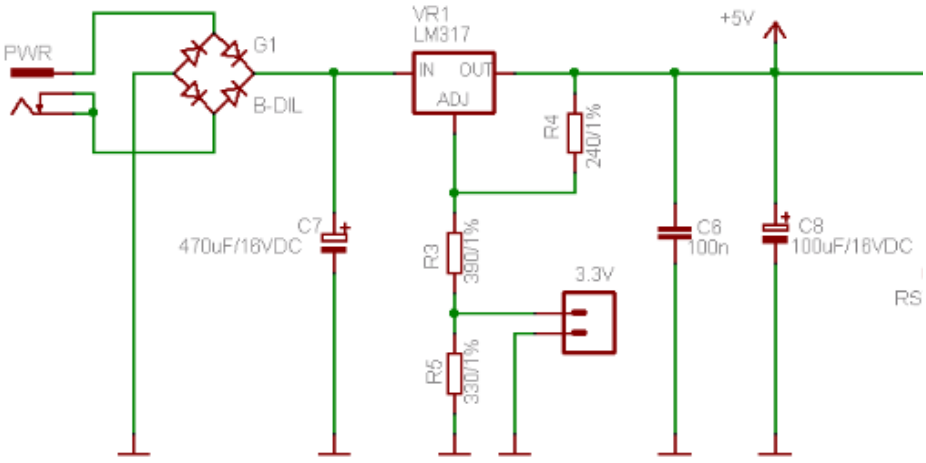


Figura 4. 38

La tensión continua de entrada se aplica al estabilizador LM317(U4) que se encarga de obtener los +5Vcc o 3.3 Vcc con los que se alimenta toda la electrónica de control.

Mediante un jumper podremos seleccionar el voltaje que daremos a nuestro microcontrolador, por tanto el PIC puede trabajar a bajo o normal voltaje .La intensidad máxima que soporta el estabilizador es de 1 A.

Tanto la tensión estabilizada (+5Vcc), así como la tierra de alimentación (GND), están disponibles a través de unos buses. De esta forma es posible alimentar directamente los periféricos que sean necesarios sin necesidad de añadir nuevas fuentes de alimentación.

Se le instaló dos bornes para aportar directamente la alimentación por batería . La alimentación por batería que se aplica a la tarjeta de control PIC-P40 es de 5Vcc. Ver figura 4.39 , círculo rojo 5V y círculo negro 0V.

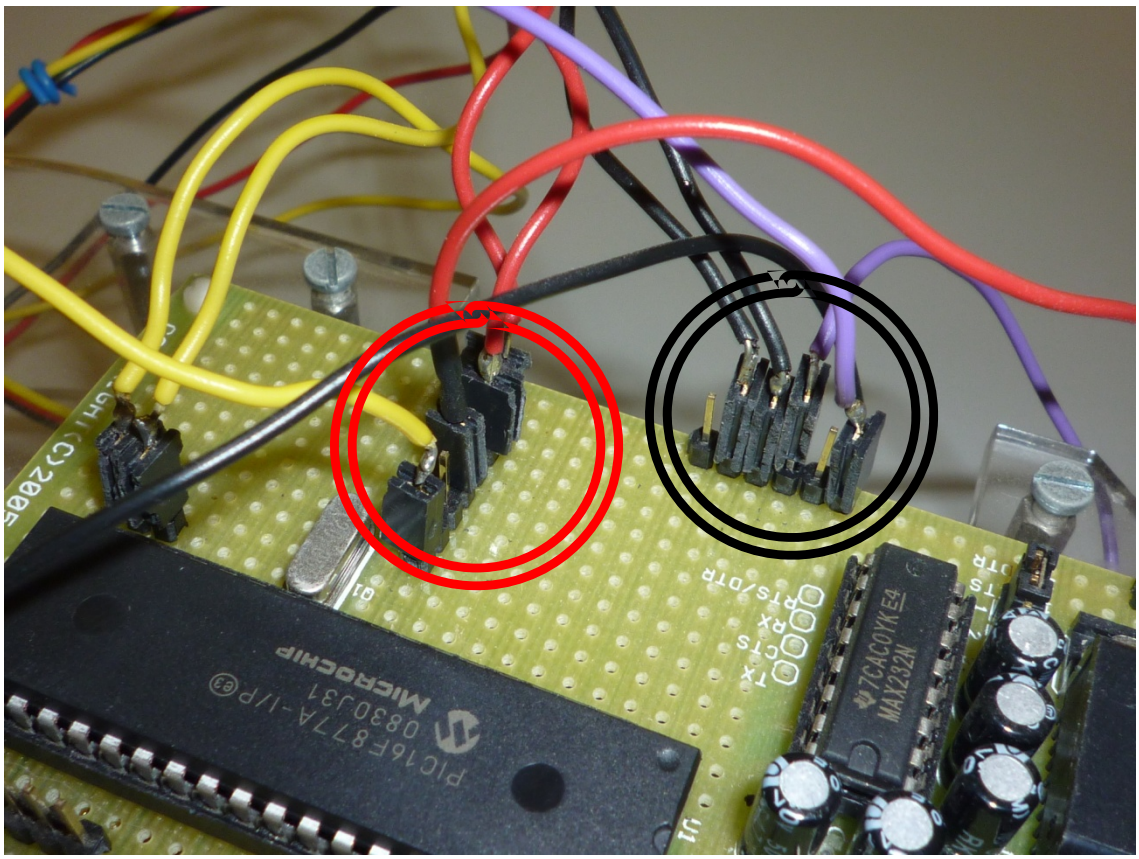


Figura 4. 39

2. Comunicación RS-232

Se muestra en la figura4.40 y proporciona una canal serie RS-232 estándar para comunicaciones.

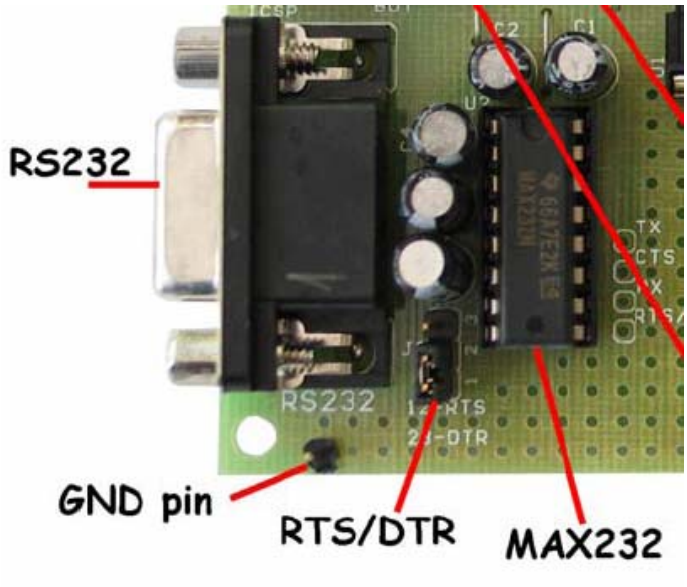


Figura 4. 40

La conexión con el canal serie del PC se realiza mediante el conector estándar hembra .Las señales de recepción (RxD) y transmisión de datos (TxD) se encuentran impresos en color blanco para su posible uso. Van a parar al adaptador MAX232 .Este se encarga de convertir los niveles lógicos RS232 a niveles TTL que se aplican a las líneas (TxD) y R (RxD) del PIC respectivamente. La señal RTS procedente del canal serie se convierte en una señal TTL gracias al MAX232 y se aplica al circuito de reset del PIC (MCLR).

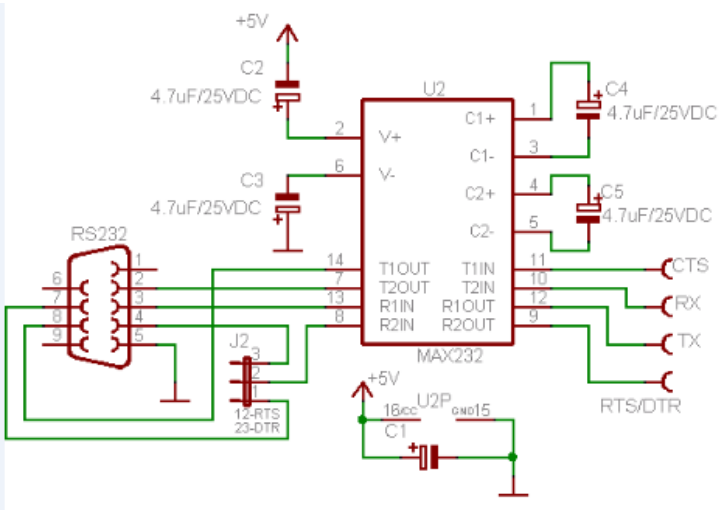


Figura 4. 41: **Esquema puerto de comunicación RS232**

3. Las líneas de E/S

Tal y como se muestra en la fotografía de la figura 4.42 ,círculo azul, todas las líneas de E/S del PIC necesarias están disponibles mediante un conjunto de pines perfectamente soldados para su uso.

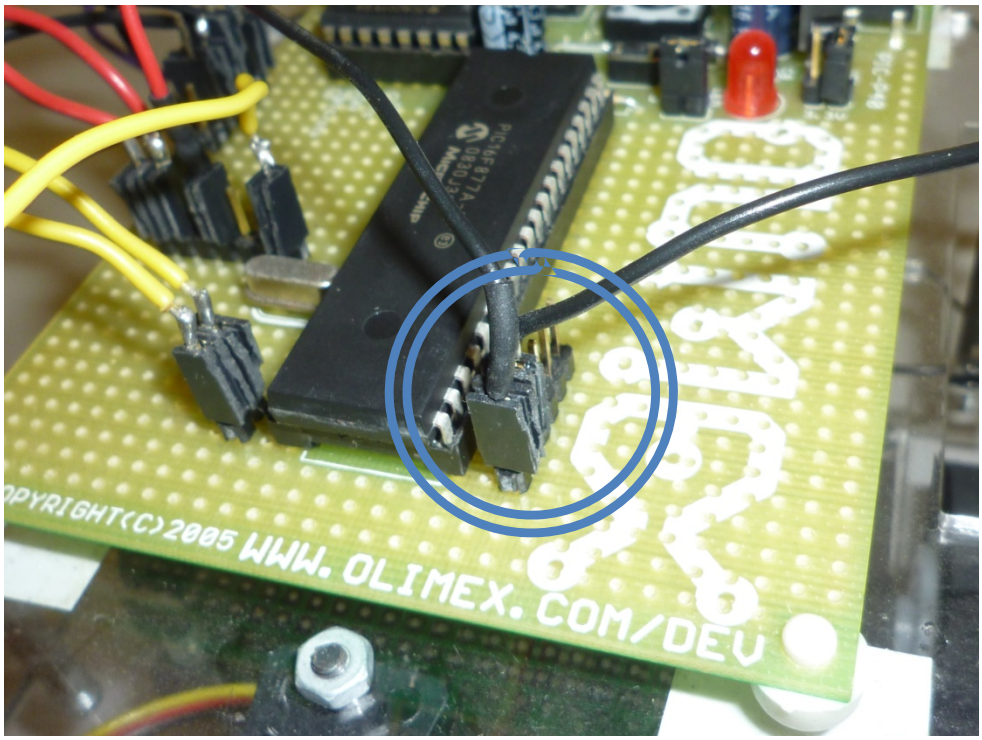


Figura 4. 42

CAPÍTULO 5 SOFTWARE UTILIZADO 16 BITS Y 32 BITS

CAPÍTULO 5. SOFTWARE UTILIZADO

5.1. MPLAB IDE

5.1.1. 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 (ver Figura 5.1).

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

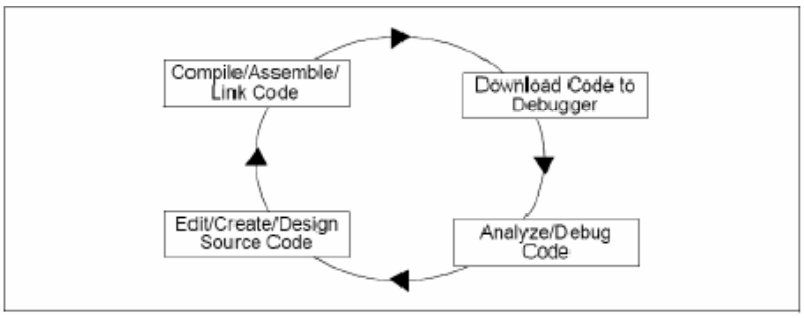


Figura 5. 1:Proceso de escritura de una aplicación

El software MPLAB IDE de Microchip [13], 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 incorporadas, se pueden añadir otras como por ejemplo:

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

Por tanto, 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.

5.1.2. CARACTERÍSTICAS

El software MPLAB IDE, que se encuentra disponible en la página de Microchip , 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.
- Realizar medidas de tiempo con el simulador o el emulador.
- 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.
- Ver variables en Watch Windows
- Programar aplicaciones fijas en los dispositivos mediante programadores.

Es un programa que corre bajo Windows y como tal, presenta las clásicas barras de programa, de menú, de herramientas de estado, etc. El ambiente MPLAB posee editor de texto, compilador y simulación (no en tiempo real).

Si observamos la pantalla inicial del programa podemos observar tres áreas principales, la ventana de edición MPLAB IDE (color azul), ventana de salida (color naranja) y el panel de navegación a la izquierda de la misma (color verde). La ventana de edición es el editor de código, donde vamos a escribir nuestro programa, mientras que en la ventana de salida se mostrará la información y el estado de la última acción realizada.

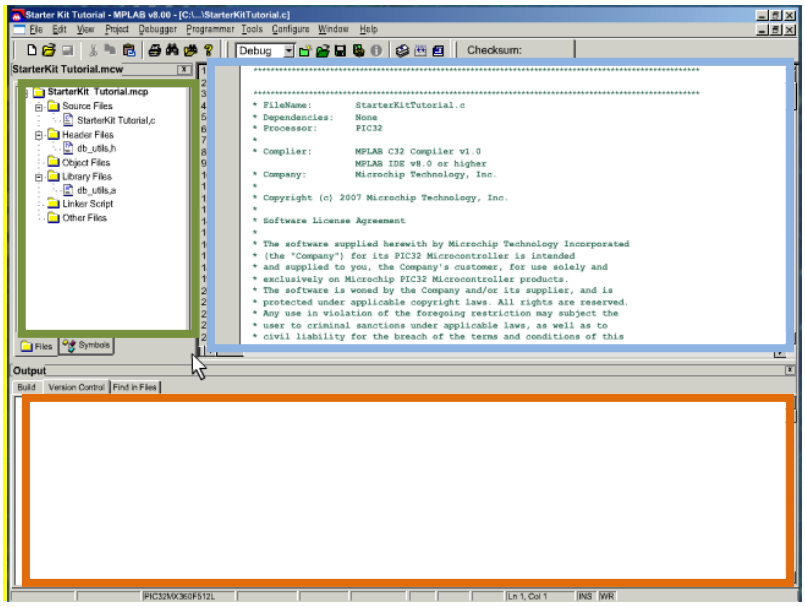


Figura 5. 2 Pantalla inicial del MPLAB IDE.

5.1.3. 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:

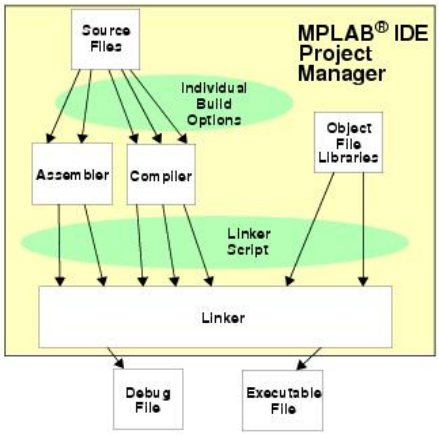


Figura 5. 3: Estructura de un proyecto con el MPLAB IDE.

Los ficheros fuente (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.

5.1.4. CREACIÓN DE UN PROYECTO

Un proyecto contiene todos los archivos necesarios para construir una aplicación (código fuente, ficheros de cabecera, librerías, etc), así como sus correspondientes opciones de construcción. Generalmente habrá un solo proyecto por workspace.

Por otra parte, un worskspace contiene: uno o más proyectos, información del dispositivo seleccionado, la herramienta de programación y depuración y las opciones de configuración del MPLAB IDE.

Además, el MPLAB IDE contiene un asistente para ayudar a crear nuevos proyectos. De tal forma que el proceso de creación de un proyecto se puede dividir en 8 tareas las cuales vamos a ir describiendo paso a paso .

1.-Selección del dispositivo:

Abrimos el MPLAB IDE y lo primero que tenemos que realizar es cerrar cualquier workspace que estuviera abierto, para ello realizamos clic en file>Close. Posteriormente hacemos clic en Project>Project Wizard, para abrir el asistente de creación de un proyecto nuevo, de tal forma que se nos abrirá una ventana como la que se muestra a continuación:

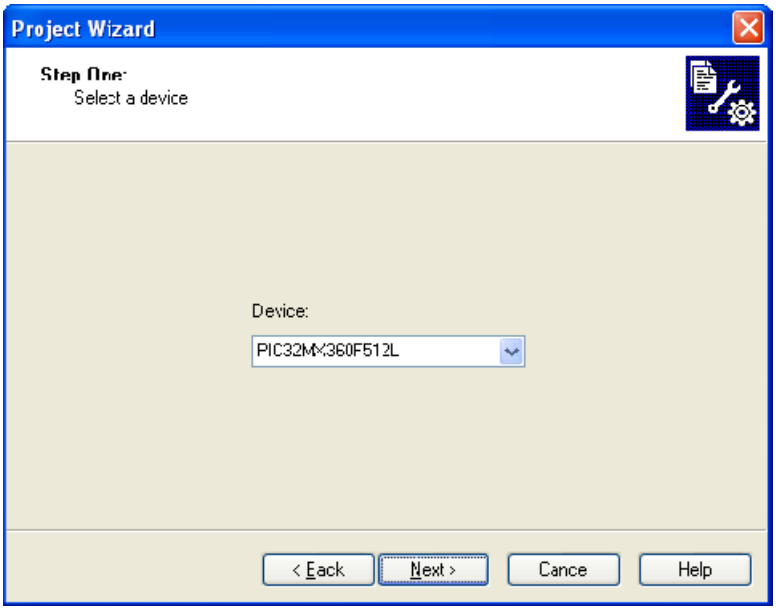


Figura 5. 4: Creación de un proyecto, selección del dispositivo.

En el menú desplegable seleccionamos uno de los dos dispositivos que a continuación mostramos :

- PIC16F877A (8 bits)
- PIC32MX360F512L (32 bits)

A continuación damos a NEXT, para continuar con el proceso.

2.- Selección del Lenguaje de trabajo:

Gama de 8 bits:

El lenguaje utilizado para el PIC16F877A será el ensamblador dado que los compiladores en C que existen en la actualidad para programar estos microcontroladores generan demasiado código innecesario. Y puesto que la memoria de esta gama es limitada se procedió al uso de este lenguaje. Otra razón por la que se usó es dado que se enseña en docencia este lenguaje para aprender a programarlos.

Continuando con el proceso de creación de un proyecto nos saldrá la ventana de la Figura 5.5. Confirmaremos el uso del Microchip Toolsuite pestaña Active Tolsuite, y seleccionaremos MPASM Assembler. Una vez realizado estos pasos damos a Next para continuar.

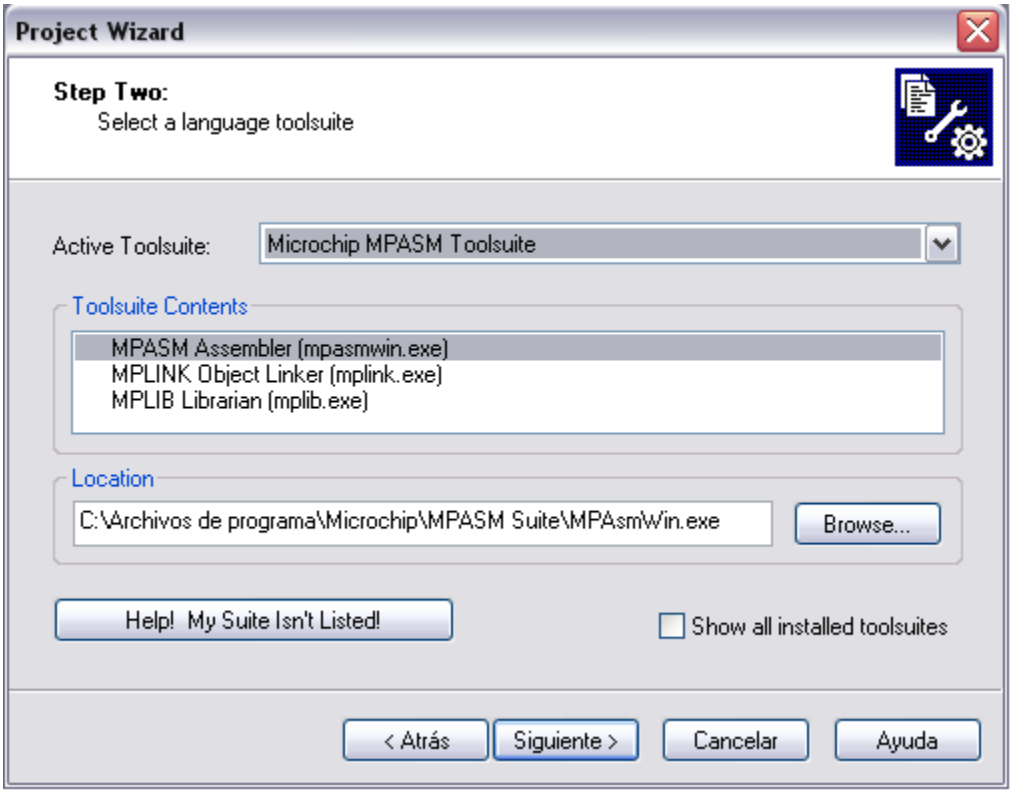


Figura 5. 5 : Creación de un proyecto, selección del compilador

Gama de 32 bits

El leguaje utilizado para el PIC32MX360F512L es un C modificado para poder usar todas las caracteristicas del microcontrolador. El uso de este lenguaje radica en la complejidad de control del mismo. Es por ello que no se usará el lenguaje ensamblador . Es necesario que se haya instalado previamente el compilador C32 [15], antes de continuar.

En caso de escoger un microcontrolador de 32 bitsnos saldrá la ventanta de la Figura 5.5 en "Active Toolsuite" seleccionamos de la lista desplegable "Microchip PIC32 C Compiler ToolsSuite". La herramienta de trabajo incluye el ensamblador y el linkado que usaremos. Si la opción del compilador no está disponible, hay que activar la casilla "show all installed toolsuites

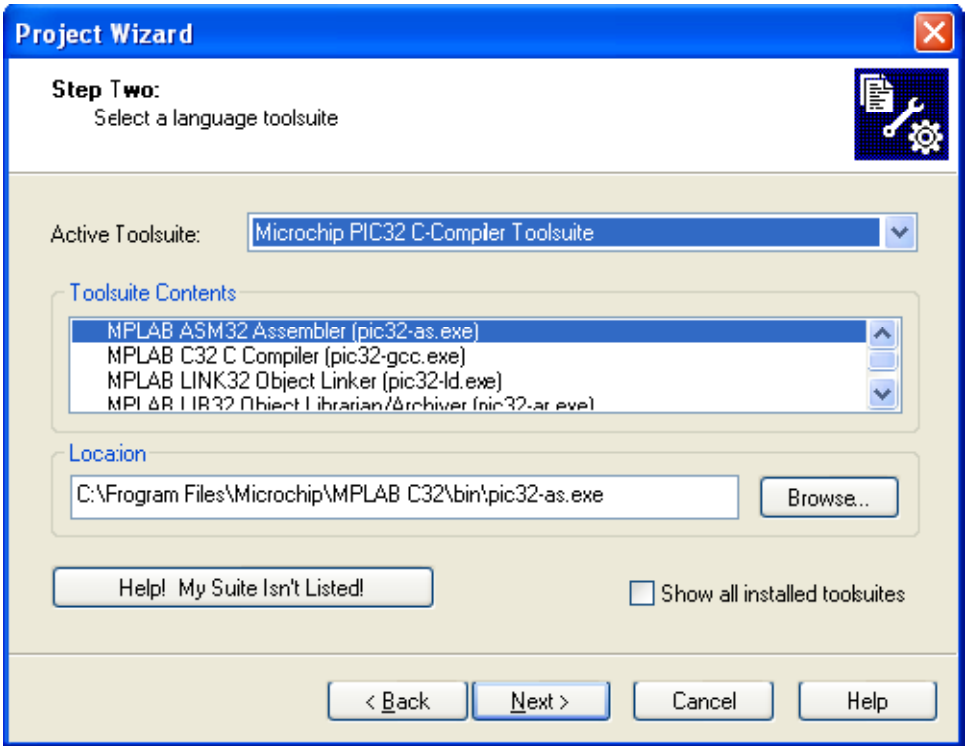


Figura 5. 6 : Creación de un proyecto, selección del compilador y el linkado.

En "Active Toolsuite" seleccionamos de la lista desplegable "Microchip PIC32 C Compiler ToolsSuite". La herramienta de trabajo incluye el ensamblador y el linkado que usaremos. Si la opción del compilador no está disponible, hay que activar la casilla "show all installed toolsuites".

En el desplegable de contenidos de las herramientas, seleccionamos “MPLAB C32 C Compiler (pic32-gcc.exe)” y abajo, en “Location”, hacemos clic en “Browse” y seleccionamos la siguiente ruta que contiene el archivo seleccionado antes: “C:\Program Files\Microchip\MPLAB C32\bin\pic32-gcc.exe”.

Por último seleccionamos en el desplegable anterior “MPLAB 32 LINK Object linker (pic20-ld.exe)” y comprobamos que la ruta es la siguiente:\Program Files\Microchip\MPLAB C32\bin\pic32-ld.exe”. Hacemos clic en NEXT para continuar.

3.- Nombrar el proyecto

A partir de ahora el procedimiento a seguir es valido para los dos microcontroladores .Se nos abrirá una pantalla como la que se muestra a continuación:

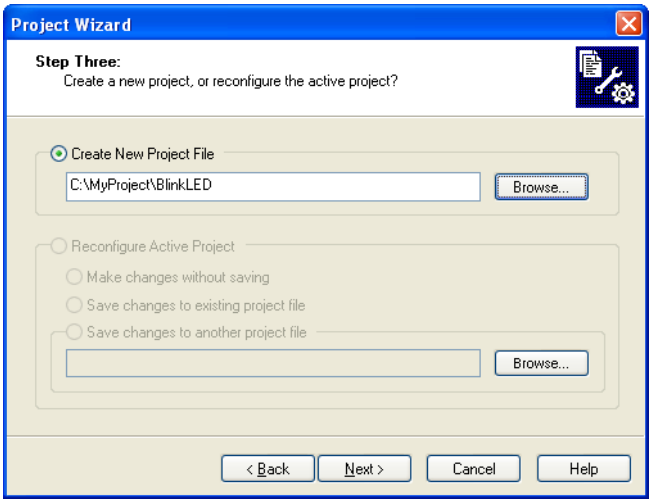


Figura 5. 7: Creación de un proyecto, nombrar el proyecto.

En esta una nueva ventana seleccionamos la carpeta en la que queremos guardar nuestro proyecto y le damos un nombre al mismo, por ejemplo “C:\MiProyecto\Luces”. Hacemos clic en NEXT para continuar.

4.-Añadir los archivos a tu proyecto:

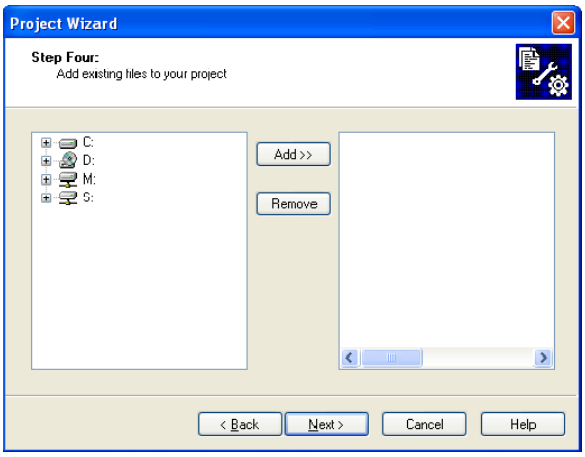


Figura 5. 8 : Creación de un proyecto, añadir archivos al proyecto.

Si los archivos en ".asm" o ".c" aun no los hemos creado podremos saltar esta pantalla y ya los añadiremos posteriormente. Hacemos clic en Finish para terminar y cerramos la ventana. Al cerrar la ventana, se ha creado en el MPLAB IDE el proyecto y el workspace, con los siguientes nombres: el workspace Luces.mcw y el archivo del proyecto Luces.mcp.

Hacemos clic en Files>New en la barra del menú para crear un archivo nuevo en el que comenzar a escribir nuestro programa. Antes que nada, lo guardamos en File>Save As como "Luces.asm" o "*Luces.c*",según el caso, en la misma carpeta en la que hemos creado el proyecto. Es importante escribir la extensión ".c" para que el MPLAB en caso de usar el microcontrolador PIC32MX360F512L y usar ".asm" en caso de usar el microcontrolador PIC16F877A.

Ahora escribimos el código en c para ejecutar el ejemplo. Una vez escrito el código añadimos el archivo "*Luces.c*"o "Luces.asm" , según el microcontrolador, al directorio de fuentes tal y como se puede apreciar en la siguiente figura:

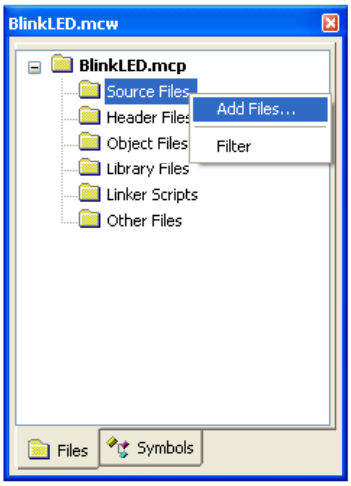


Figura 5. 9 : Añadir archivos al proyecto desde el menú principal.

Posteriormente, seleccionamos nuestro depurador:

Para el microcontrolador PIC16F877A hacemos clic en Debugger>Select Tool> MPLAB SIM.

Para el microcontrolador PIC32MX360F512L hacemos clic en Debugger>Select Tool>PIC32MX Starter Kit de la barra de menú. Antes de seleccionarlo, hay que asegurarse que nuestro Starter Kit [14] esté conectado a nuestro PC a través del cable USB. Una vez lo hayamos conectado, en la ventana de salida del MPLAB, se mostrará un mensaje en el que nos indicará "Starter kit Found", a la vez que el LED naranja de depuración de la tarjeta se encenderá.

6.- Construir el proyecto:

Hacemos clic en Project>Build all en la barra del menú y observamos (en la ventana de salida) que el proceso transcurra sin errores hasta que aparezca el mensaje "BUILD SUCCEEDED", momento en el que el programa estará preparado para que lo probemos .

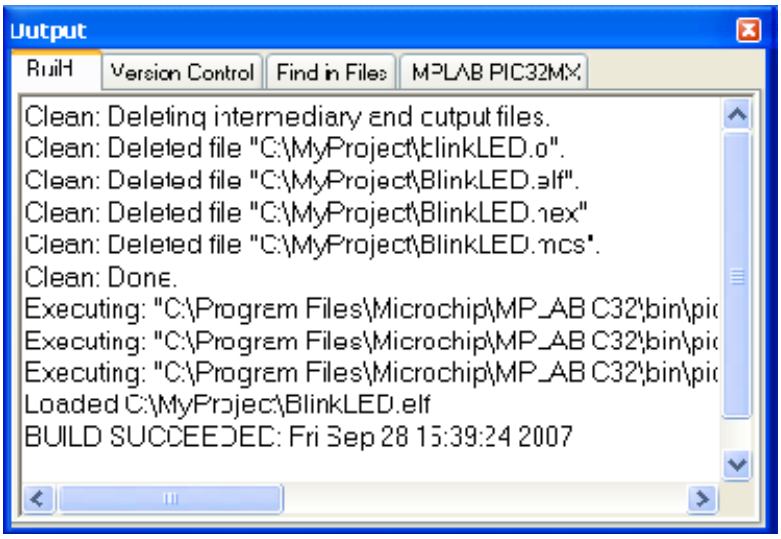


Figura 5. 10: Construcción del programa, build.

7.- Programar el dispositivo y probarlo

Gama de 8 bits:

Se describirá en el apartado 5.2 mediante el uso de dos programas externos.

Gama de 32 bits:

Hacemos clic en el icono de Program All Memories en la barra Herramientas del dispositivo, tal y como muestra la figura, con tal de programar el mismo:

Figura 5. 11 : Programar el microcontrolador.

Una vez pulsado, nos aparecerá un mensaje advirtiéndonos acerca de que vamos a sobrescribir la memoria, le damos a Yes para continuar. En la ventana de salida nos fijamos en el progreso y cuando aparezca “DONE” significará que la

programación del dispositivo ha sido completada. Esto verifica que la imagen ha sido cargada en la memoria Flash y está preparada para su ejecución.

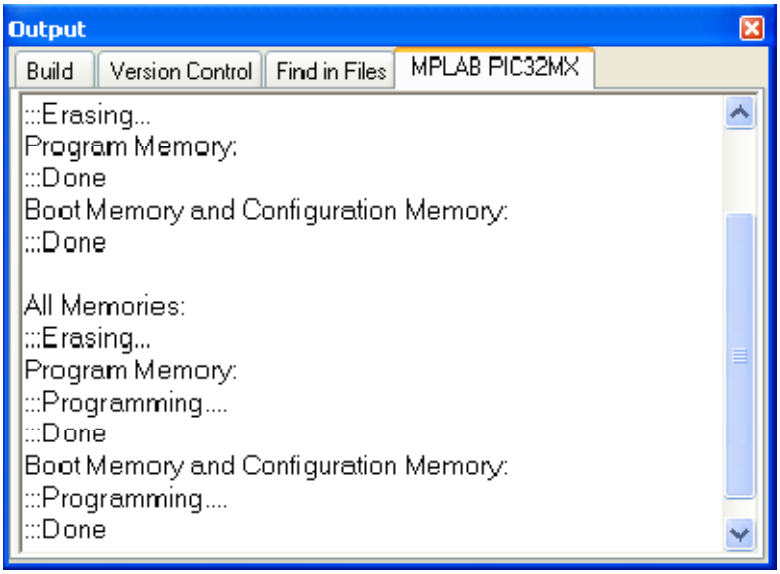


Figura 5. 12 : Verificación de la programación en el dispositivo.

Una vez programada la memoria del dispositivo hacemos clic en Debugger>Run en la barra del menú o en el icono de Run (color azul) en el menú del depurador tal y como indica la siguiente figura, comenzando el funcionamiento del programa en nuestro Starter Kit.

Figura 5. 14 : Ejecutar el programa en modo debugger.

5.2. Software de grabación del microcontrolador 16F877A

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).

5.2.1. SOFTWARE DE GRABACIÓN PICFLASH

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.

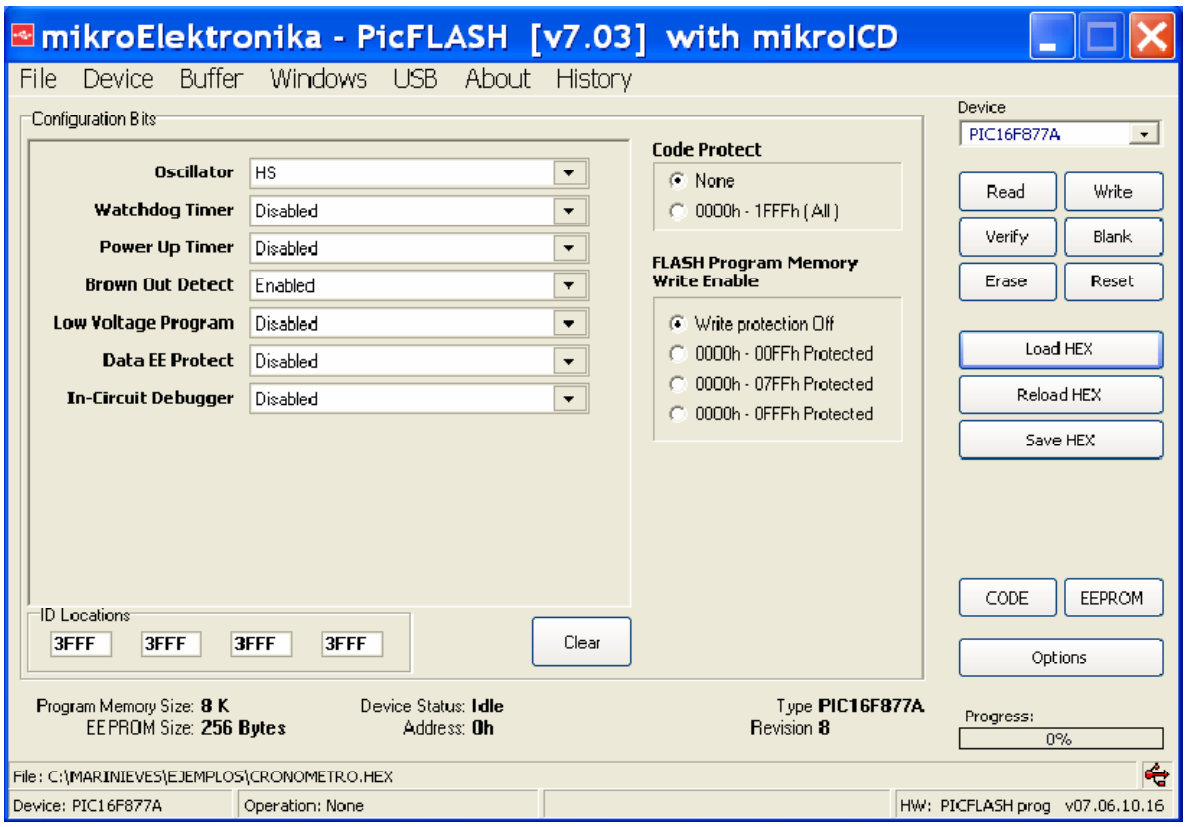


Figura 5. 15 : Programa de grabación PiCFLASH

El microcontrolador es conectado al programador de PICFlash [16] 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.

5.2.2. SOFTWARE DE GRABACIÓN IC-PROG

Dado que la tarjeta de control OLIMEX no posee grabador se tuvo que adquirir uno denominado PIC-PG2 [17] (ver 3.15)

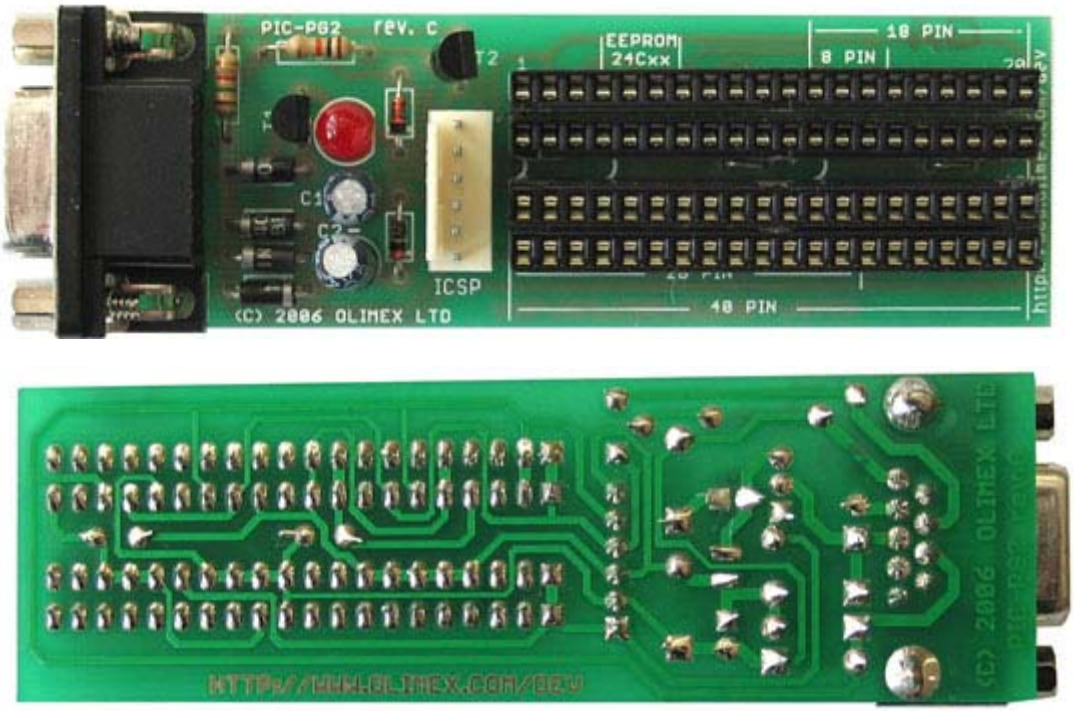


Figura 5. 16 : Programador PIC-PG2

Es un programador de puerto serial destinado para microcontroladores PIC de 8/18/28/40 patillas y para EEPROMS. Posee un conector ICSP para realizar grabaciones externas. Nuestra tarjeta de control OLIMEX poseia una entrada ICSP . Se intento grabar usando este puerto con resultado nulo, es por ello que se procedió al sistema de grabación manual.

Para programar se usó el programa IC- Prog [18] . Este software está destinado para la grabación de microcontroladores PIC de gama baja. Es un software de facil manejo. Antes de nada hay que conectar el programador a unos de los puertos COM disponibles en el ordenador. A continuación insertamos en el microcontrolador PIC en el zócalor del programador respetando la correcta orientación de la cápsula (ver curvas blancas).

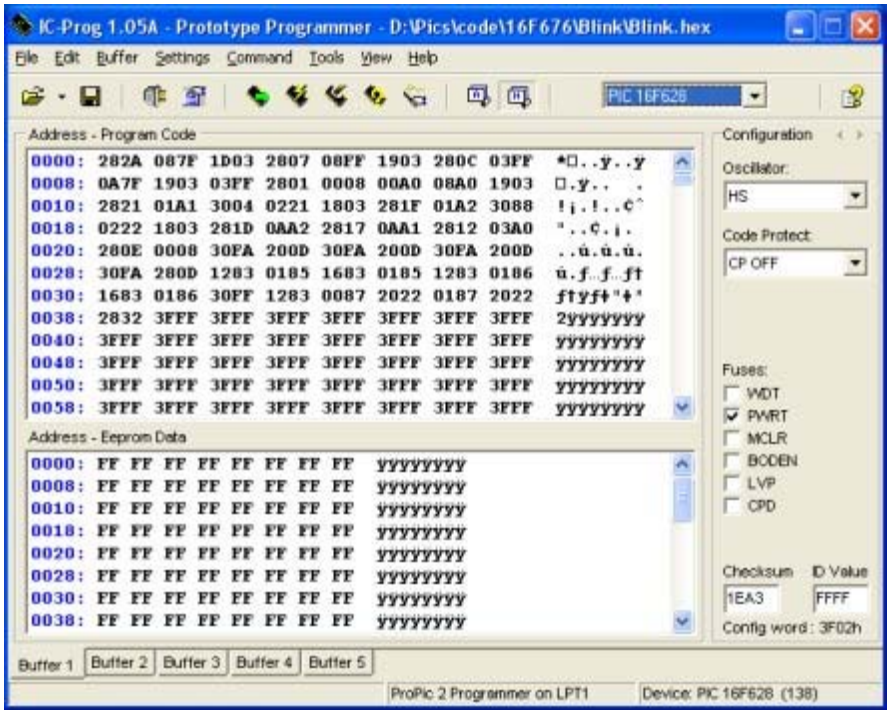


Figura 5. 17: Software de grabación IC-PROG

Ahora hay que empezar a comunicarse con el PG2C. Para configurar el ICProg hacemos click en setting y luego seleccionamos Hardware.

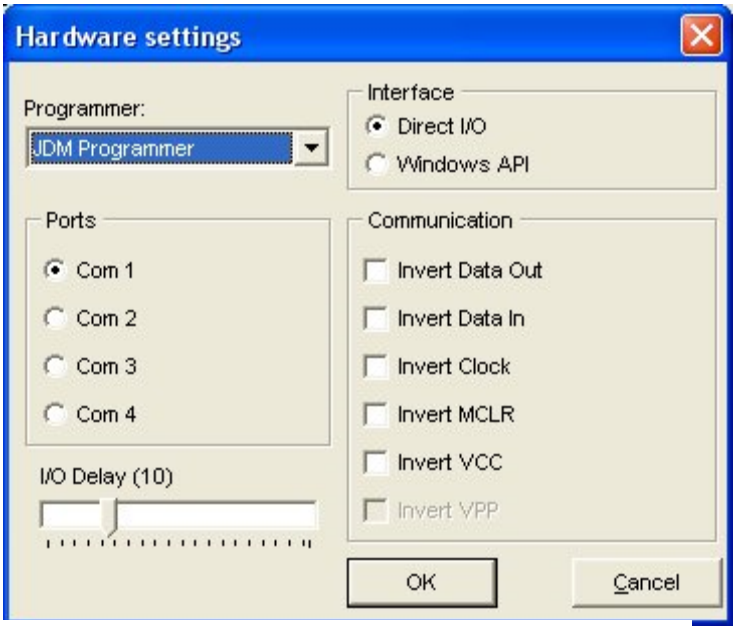


Figura 5. 18: Configuración IC-PROG

Seleccionamos JDM Programmer" desde el menú. Nos Aseguramos de que no este ninguna de las opciones de invertir seleccionadas. Luego seleccionamos "OK" y cerramos la ventana. A continuación seleccionamos el PIC16F877A y modificamos los bits de configuración . Para finalizar damos al botón en forma de rayo.

El proceso de grabación y de verificación se nos mostrará mediante una ventana . En caso de que la grabación haya finalizado con éxito nos informará con otra ventana.

CAPÍTULO 6 DISEÑO Y CONSTRUCCIÓN DE UN MICROBOT

CAPÍTULO 6. DISEÑO Y CONSTRUCCIÓN DE UN MICRO ROBOT

6.1. INTRODUCCIÓN

Este proyecto se centra en el desarrollo de una plataforma para el impulso de algoritmos en robótica móvil. Siendo la robótica móvil un área de la robótica dedicada a la navegación en entornos tanto estructurados como no estructurados, es decir, el robot debe realizar sus tareas en un entorno dinámico, cambiante y en muchos casos desconocido.

En este capítulo se va a detallar la construcción de un micro robot experimental, a partir del cual. A este micro robot experimental le llamaremos **"MYBOT".**

6.2. BREVE HISTORIA DE LA ROBOTICA REACTIVA

En sus orígenes la robótica buscaba conseguir máquinas inteligentes, capaces de desarrollar tareas complejas y de establecer modelos abstractos del entorno. Sin embargo, este entusiasmo y halagüeñas perspectivas fueron perdiéndose poco a poco, ya que la evolución de los resultados no estaba acorde con lo esperado.

Lo cierto es que el entusiasmo inicial no estaban en consonancia con la tecnología del momento, pues en la década de los 70 y 80 se construían robots en los laboratorios que utilizaban cámaras de video como elemento principal para la captación del mundo, pero el tratamiento de la imagen en robótica es algo bastante complejo, que en general requiere de procesos costosos en tiempo, aun con los procesadores actuales.

Cuando los procesadores de la época habían calculado la siguiente acción del robot, el entorno podía haber cambiado sus condiciones, lo que obligaba al robot a recalcular sus acciones. Esto producía que los robots móviles recorriesen habitaciones en plazos de hasta 4 y 5 horas de tiempo.

En la década de los 80 surgió una nueva corriente dentro de la robótica denominada robótica reactiva y posteriormente teoría de agentes. Esta corriente fue encabezada por Brooks [19].

En ella existe una ruptura con el clásico bucle PPA (percepción planificación y acción) usado hasta entonces. Para Brooks era como si se estuviese empezando la casa por el tejado, ya que se intentaba descomponer le razonamiento humano en diversas partes más sencillas, que se utilizan en los denominados sistemas expertos, con la idea de que algún día la unión de estos elementos aislados de la inteligencia humana estuviesen lo suficientemente desarrollados para poder ser unidos en una máquina inteligente.

Sin embargo Brooks era de la opinión de que jamás se conseguiría de este modo. Sería más sencillo practicar intentando emular inteligencias mucho más simples, como los insectos, e ir avanzando en la escala evolutiva según se encontrasen resultados y se aumentasen los conocimientos sobre el pensamiento.

De este modo surgen las arquitecturas reactivas, que precisan de un tiempo de cálculo mucho menor que en el caso de los bucles PPA, incrementando en mucho la capacidad de reacción de los robots además de reducir la capacidad de cómputo necesaria, más acorde con la tecnología del momento.

Se denominan reactivas porque el robot no tiene ningún modelo del entorno, sino que reacciona ante los estímulos de este. La descomposición de las tareas se realiza en tareas más simples que se ejecutan en paralelo, buscando ciertas características del entorno y actuando cuando estas aparecen. De este modo las distintas subtareas compiten por el control de los actuadores del robot.

6.3. INTRODUCCIÓN A LA MICROBOTICA

Otro elemento importante que se debe introducir para comprender el desarrollo del proyecto es la Torrebot. Se trata de una serie de niveles cuyo cumplimiento por parte del robot lo sitúa en una escala de desarrollo. El concepto de la torrebot ha sido desarrollado por el grupo J&J , actualmente Microbótica S.L.[20] y se utiliza su propia definición de cada uno de los niveles:

- **Nivel Físico:** Comprende la estructura física, las unidades motoras, y las etapas de potencia. Es posible encontrar desde sistemas sumamente sencillos basados en un único motor, hasta estructuras sumamente complejas que buscan emular las capacidades mecánicas de algunos insectos.

- **Nivel de Reacción:** Está formado por el conjunto de sensores, así como los sistemas básicos para su manejo (circuitos de polarización). Estos transductores cubren un amplio margen de posibilidades, tal que podemos encontrar desde simples "bumpers", hasta micro cámaras digitales con sistemas de reconocimiento. Un microbot que haya superado en cuanto a su construcción tanto el nivel físico como el de reacción, se denomina microbot reactivo .Este tipo de unidades, trabajan cumpliendo la premisa, "acción-reacción". En estos casos, los sensores son los propios controladores de las unidades motoras, sin ningún tipo de control intermedio.

- **Nivel de Control:** Incluye los circuitos más básicos que relacionan las salidas de los sensores con las restantes unidades. Partiendo de una simple lógica digital hasta potentes microcontroladores, se busca dotar al microbot de la capacidad para procesar la información obtenida por los sensores, así como actuar de una manera controlada sobre las unidades motoras.

- **Nivel de Inteligencia:** Abarca el planificador a largo plazo; en este nivel, se introducen los objetivos del microbot que tienen relativa independencia de los sensores. Este es el más alto nivel de inteligencia que puede alcanzar un microbot como una unidad individual.

- **Nivel de Comunidad:** Se trata de la puesta en funcionamiento de más de un microbot dentro de un mismo entorno de forma simultánea y sin que ninguno de ellos tenga conocimientos explícitos de la existencia de otros en su mismo entorno. A estos recintos se los denomina granjas. Los centros de investigación utilizan las granjas como entornos de observación de los microbots. Dichos establecimientos pueden contar con sistemas sofisticados que permitan a un operario monitorizar el

comportamiento de la comunidad así como alterar las condiciones externas del sistema (agregar obstáculos, cambiar la temperatura, etc...).

- **Nivel de Cooperación:** Comprende los sistemas donde a partir de un nivel de comunidad, se planifican o programan los microbots para que tengan conocimiento de la existencia de otros, tal que posean la capacidad de cooperar para el buen desarrollo de una tarea .

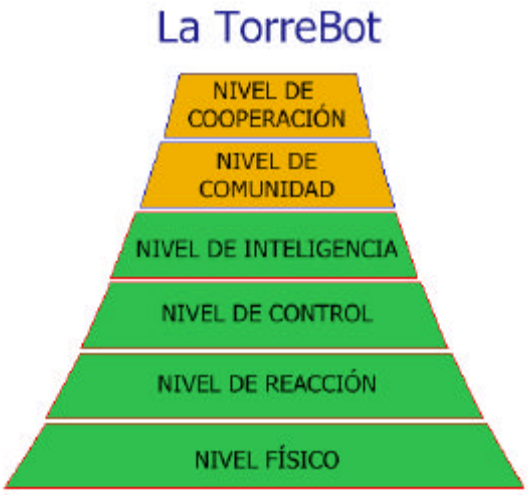


Figura 6. 1 : Torrebot

En este proyecto vamos construir un robot que va desde el nivel físico y llegue hasta el nivel de control. En este tema se abordará solo el nivel físico. El nivel de reacción se abordará en el capítulo siguiente. Finalmente el resto de capítulos se abordará el nivel de control.

6.4. NIVEL FÍSICO. RUEDAS

6.4.1. TIPOS DE RUEDAS

En la industria micro robótica podemos encontrar diferentes tipos de rueda para poder mover nuestro micro robot. A continuación se muestra los cuatro tipos de rueda mas comunes que se pueden encontrar en los micros robots:

- **Ruedas Fijas:**
  - Velocidad del punto P $V = (r \times w)a_x$:
  - El vector normal en la dirección del eje x es ax.
  - Restricciones: El punto P no puede moverse en la dirección perpendicular al plano de la rueda

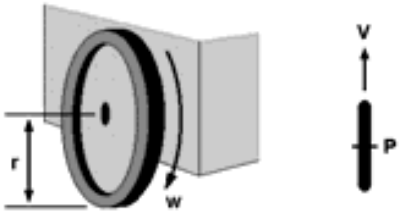


Figura 6. 2 : Rueda fija

- **Rueda orientable centrada:**
  - Velocidad del punto P: $V = (r \times w)a_x$
  - El vector normal en la dirección del eje x es ax.El vector normal en la dirección del eje y es ay
  - Restricciones: El punto P no puede moverse en la dirección perpendicular al plano de la rueda

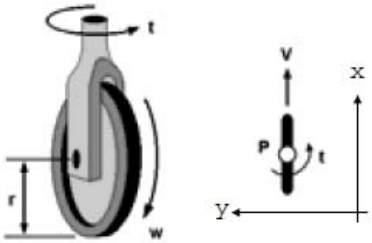


Figura 6. 3 : Rueda orientable centrada

- **Rueda orientable descentrada (Rueda loca)**
  - Velocidad del punto P $v = (r \times w)a_x + (d \times t)a_y$
  - El vector normal en la dirección del eje x es ax. El vector normal en la dirección del eje y es ay
  - Es omnidireccional

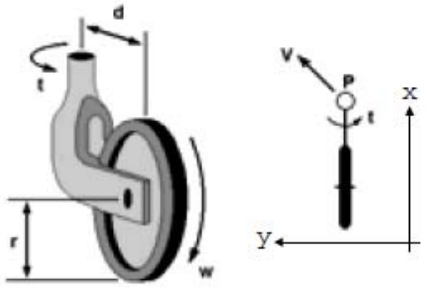


Figura 6. 4: Rueda orientable descentrada

- **Rueda sueca**
  - Velocidad del punto P $v = (r \times w)a_x + Ua_s$
  - El vector normal en la dirección del eje x es ax.El vector normal en la dirección del rodillo es as
  - Es omnidireccional

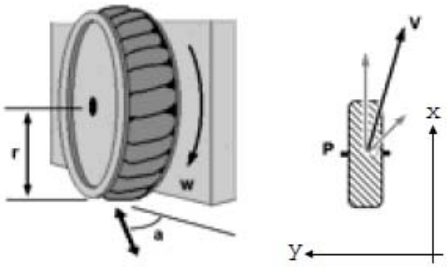


Figura 6. 5: Rueda sueca

6.4.2. CONFIGURACIONES DE MICROROBOTS SEGÚN LA DIPOSICIÓN DE LAS RUEDAS

Según la disposición de las ruedas, vistas en el subapartado anterior, obtendremos distintas configuraciones [21] de micro robots. Las configuraciones más usuales que se pueden encontrar son :

- **Configuración Diferencial**

Uno de los esquemas más sencillos .Consiste de dos ruedas en un eje común, donde cada rueda se controla independientemente. Utiliza una o dos ruedas adicionales (caster) para mantener el balance.

Movimientos:

- En forma recta
- En arco
- Vuelta sobre su propio eje

- **Configuración Tipo triciclo**

Posee dos ruedas fijas que le dan tracción. Una rueda para dirección que normalmente no tiene tracción .Presenta buena estabilidad y simplicidad mecánica. Tiene facilidad para ir recto pero la cinemática más compleja .

- **Configuración Tipo carro**

El similar al triciclo. Posee dos ruedas de tracción y dos ruedas para dirección. Presenta mayor complejidad mecánica que el triciclo por acoplamiento entre las 2 ruedas de dirección Tiene buena estabilidad y facilidad de ir derecho pero la cinemática es compleja .

6.4.3. CONFIGURACIÓN DEL MICRO ROBOT "MYBOT"

La configuración adoptada para nuestro micro robot experimental ha sido el **tipo triciclo** dado que presenta buena estabilidad simplicidad mecánica y permite un control más sencillo del sistema.

Puesto que se hará uso de la configuración triciclo la configuración de las ruedas serán las siguientes:

Rueda orientable descentrada (Rueda loca):

Compuesta por un acoplador a la estructura del bot, una barra, una rueda y tornillos para su fijación (ver Figura 6.6).

Figura 6. 6 : Rueda loca MYBOT

Ruedas de tracción

Para la tracción del micro robot hemos utilizado dos ruedas acopladas directamente al servomotor.

6.5. NIVEL FISICO. ESTRUCTURA

6.5.1. ESTRUCTURAS COMERCIALES

Unas de las estructuras más utilizadas es la de los juegos educacionales de construcción Lego,Mecano o Eitech. Para un diseño un poco más profesional se pueden utilizar las estructuras del Fischer Technik.

Un ejemplo de estructura realizada con Lego es el caso del Lego MINDSTORM NTX preparados para armar, posee una unidad de procesamiento, motores y diversos sensores.

Figura 6. 7: Lego MINDSTORM NTX

6.5.2. ESTRUCTURA DEL MICROROBOT "MYBOT"

La estructura de nuestro micro robot se pensó de forma que cumpliera los siguientes requisitos mínimos :

- Coste reducido
- Calidad adecuada
- Resistencia a golpes
- Peso reducido
- Capacidad para soportar peso (tarjetas ,sensores, baterías…)
- Material que permita operaciones secundarias(taladrado)

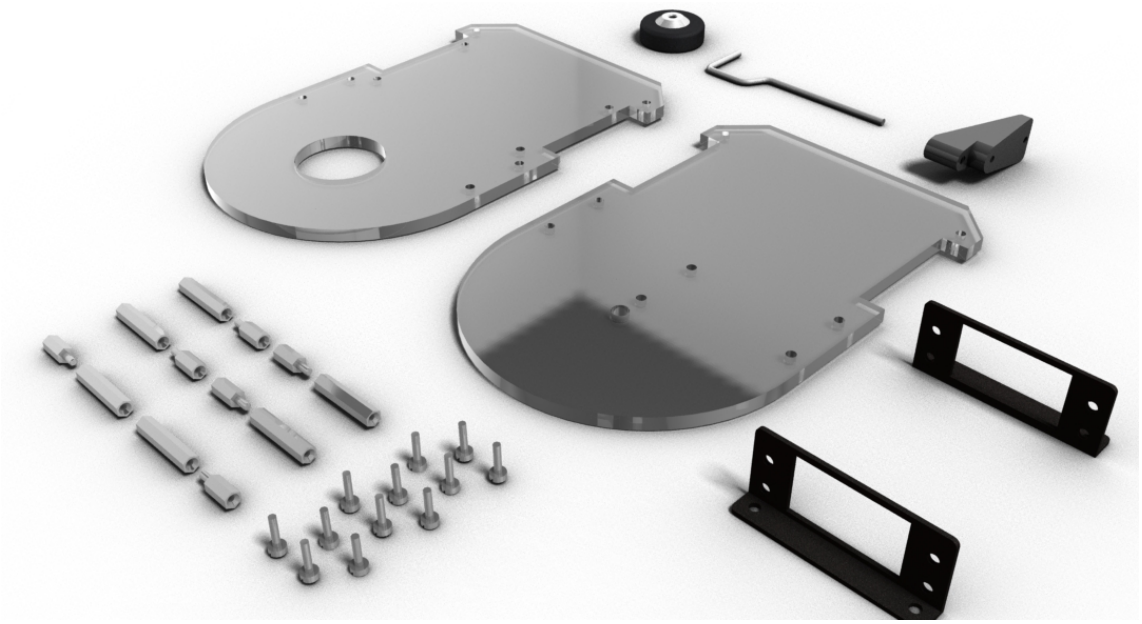


Figura 6. 8 : Despiece MYBOT

Por todo ello se procedió a diseñar y construir el micro robot "MYBOT" en metacrilato. Obteniendo una estructura de dos pisos:

- **Piso superior :** En donde se depositará la placa principal de control ,las baterías y se atornillará la sujeción de la rueda loca (Figura 6.9).

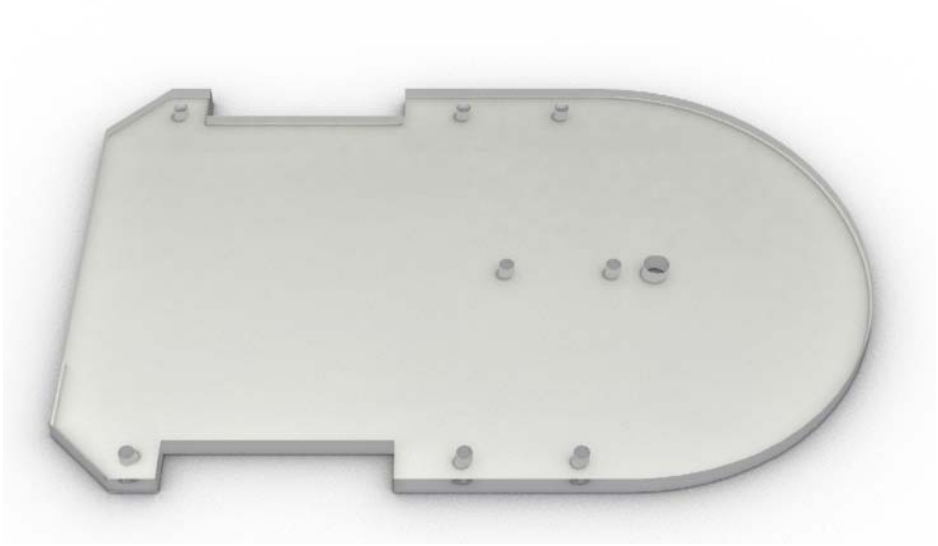


Figura 6. 9

- **Piso inferior:** En donde se depositará algunos sensores, placas secundarias y se atornillará las fijaciones de los servomotores al piso. Se caracteriza por un gran orificio en la zona curvada(Figura 6.10)

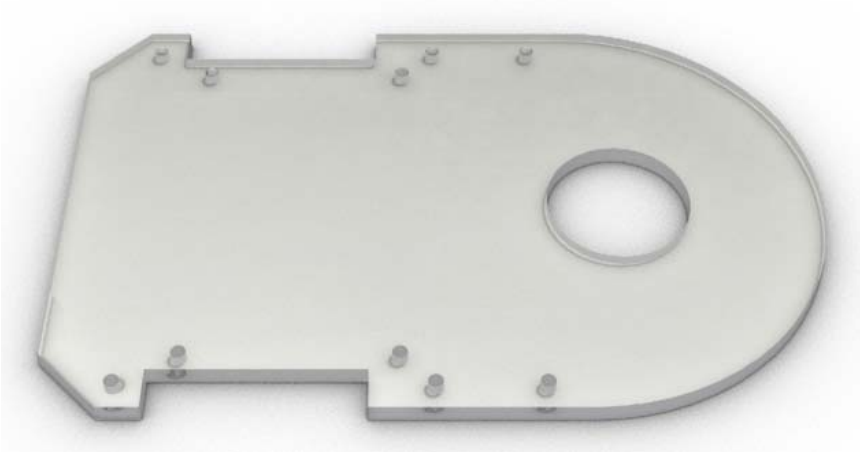


Figura 6. 10

Se hace uso de dos tipos de separadores distintos. El mas largo posee dos roscados hembra mientras que el más pequeño posee un roscado hembra y otro macho . Atornillados adecuadamente a cada una de las caras su propósito no es otro que dar firmeza a la estructura, ver Figura 6.11 para observar los separadores. En total se usan 6 separadores largos y otros 6 cortos.

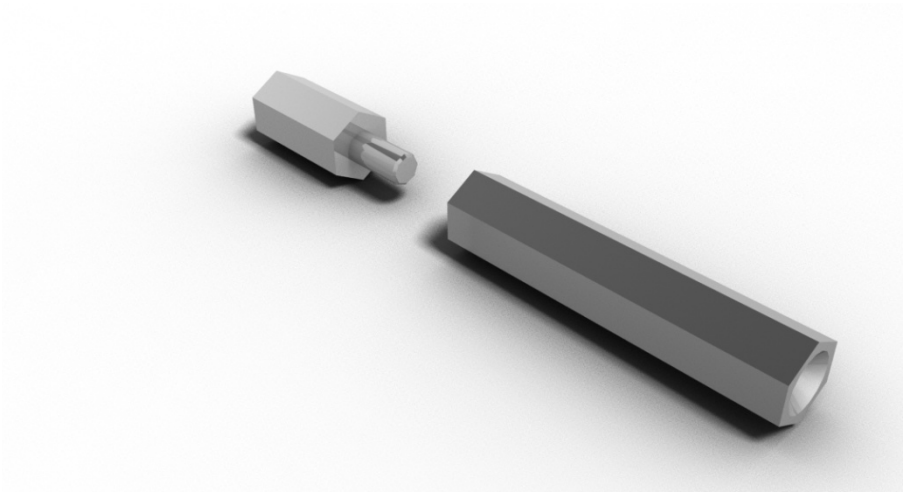


Figura 6. 11

En primer lugar atornillamos los respectivos separadores a cada una de las caras, con unos tornillos DIN84 2.5x10 (color ocre) . Como se puede apreciar en la Figura 6.12 insertaremos primero los tornillos en uno de los pisos seguidamente atornillaremos los separadores (colores rojo y azul) y cerraremos la estructura insertando los tornillos que faltan en el otro piso.

Toda la información referida al despiece del microrobot y las dimensiones de los pisos se encuentran en la sección de planos del presente proyecto final de carrera y en el CD adjunto.

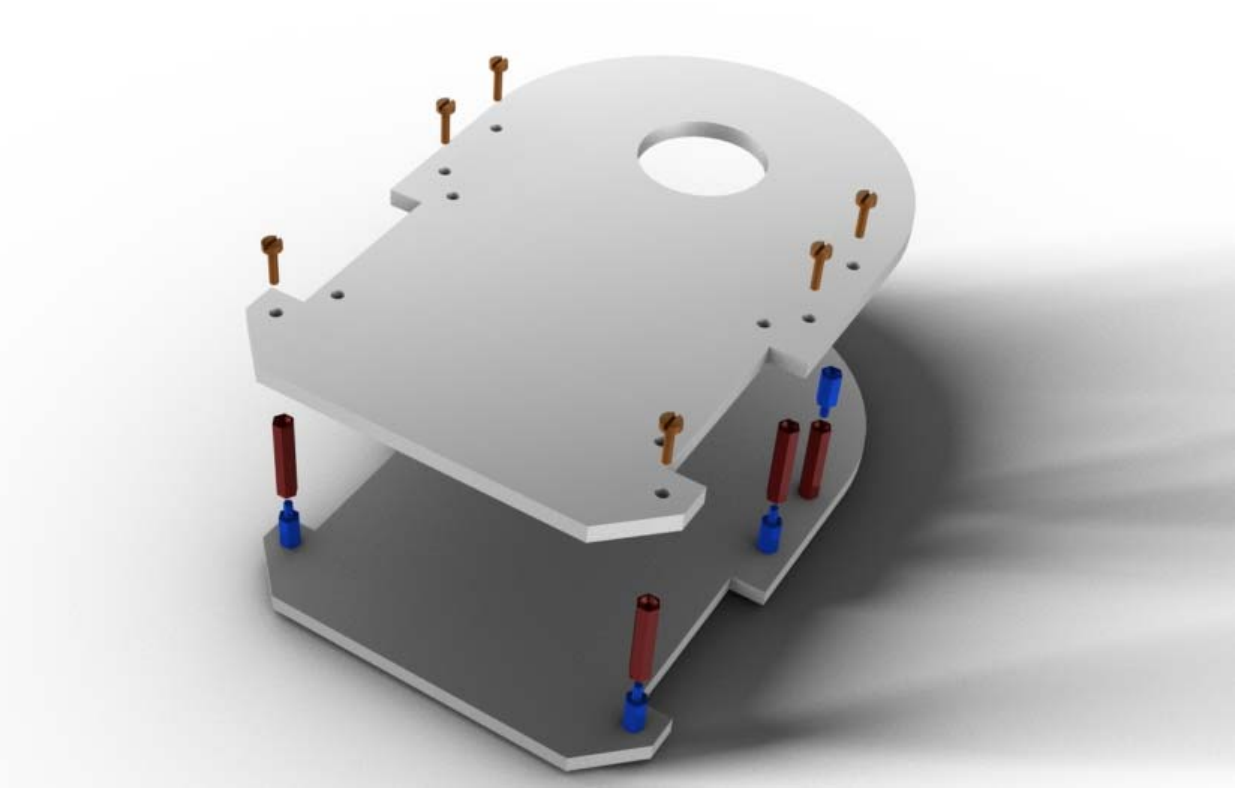


Figura 6. 12 : Estructura MYBOT en proceso de construcción

En la figura 6.13 podemos comprobar la estructura montada. Las Figuras 6.14 ,6.15 y 6.16 son posibles presentaciones que podría poseer la estructura.

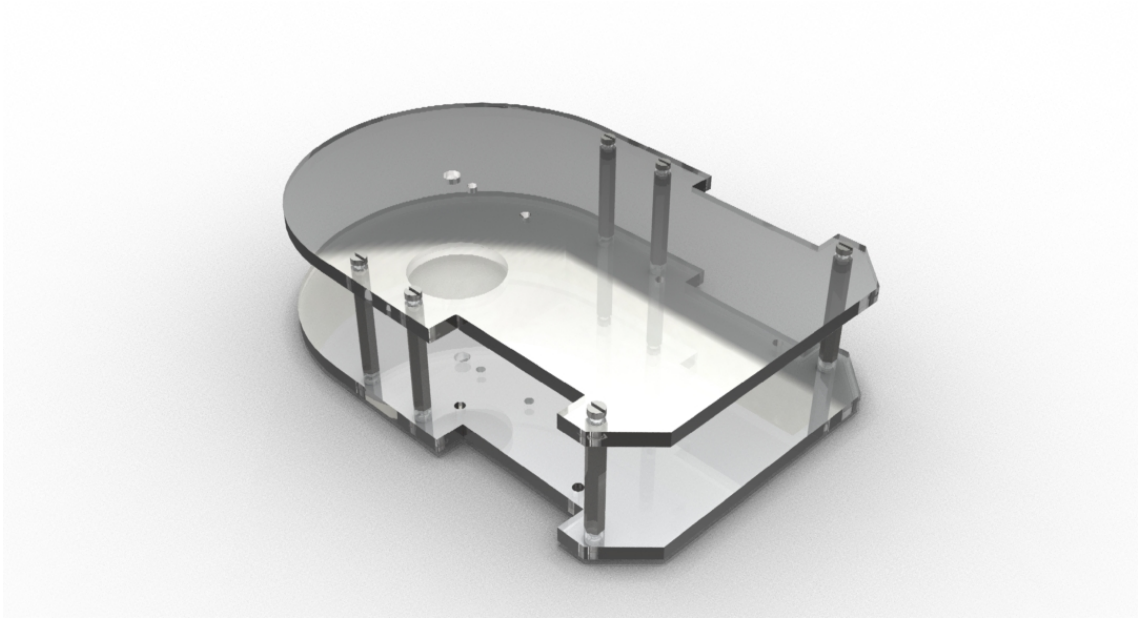


Figura 6. 13

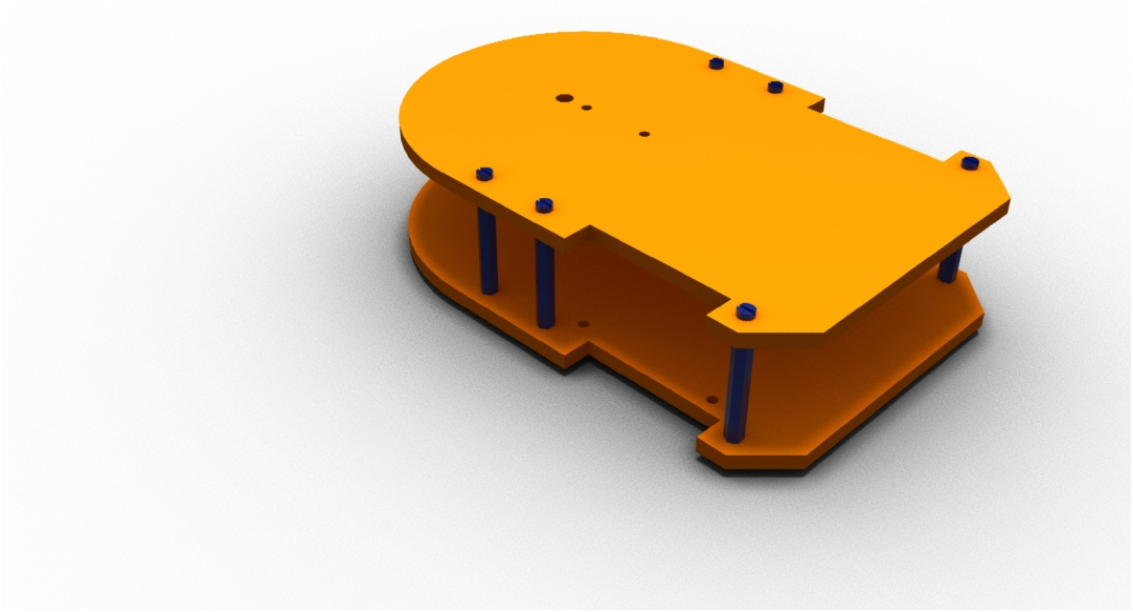


Figura 6. 14

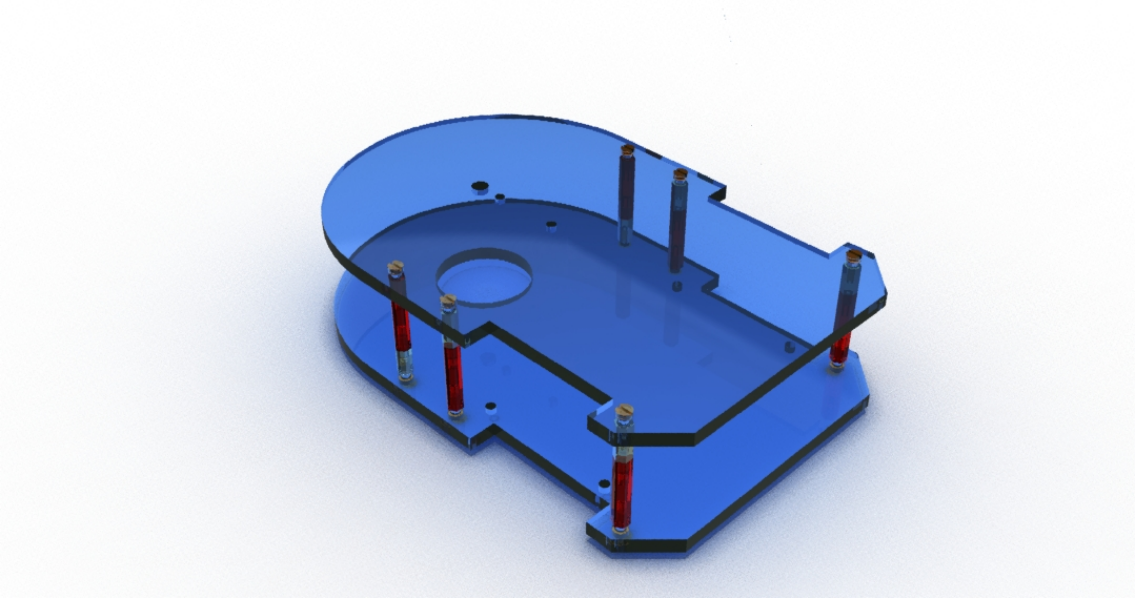


Figura 6. 15

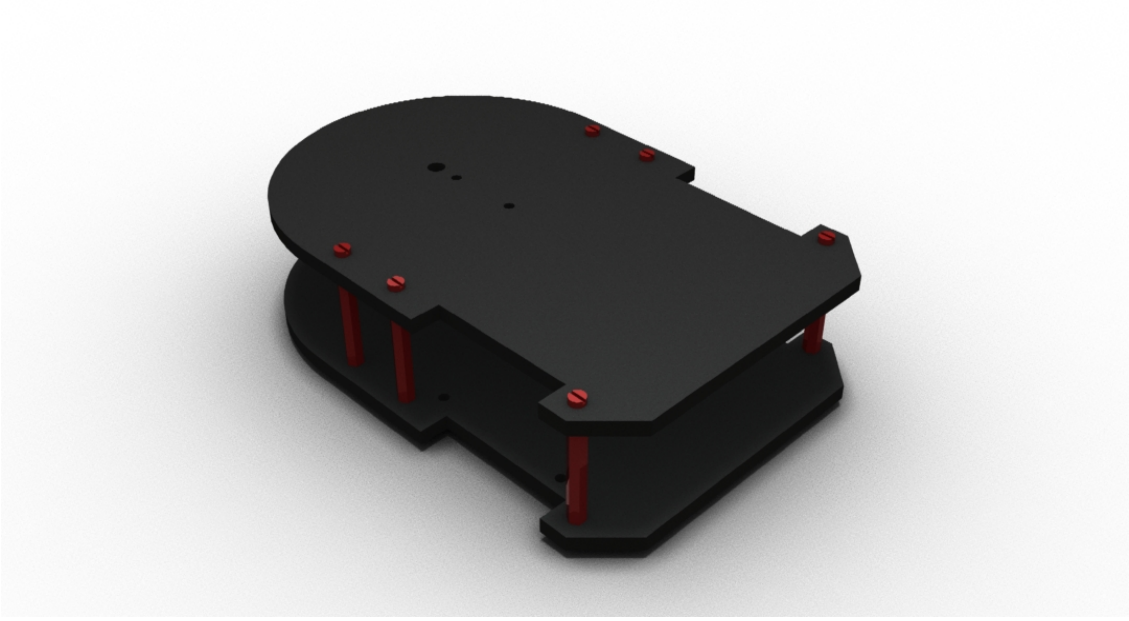


Figura 6. 16

6.6. NIVEL FISICO. MOTORES

Dentro del nivel físico nos encontramos con la sección de motores. A la hora de elegir un motor para aplicaciones de micro robótica, debemos tener en cuenta que existen varios factores como la velocidad, el par, el frenado, la inercia y el modo de control. Si lo que queremos es utilizar un motor de corriente continua existen varias posibilidades en el mercado.

6.6.1. MOTORES DC SIN REDUCTORA

Dentro de la gran variedad de motores existentes en el mercado los mas económicos son los que se utilizan en algunos juguetes. Tienen el inconveniente de que su número de revoluciones por minuto es muy elevado y su par es pequeño, por tanto no son muy apropiados para la construcción de un microbot si no se utilizan reductoras adicionales o un sistema de regulación electrónico

Figura 6. 17 :
Motor DC

6.6.2. MOTORES DC CON REDUCTORA

En la industria del aeromodelismo y modelismo en general podemos encontrar motores con reductoras o la posibilidad de construirlos. También podemos encontrar en el mercado motor con reductores, que además de disminuir la velocidad le dan más par, lo que permite mover el microbot con su estructura y batería sin problemas.

Figura 6. 18 : Motor DC con reductora

6.6.3. SERVOMOTORES

En micro robótica se suelen utilizar los mismos servomotores que en modelismo y radiocontrol. Se trata de unos motores con un circuito electrónico para su control.

Los servomotores cumplen características que los hacen idóneos para la construcción de nuestro microbot, como un buen par de salida, potencia suficientemente para trasladar objetos o una batería , baja inercia , capacidad de mover 2.8 kg x cm, incluyen multitud de accesorios para poder fijar las ruedas del microbot y son fáciles de fijar a una estructura plana al ir dentro de una carcasa de plástico rectangular con soportes para fijar los tornillos. En el capítulo 8 se detallara con profunidad el control del servomotor.

Figura 6. 19: Servomotor HITEC HSR-1422CR

Por todo lo mencionado se procede a usar servomotores de rotación continua con control de velocidad **HITEC HSR -1422CR en el micro robot "MYBOT".**

6.6.4. FIJACIÓN DELSERVOMOTOR A LA ESTRUCTURA

Para fijar los servomotores a la estructura, descrita en el apartado anterior, se usó unas escuadras con una serie de perforaciones en ambos lados de las caras para permitir fijar los motores a las escuadras , y estas a la estructura.(Figura 6.20)

Figura 6. 20

Finalmente la estructura con las fijaciones de los servomotores y la rueda loca se pueden observar en la Figura 6.21

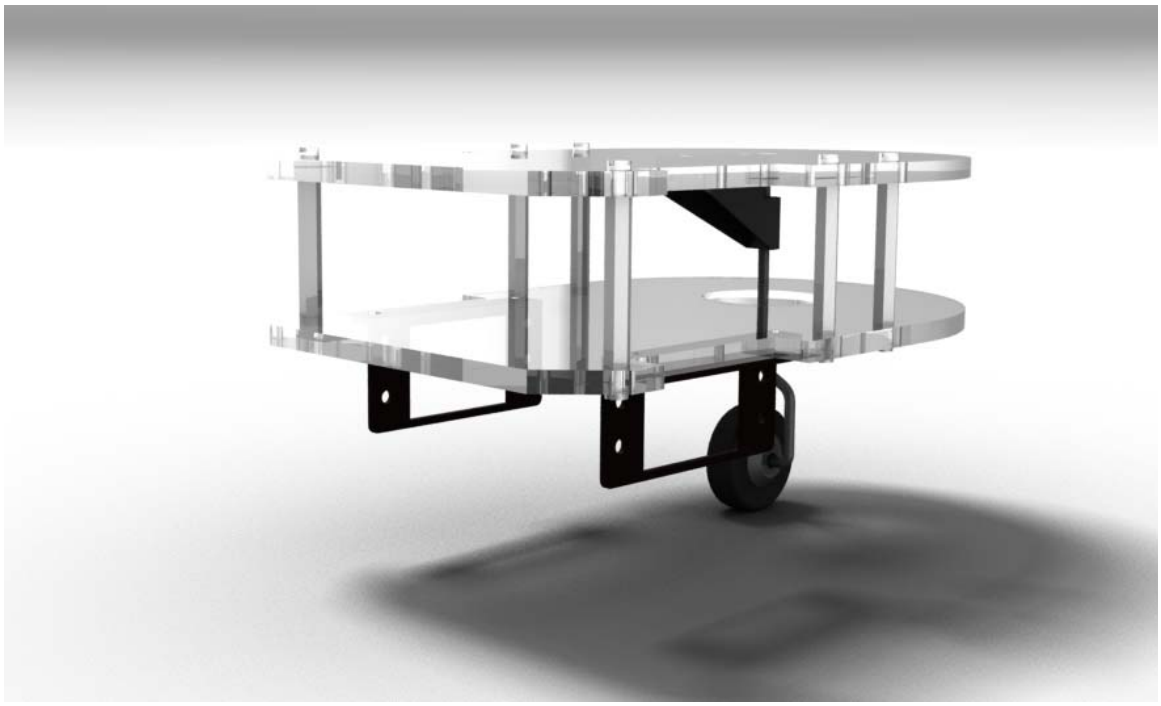


Figura 6. 21

6.7. NIVEL FISICO. CINEMATICA DEL MICRO ROBOT "MYBOT"

La configuración elegida nos permitirá realizar movimientos hacia delante, hacia atrás, giro a la derecha, a la izquierda y sobre sí mismo.

En la figura 6.22 se muestra como se realiza un movimiento hacia delante. Se hacen girar los dos motores en sentido opuesto al del otro, esto provoca un movimiento rectilíneo.

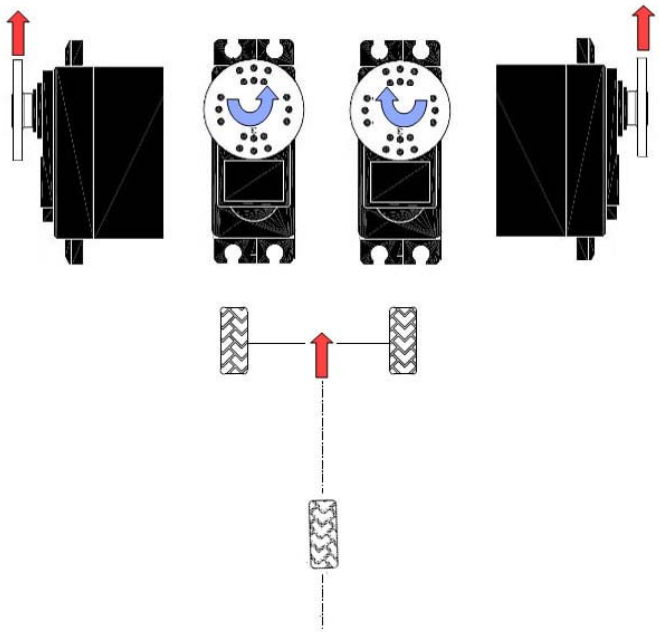


Figura 6. 22

En la figura 6.23 se muestra como se realiza un movimiento hacia atrás. Se hacen girar, nuevamente los dos motores en sentido opuesto al otro , esto provocara un movimiento rectilíneo.

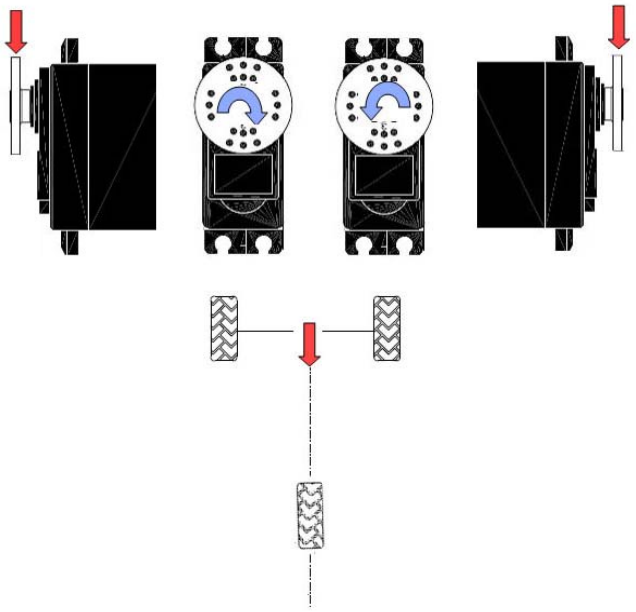


Figura 6. 23

Para poder realizar un movimiento de giro a la derecha, se hace girar el motor derecho mientras que el motor izquierdo permanece parado. En la figura 6.24 se muestra como realizar un movimiento de giro a la derecha.

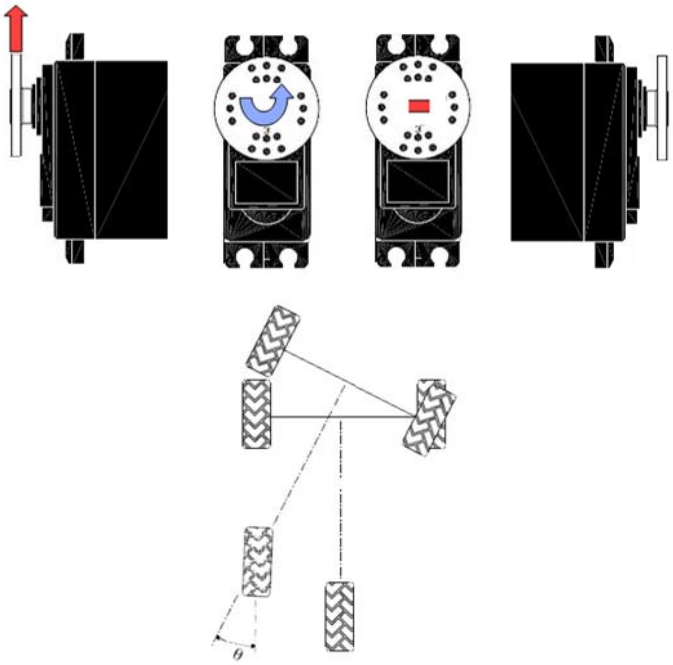


Figura 6. 24

Así mismo, Para poder realizar un movimiento de giro a la izquierda, se hace girar el motor izquierdo mientras que el motor derecho permanece parado. En la figura 6.25 se muestra como realizar un movimiento de giro a la izquierda

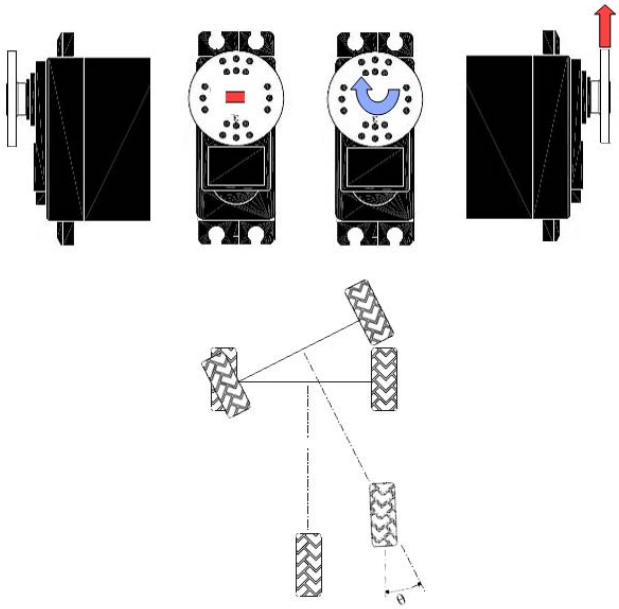


Figura 6. 25

La figura 6.26 indica cómo realizar un movimiento de giro completo sobre su propio eje . Evidentemente la estructura no la hace muy adecuada para realizar dicho movimiento en recintos muy pequeños.

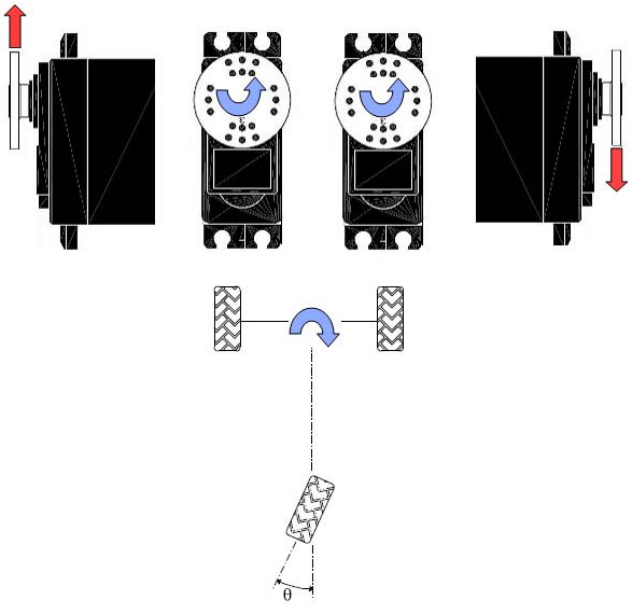


Figura 6. 26

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

CAPÍTULO 7 NIVEL DE REACCIÓN.SENSORES PARA MICRO ROBOTICA

CAPÍTULO 7. NIVEL DE REACCIÓN. SENSORES PARA MICRO ROBÓTICA

7.1. INTRODUCCIÓN

En el presente capítulo vamos a describir los diferentes sensores utilizados , placas construidas y cómo conectarlos al microcontrolador para posteriormente en el capítulo 9 realizar aplicaciones con el micro robot "MYBOT". Por tanto nos encontramos en el segundo nivel de la Torrebot, el **Nivel de Reacción**. (Figura 7.1)

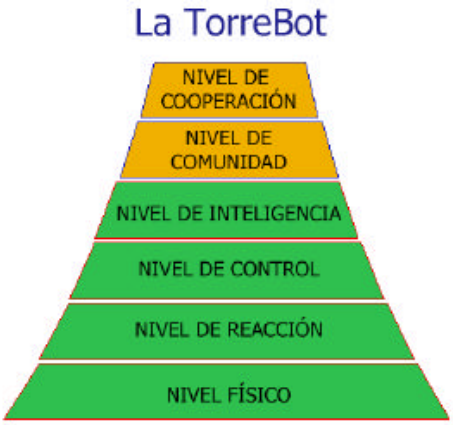


Figura 7. 1

Como se comentó en el tema anterior el Nivel de Reacción está formado por el conjunto de sensores, así como los sistemas básicos para su manejo (circuitos de polarización).Los sensores que se van a estudiar son:

- **LDR**, detector pasivo de luz. Se utiliza para detectar ausencia o presencia de luz
- **CNY70,** sensor óptico reflexivo por infrarrojo con salida a transistor. Diferencia colores blanco y negro.
- **GP2Y0A21YK,** sensor infrarrojo para medición de distancia.
- **ADLX330 ,** acelerómetro para la detección de movimiento en el espacio.

7.2. LDR

Las resistencias dependientes de la luz, LDR [22] (Light Dependet Resistor) o fotorresistencias, son dispositivos que varían su resistencia en función de la luz que incide sobre su superficie. Cuanto mayor sea la intensidad luminosa que incide sobre ella menor será la resistencia entre extremos de la misma. Para su fabricación se utilizan materiales fotosensibles. Su aspecto físico y simbología más común se muestran en las figuras 7.2 y 7.3

Figura 7. 2

Figura 7. 3

Un foto resistor está hecho de un semiconductor de alta resistencia como el sulfuro de cadmio, CdS. Si la luz que incide en el dispositivo es de alta frecuencia, los fotones son absorbidos por la elasticidad del semiconductor dando a los electrones la suficiente energía para saltar la banda de conducción. El electrón libre que resulta, y su hueco asociado, conducen la electricidad, de tal modo que disminuye la resistencia. Los valores típicos varían entre 1 MΩ, o más, en la oscuridad y 100 Ω con luz brillante.

La variación del valor de la resistencia tiene cierto retardo, diferente si se pasa de oscuro a iluminado o de iluminado a oscuro. Esto limita a no usar los LDR en aplicaciones en las que la señal luminosa varía con rapidez. El tiempo de respuesta típico de un LDR está en el orden de una décima de segundo. Esta lentitud da ventaja en algunas aplicaciones, ya que se filtran variaciones rápidas de iluminación que podrían hacer inestable un sensor (ej. tubo fluorescente alimentado por corriente alterna). En otras aplicaciones (saber si es de día o es de noche) la lentitud de la detección no es importante.

Utilizaremos el siguiente montaje:

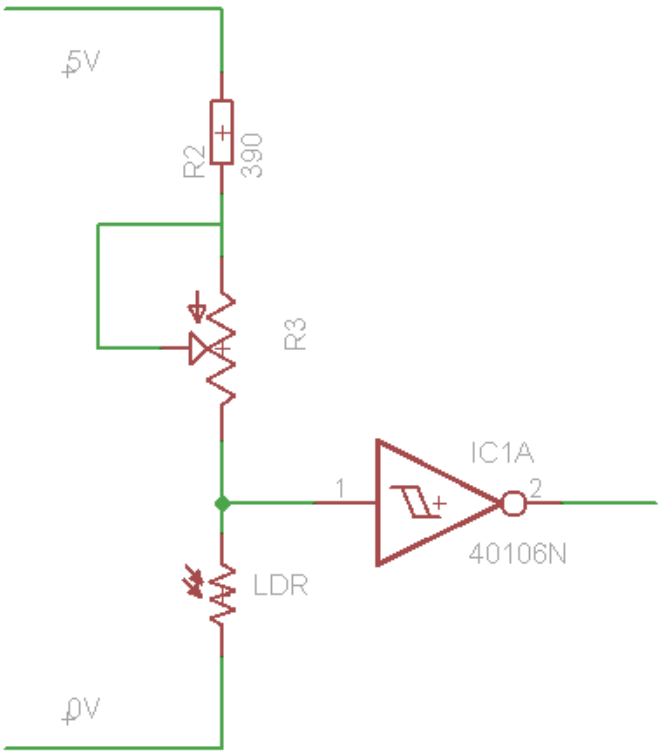


Figura 7. 4: Divisor de tensión para el LDR

Como podemos observar es un divisor de tensión cuya salida se pasa por un Trigger Smith para convertir la señal en digital. Teniendo en cuenta el esquema:

$$Vout1 = \frac{Rldr}{Rldr + R3 + R2} * Vin$$

Siendo Vin =5V tenemos que:

LUZ	LDR	Vout1	Vout2
SI	Ej: 100 omh	Aprox 0V	5V
No	Ej: 100.000 omh	Aprox 5V	0V

Tabla 1

Para introducir a un microcontrolador la señal del LDR es conveniente hacer pasar las salidas a través de un circuito trigger schmitt como se puede observar en la figura anterior. Para más información sobre el Trigger Schmitt ver Anexo B.

7.2.1. CIRCUITO DESARROLLADO PARA EL MICRO ROBOT "MY BOT"

La utilidad de este sensor será la que el programador desee. En el capítulo 9 se detallará dos códigos ejemplos para su uso. En este subapartado describiremos el circuito utilizado en ese capítulo.

Los elementos que vamos a utilizar van a ser:

- 1 LDR
- 1 resistencias de 390 Omh
- 1 potenciómetro 4K
- 1 Trigger schmitt
- 1 Zócalo para el Trigger
- 1 Protoboard

El objetivo es diseñar una placa que tenga como integrado los componentes descritos obteniendo la señal del LDR en digital. También habrá que diseñar una entrada para suministrar corriente.

Mediante el uso del software de desarrollo de placas EAGLE[23] se realizó el esquema de la figura 7.5. A continuación se muestra una tabla (tabla2) con los elementos que componen el esquema mencionado.

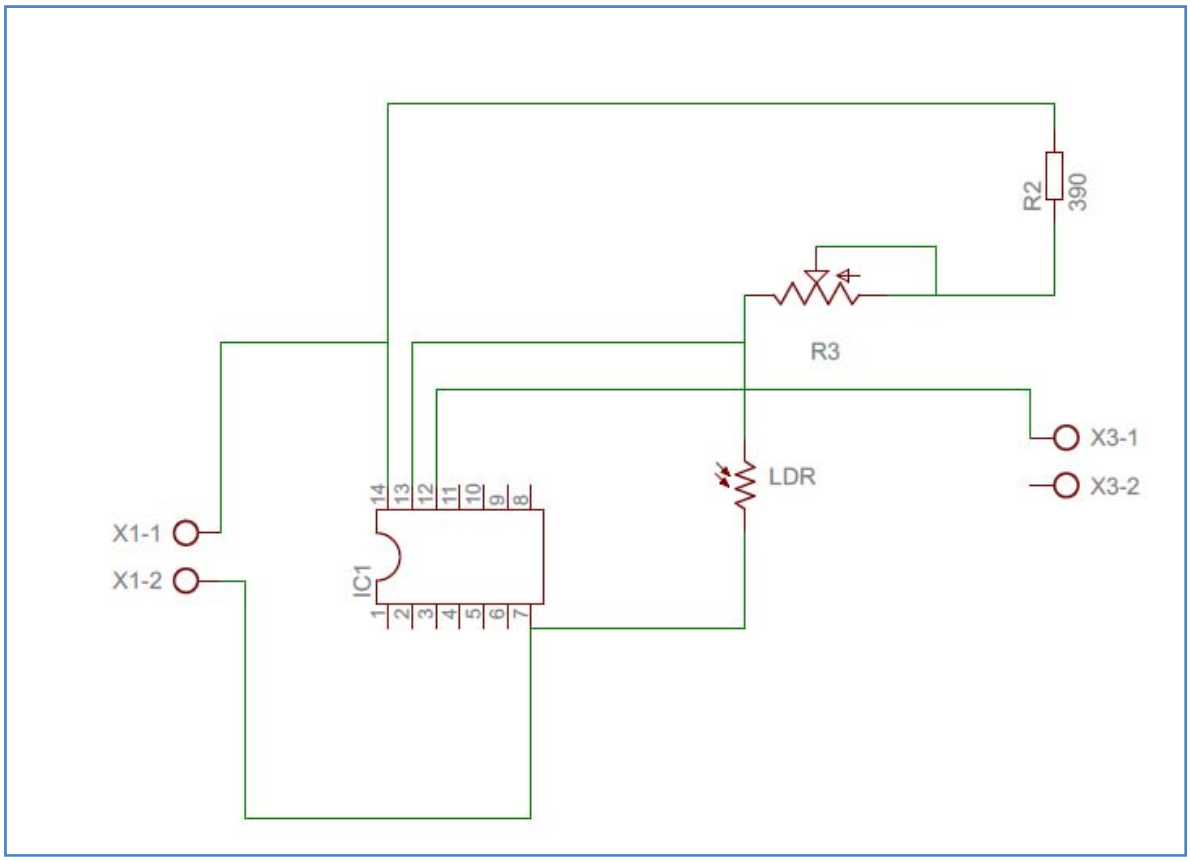


Figura 7. 5 : Esquema placa LDR

SIMBOLO	VALOR
X1-1	V+
X1-2	V-
X3-1	Salida en digital del LDR
R3	Potenciometro 4K
IC-1	Zócalo para el Trigger

Tabla 2

Una vez realizado el esquema y teniendo en cuenta todos los elementos que lo componen. Se procedió al diseño de las pistas como también a la disposición de los elementos dentro de un área determinada .

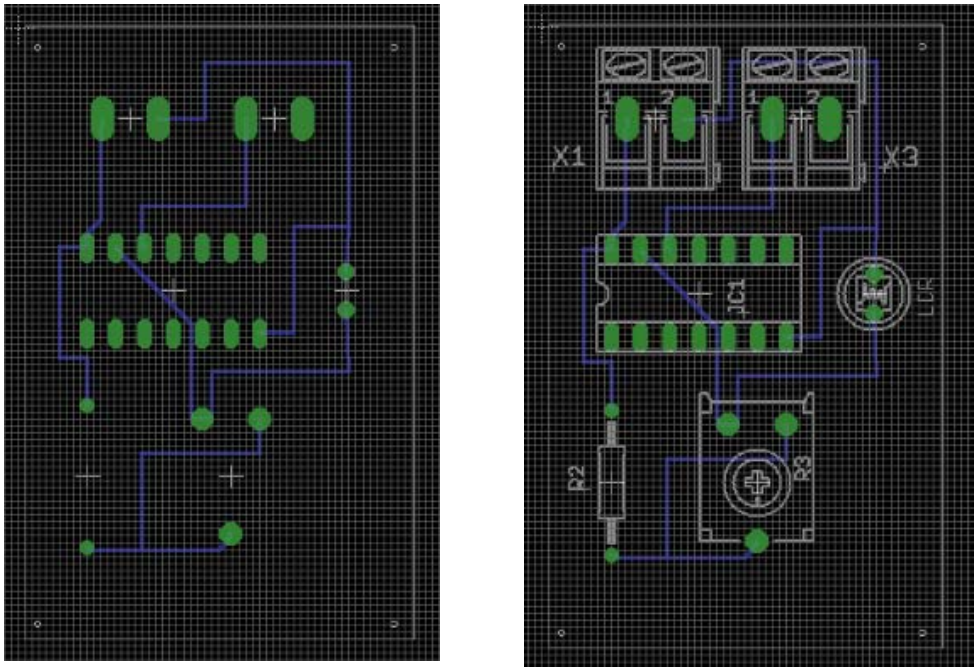


Figura 7. 6 : Diseño de las pistas

La figura 7.6 de la izquierda muestra las pistas sin los elementos y la de la derecha se muestra con los elementos. Si observamos fijamente la figura de la derecha veremos que para realizar las conexiones de entrada y salida de la placa se usan los mismos bornes que usa la placa de control desechada MSE-F87X explicada en el capítulo 4

El prototipo de esta placa se realizó de manera sencilla y barata. En la figura 7.7 se muestra el prototipo. Se adjunta la serigrafía de la placa desarrollada en el CD del presente proyecto final de carrera.

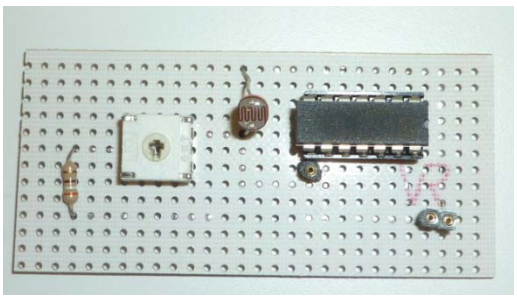


Figura 7. 7

7.3. CNY70

El CNY70 [24] es un sensor óptico reflexivo que tiene una construcción compacta dónde el emisor de luz y el receptor se colocan en la misma dirección para detectar la presencia de un objeto utilizando la reflexión del infrarrojo sobre el objeto. La longitud de onda de trabajo es 950nm. El detector consiste en un fototransistor.

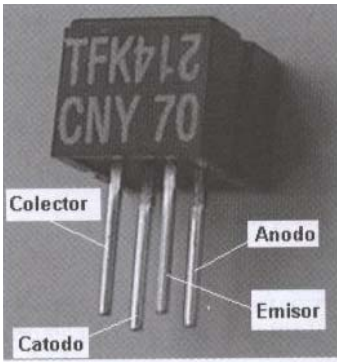


Figura 7. 8

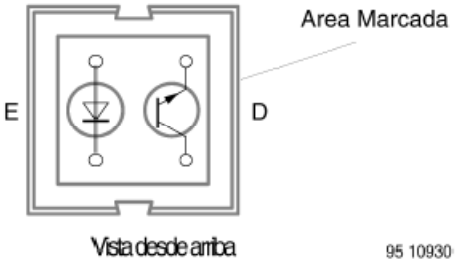


Figura 7. 9

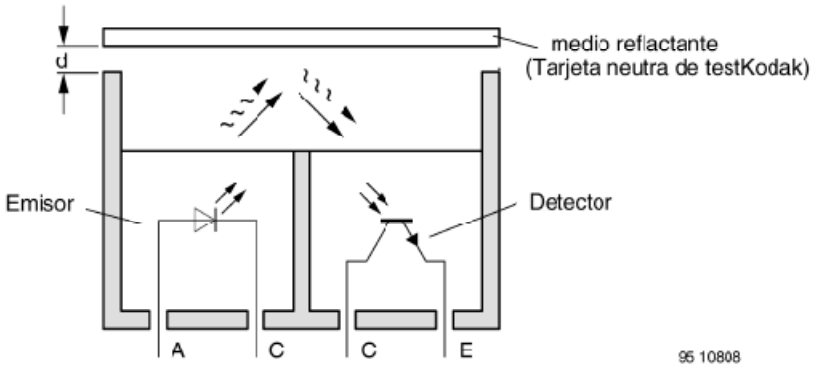


Figura 7. 10

Como se observa en la figura 7.8 el CNY70 tiene cuatro pines de conexión que se corresponden con el emisor, colector del transistor y al ánodo y cátodo del diodo emisor, en la figura de las vistas donde se indica “Área Marcada”, se muestra la inscripción con letras blancas del fabricante.

Se pueden utilizar cualquiera de los siguientes montajes de la figura 7.11. Permiten obtener a la salida un nivel alto o un nivel bajo respectivamente cuando están activados por la reflexión del haz infrarrojo.

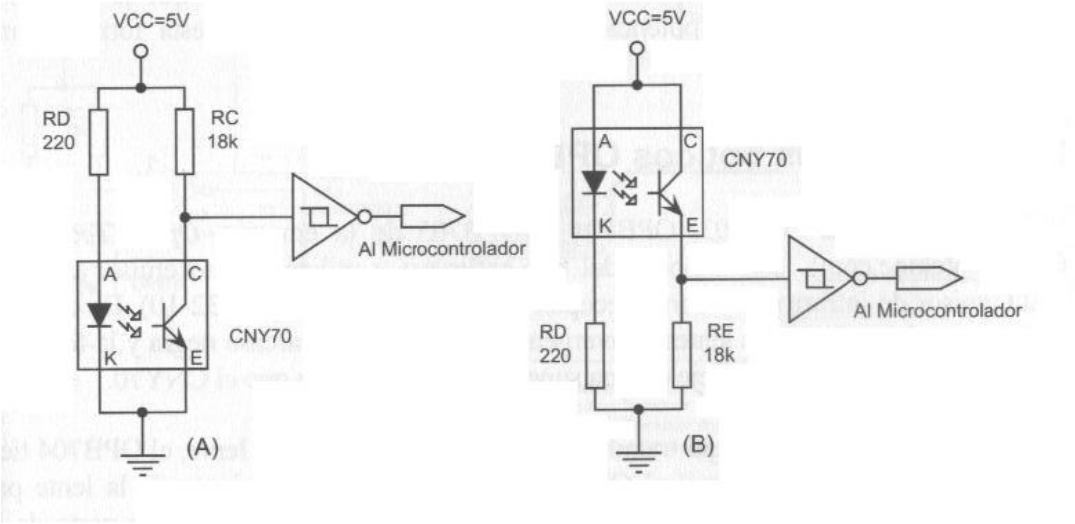


Figura 7. 11

El circuito (a) entrega a la salida un nivel bajo cuando no refleja el haz infrarrojo y un nivel alto cuando encuentra un material sobre el que refleja el haz.

El circuito (b) entrega un nivel alto cuando el haz no refleja y un nivel bajo cuando se detecta un material reflectante.

Si la señal se quiere introducir a un microcontrolador es conveniente hacer pasar las salidas a través de un circuito trigger schmitt que conforme las señales. (anexo B).

7.3.1. CIRCUITO CNY70 DESARROLLADO PARA EL MICRO ROBOT “MY BOT”

La utilidad de este sensor será el seguimiento de una línea con el micro robot. En el capítulo 9 se detallará el procedimiento para realizarlo . En este subapartado describiremos el circuito utilizado en ese capítulo .

Vistos los esquemas de la figura 7.11 se eligió el circuito (a). Como se ha dicho en el apartado anterior entrega a la salida un nivel bajo cuando no refleja el haz infrarrojo y un nivel alto cuando encuentra un material sobre el que refleja el haz.

Los elementos que vamos a utilizar van a ser:

- 2 sensores CNY70
- 2 resistencias de 220 Omh
- 2 resistencias de 19 Komh
- 1 Trigger schmitt
- 1 Zócalo para el Trigger
- 1 Protoboard
- Conectores PTR

El objetivo es diseñar una placa que tenga como entradas las salidas de los sensores CNY70 en analógica y la tensión a suministrar. Y como salidas de la placa la señal de los sensores en formato digital

Mediante el uso del software de desarrollo de placas EAGLE se realizó el esquema de la figura 7.12 . A continuación se muestra una tabla con los elementos que componen el esquema mencionado.

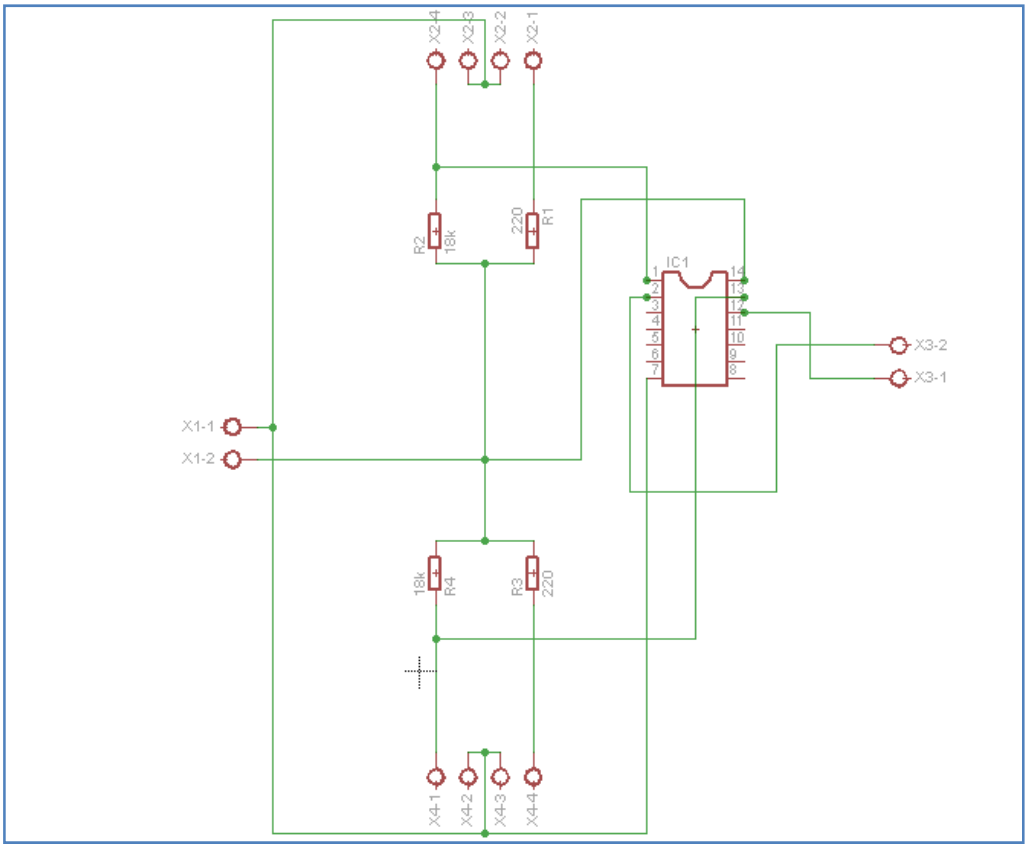


Figura 7. 12

SIMBOLO	VALOR
X1-1	V-
X1-2	V+
X2-1	Anodo del sensor 1
X2-2	Cátodo del sensor 1
X2-3	Emisor del sensor 1
X2-4	Colector del sensor 1
X3-1	Salida en digital del sensor 2
X3-2	Salida en digital del sensor 1
X4-1	Colector del sensor 2
X4-2	Emisor del sensor 2
X4-3	Cátodo del sensor 2
X4-4	Anodo del sensor 2
IC-1	Zócalo para el Trigger

Tabla 3

Una vez realizado el esquema y teniendo en cuenta todos los elementos que lo componen. Se procedió al diseño de las pistas como también a la disposición de los elementos dentro de un área determinada .

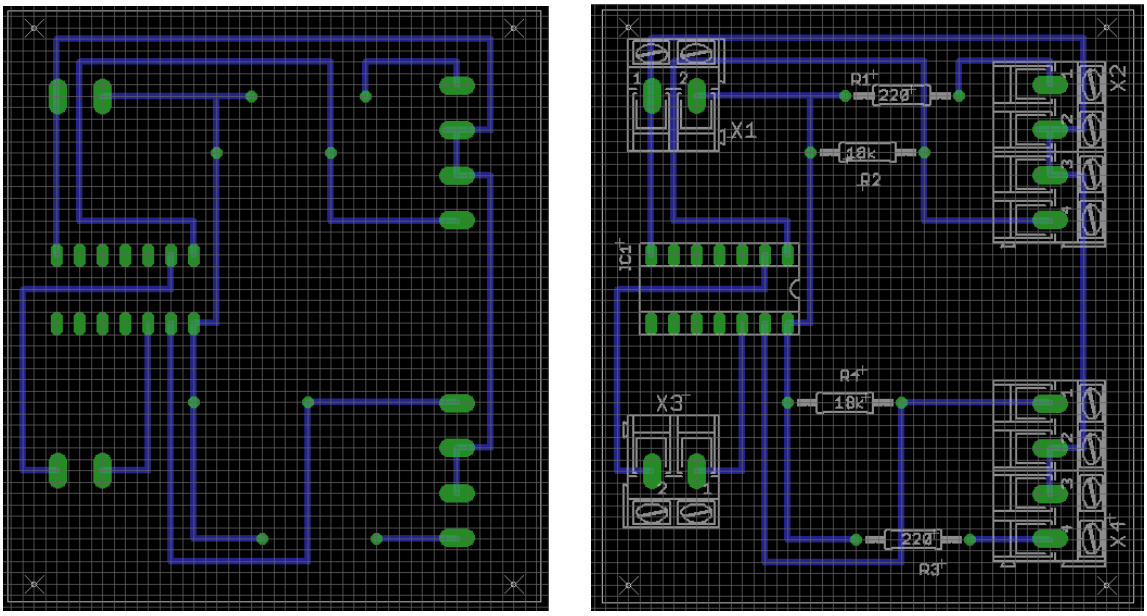


Figura 7. 13 : Placa desarrollada

La figura 7.13 de la izquierda muestra las pistas sin los elementos y la de la derecha se muestra con los elementos. Si observamos fijamente la figura de la derecha veremos que para realizar las conexiones de entrada y salida de la placa se usan los mismos bornes (conectores PTR) que usa la placa de control desechada MSE-F87X explicada en el capítulo 4 .Se muestra la numeración explicada en la tabla anterior para el conexionado de la misma.

El prototipo de esta placa se realizó de manera sencilla y barata. En la figura7.14 se muestra el prototipo. Se adjunta la serigrafía de la placa desarrollada en el CD del presente proyecto final de carrera.

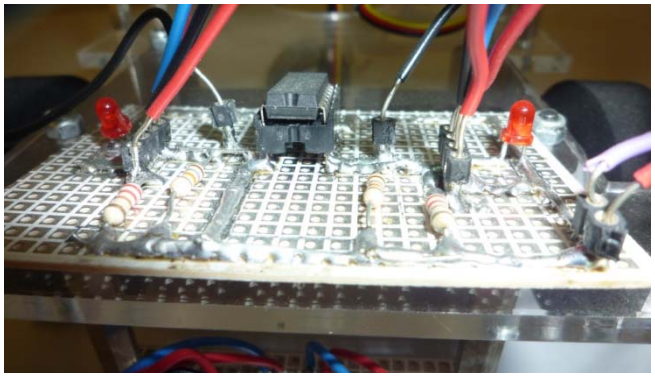


Figura 7. 14

7.4. GP2Y0A21YK

El Sharp GP2Y0A21YK [25] es un sensor que mide distancias por infrarrojos. El dispositivo indica mediante una salida analógica la distancia medida al objeto sobre el que refleja el haz de luz.

La tensión de salida varía de forma no lineal cuando se detecta un objeto en una distancia entre 10 y 80 cm , tal y como se aprecia en la curva de la figura 7.15.

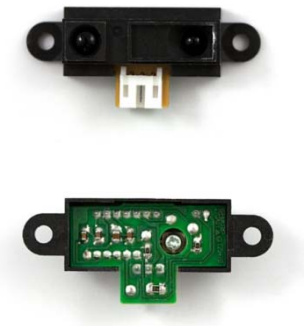


Figura 7. 16:

Sensor GP2Y0A21YK

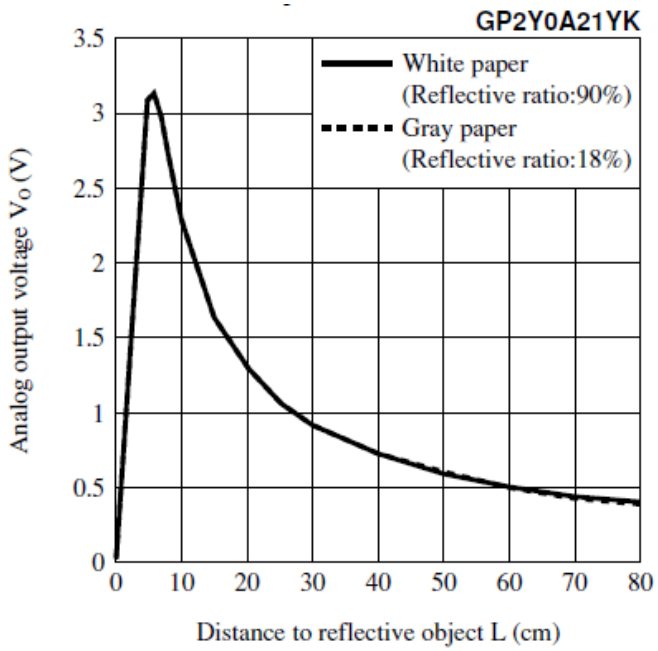


Figura 7. 15

El sensor posee un Terminal Japonés sin soldadura (JST) .La salida de este sensor se conectará a la entrada del microcontrolador que hará la función de convertidor analógico digital, el cual convertirá el numeró en valor binario . Por tanto tenemos que asegurarnos de conectarlo en el puerto correspondiente que realice la tarea de conversión.(ver anexo A)

7.5. ADLX330

El sensor ADXL330 [26] (Figura7.17) es un acelerómetro que mide la aceleración en los tres ejes, a través de un circuito integrado (CI monolítico, es un tipo de dispositivo electrónico en un circuito integrado el cual contiene dispositivos pasivos y activos como diodos, transistores, etc, los cuales están montados sobre una superficie de una pieza de un solo semiconductor, como una oblea de Silicio) .

Figura 7. 17

Para evitar posibles problemas derivados de la electrónica se hace uso de una placa montada para el uso directo. Dicha placa la proporciona la compañía Mikro Elektronika y se denomina AccelBoard [27.]

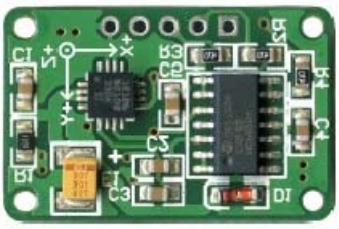


Figura 7. 18: **AccelBoard**

El diagrama (figura 7.19) que presenta la placa muestra el uso de amplificadores operacionales para amplificar la señal de salida del sensor ADXL330

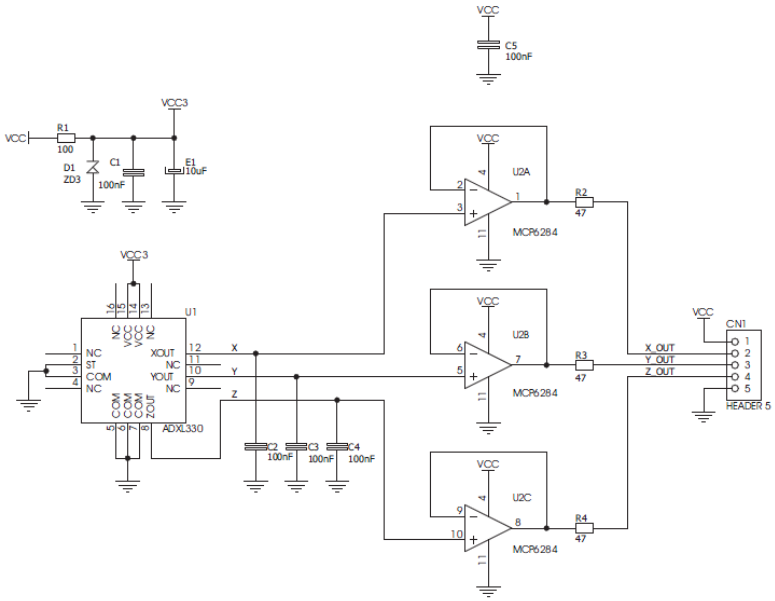


Figura 7. 19

Las señales de salida del sensor son voltajes analógicos los cuales son proporcionales a la aceleración, siendo el rango de este de ±3g. El acelerómetro puede medir la aceleración de la gravedad estática en las aplicaciones de detección de inclinación, así como la aceleración dinámica resultante de movimiento, choque o vibraciones. Además su conexionado es muy sencillo, sólo necesita 5 líneas: alimentación, tierra y 3 líneas de entrada hacia el microcontrolador (Ver figura 7.20)

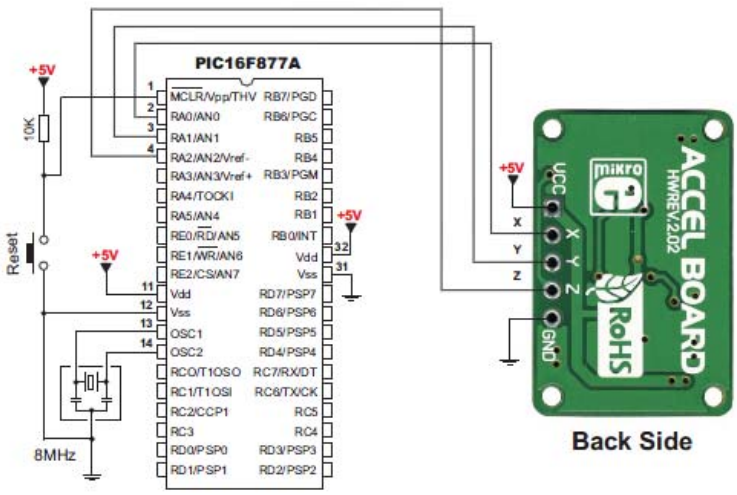


Figura 7. 20

CAPÍTULO 8 SOFTWARE DESARROLLADO PARA MOTORES HITEC

CAPITULO 8. SOFTWARE DESARROLLADO PARA MOTORES HITEC

8.1. INTRODUCCIÓN

El presente capítulo tiene como finalidad describir los métodos utilizados para el control de velocidad y sentido del giro de los servomotores HITEC HSR1422CR, y las aplicaciones desarrolladas para el microcontrolador de 8 bits PIC16F877A y el microcontrolador PIC32XXXX.

El objetivo final es la creación de una librería que permita el control de la velocidad , el sentido y de la dirección del micro robot de forma muy simple.

Este capítulo se desarrolla en dos plataformas hardware diferentes :

- EasyPIC4
- Placa Protoboard Olimex

El capítulo se centrará la mayor parte en el uso del microcontrolador de 8 bits PIC16F877A para controlar del micro robot.

En el apartado 8.6 se describirá el motivo por el cual se decidió también programar un microcontrolador de 32 bits. Indicando las ventajas que posee dicho microcontrolador con respecto el PIC16F877A .

8.2. FUNDAMENTOS DE CONTROL DE VELOCIDAD Y SENTIDO DEL GIRO DE SERVOMOTORES HITEC HSR1422CR

8.2.1. Fundamentos PWM

Puesto que no se encontró ningún tipo de driver o programa desarrollado para dicho servomotor se desarrolló todo el software, incluido las librerías. Para un mayor entendimiento del procedimiento de construcción de la librería se empezará explicando cómo variar de velocidad y sentido del giro del

servomotor utilizando la técnica de PWM (Pulse Width Modulation), es decir modulación por ancho de pulsos.

La modulación por ancho de pulsos (PWM,) es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (en nuestro caso cuadrada, ver figura 8.1)

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación al período. Matemáticamente:

$$D = \frac{\tau}{\mathrm{T}}$$

- D es el ciclo de trabajo
- τ es el tiempo en que la función es positiva (ancho del pulso)
- T es el período de la función

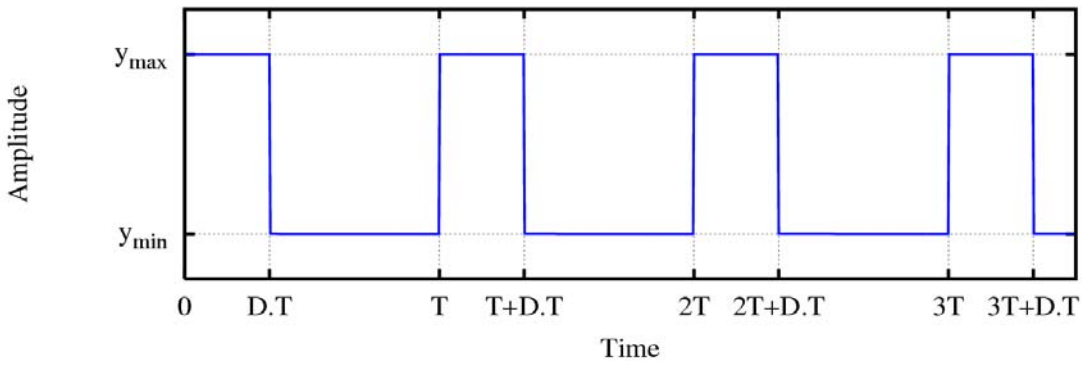


Figura 8. 1 : Función PWM

La construcción típica de un circuito PWM se lleva a cabo mediante un comparador con dos entradas y una salida. Una de las entradas se conecta a un oscilador de onda triangular, mientras que la otra queda disponible para la señal moduladora. En la salida la frecuencia es generalmente igual a la de la señal triangular y el ciclo de trabajo esta en función de la portadora.

La principal desventaja que presentan los circuitos PWM es la posibilidad de que haya interferencias generadas por radiofrecuencia. Estas pueden minimizarse ubicando el controlador cerca de la carga y realizando un filtrado de la fuente de alimentación.

Por tanto para evitar construir un circuito se procede a desarrollar una señal PWM mediante el PIC16F877A utilizando interrupciones del TIMER0, sumando como ventaja principal de tener todo integrado en un sistema.

8.2.2. Control de velocidad y sentido utilizando la técnica del PWM

Como se citó en el capítulo 6 apartado 6.6 se llego a la conclusión de hacer uso del servomotor HITEC HSR 1422CR [28] para mover a nuestro micro robot "MyBot".

El servomotor HITEC HSR 1422 CR es un servomotor de rotación continua pensado para utilizarse como sistema de locomoción de micro robots. Consta de 3 cables tal y como se muestra la figura 8.2.

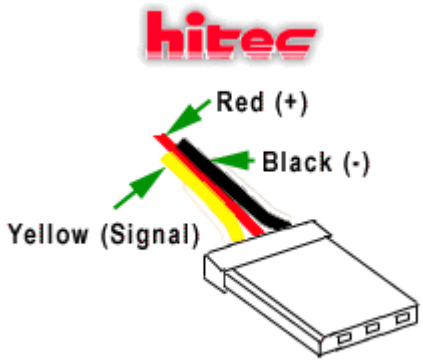


Figura 8. 2: Cables servomotor Hitec

La señal de control se envía por el cable amarillo mientras que los otros dos, rojo y negro, son para alimentación.

Según la hoja técnica , adjunta en el CD del presente proyecto fin de carrera ,el control de velocidad se realiza mediante PWM teniendo en cuenta que si se mandan pulsos con un $\tau = 1.5ms$ el servomotor está parado. Así pues tras realizar diversos métodos iterativos cambiando , el ancho de pulso , se llegó a la conclusión de que el servomotor posee unos márgenes de operación, que se corresponden con el ancho de pulso máximo y mínimo y que, en principio, no puede sobrepasarse.

Estos valores varían dependiendo del modelo de servomotor utilizado. Para el servomotor HITEC HSR1422CR los valores de la señal al nivel alto están entre 1.9 y 1.1 ms, y los resultados de elegir un valor ancho de pulso u otro se muestra en la siguiente figura:

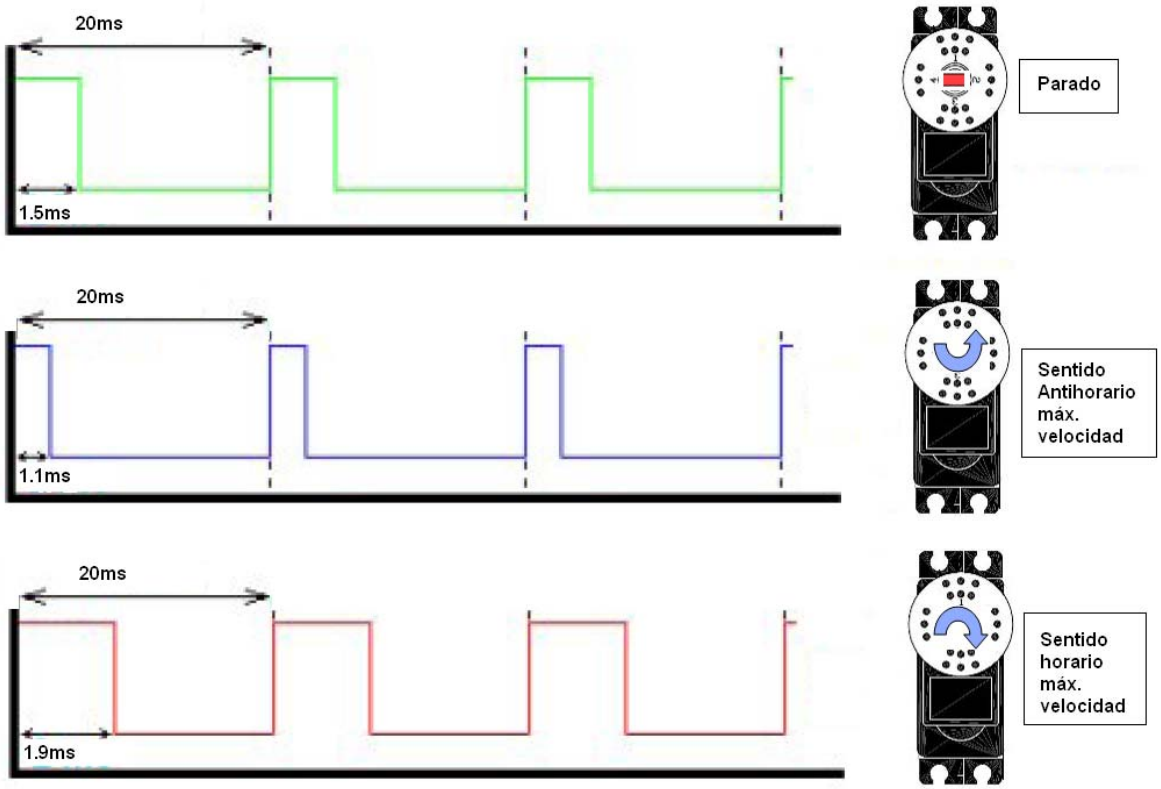


Figura 8. 3 : Diferentes ancho de pulso

A partir de estos resultados se dedujo la siguiente tabla 8.1 de velocidades y sentido de giro:

Valor τ	velocidad	Sentido del giro
1.9 ms	**más**	
1.8 ms		
1.7 ms		
1.6 ms	**menos**	
1.4 ms	**menos**	
1.3 ms		
1.2 ms		
1.1 ms	**más**	

Tabla 8. 1:

Existiendo una equivalencia de velocidad tal que:

- 1.9 ms tiene la misma velocidad que 1.1ms pero sentidos opuestos
- 1.8 ms tiene la misma velocidad que 1.2ms pero sentidos opuestos
- 1.7 ms tiene la misma velocidad que 1.3ms pero sentidos opuestos
- 1.6 ms tiene la misma velocidad que 1.4ms pero sentidos opuestos

El periodo utilizado es 20ms , que implica una frecuencia de 50 Hz, también se suelen emplear valores entre 10ms y 30ms. Si se elige un intervalo entre pulso y pulso inferior al mínimo puede interferir con la temporización interna del servo causando zumbido en cambio si es mayor puede provocar que se mueva en pequeños intervalos

Hay que destacar que para que el servo mantenga la misma velocidad y sentido del giro es necesario enviarle continuamente un pulso de anchura constante. Si dejamos de hacerlo el motor se parará.

8.3. 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 bajo nivel.

La necesidad de utilizar un lenguaje de bajo nivel radica en las complicaciones que pueden surgir debido a que solo disponemos de un microcontrolador para gobernar todo el sistema, lo cual implica un exhaustivo control de la memoria de programa y del tiempo de ejecución de cada elemento que compone el micro robot .

La consecuencia directa de la elección de dicho lenguaje acarrea en la complejidad de los programas desarrollados y en la posibilidad de encontrarnos con diversas complicaciones en temas de comunicación de dispositivos, control de una actividad etc.

Para desarrollar el control de los servomotores se utilizó el programa MPLAB, explicado en el capítulo 5. 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.

8.4. DESARROLLO DE APLICACIONES PARA CONTROLAR MOTORES HITEC HSR1422CR EN PLATAFORMAS HARDWARE EASYPIC4

En este apartado se detallan los distintos programas creados en lenguaje ensamblador, que reflejan el control de los servomotores mediante el entrenador EASYPIC4 (ver figura 8.4). Puesto que son funciones propias creadas, a priori, para una aplicación determinada se comentará con detalle el proceso de creación de la librería.

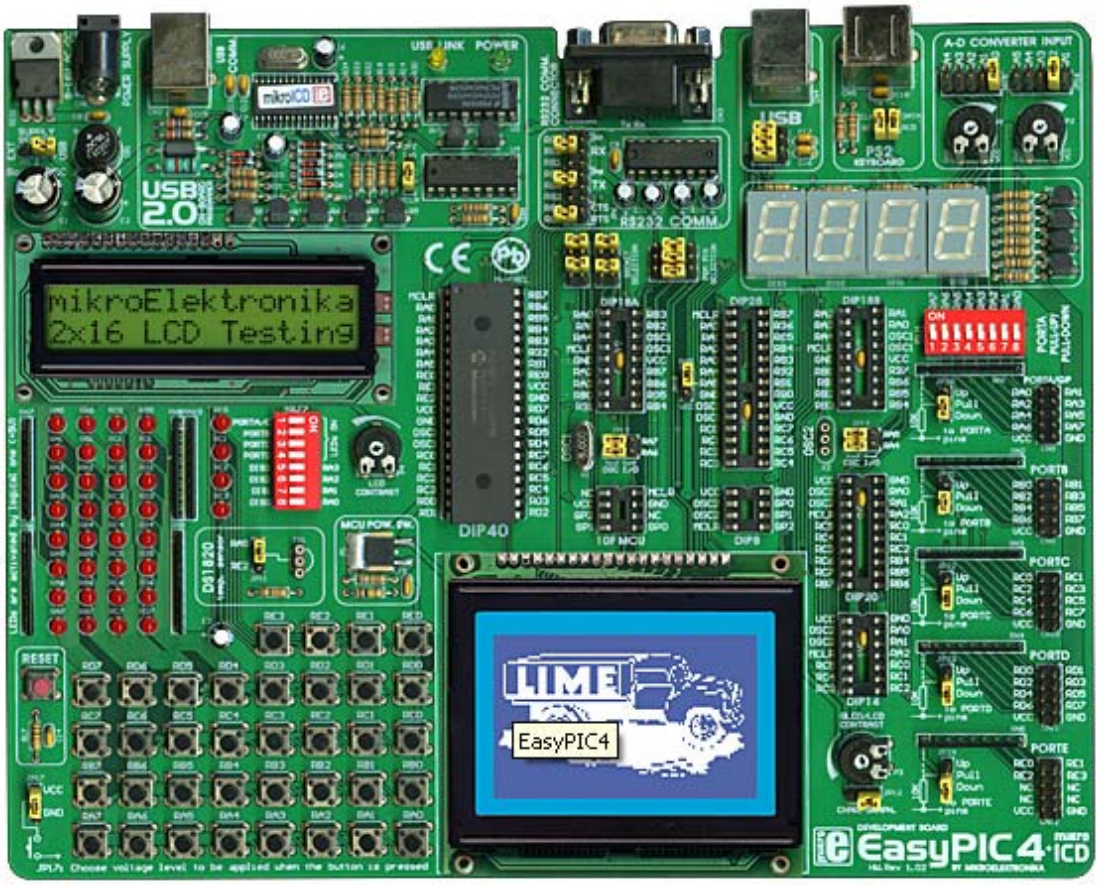


Figura 8. 4

Hay que mencionar que en este apartado solo se trabaja con un servomotor. También hay que destacar que la alimentación no se recibe por el puerto USB debido a los problemas que generó en un principio, por ello se usa alimentación externa para poder alimentar a los servomotores y a la placa

8.4.1. Aplicación PWM

1. **Nombre del fichero:** PWM.asm

2. **Descripción**

Programa que permite mover el servomotor en sentido anti horario a velocidad máxima ($\tau = 1.1ms$)

3. **Características principales del programa:**

En primer lugar se desarrolló esta pequeña aplicación, sin posibilidad de cambiar el parámetro τ mientras se ejecuta el programa. Para cambiar el sentido y la velocidad hay que cambiarlo manualmente en el código y después ensamblarlo para su ejecución.

Se definen las siguientes variables:

- **fams**(factor alto en milisegundos; τ) : es el ancho de pulso, es decir el tiempo que está a 5 V

- **fbms**(factor bajo en milisegundos): es el tiempo que está a 0 V

Estas dos variables tiene un formato especial. Para indicar que fams= 1.1 ms hay que multiplicar ese valor por 10 , luego quedaría fams= 11. Se realiza mismo procedimiento para fbms.

El motivo de este formato es muy simple, puesto que vamos a utilizar un temporizador de 100 microsegundos para obtener un valor de 1.1ms bastaría con multiplicar 11 veces 100 microsegundos para obtener 1100 microsegundos , o lo que es lo mismo 1.1 milisegundos.

Una pregunta que el lector debería de hacer es por qué no se puede cargar directamente en el registro del temporizador el valor deseado, en lugar de realizar lo expuesto en el párrafo anterior. La respuesta es porque deseamos una temporización fina y la forma de hacerlo con un microcontrolador PIC 16f877A es la expuesta en el párrafo anterior

El principal problema cuando se configura el Tmr0 como temporizador es el cálculo de los tiempos de temporización. Se puede utilizar la siguiente fórmula:

$$Temporización = T_{CM} \cdot PRESCALER \cdot (256 - CARGATMR0)$$

- Temporización, es el tiempo deseado
- T_{CM} , es el período de un ciclo de máquina e igual a $T_{CM} = 4 \cdot T_{OSC}$ .Para 20MHz $T_{OSC} = \frac{1}{f}$ por tanto $T_{CM} = 0.2\mu s$
- Prescaler, es el rango de divisor de frecuencia elegido.

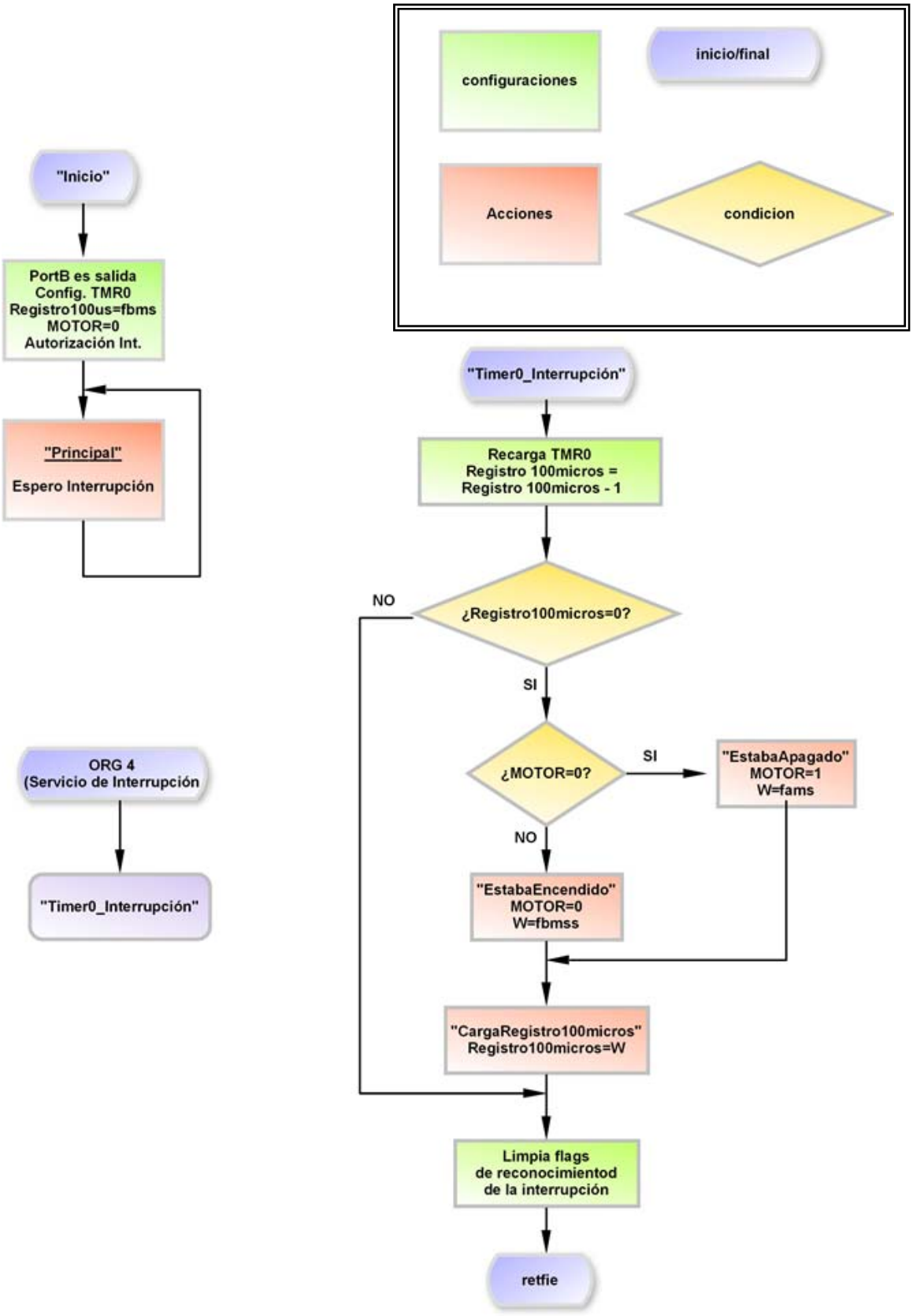
$(256 - CARGATMR0)$, es el número total de impulsos a contar por el TMR0 antes de desbordarse en la cuenta ascendente. "CARGATMR0" es el valor cargado inicialmente en el TMR0.

Luego como queremos temporizar 100 microsegundos con un preescaler de 2 nos da un valor teórico de cargatmr0 = 6. Pero debido a que nuestra aplicación requerimos temporizaciones precisas hay que tener en cuenta el tiempo de ejecución de las instrucciones, saltos etc. El valor de carga de TMR0 es algo mayor al obtenido y se ha realizado ajustes finos con la instrucción nop. La forma más sencilla de calcularlo es experimentando con el simulador MPLAB y el reloj de la ventana Stopwatch obteniendo un valor de cargatmr0=11

Como se expuso anteriormente en el capítulo 2 el TMR0 se puede utilizar aplicando interrupciones para que se pueda ejecutar otras tareas mientras se temporiza .

De aquí en adelante se trabajara con esa filosofía dado que se gobierna todo el micro robot con un microcontrolador

4. **Diagrama de flujo**



5. Código

```
;////////////////////////////////////////////////////////////////////////////////
;//                                                                            //
;//                              PWM.asm                                       //
;//                                                                            //
;//                    PFC DIEGO P. GONZALEZ GONZALEZ                          //
;//                                                                            //
;////////////////////////////////////////////////////////////////////////////////

; DESCRIPCIÓN ********************************************************************

;DESCRIPCIÓN PROGRAMA: El programa consiste en introducir dos variables,
;la primera el tiempo de valor bajo  y la segunda el  tiempo de valor
;alto de la señal cuadrada. Así pues se introduce de la siguiente manera:
; 18.9ms---->189  ;  1.1ms---->11 .
;Para que el motor gire con otra velocidad hay que cambiar dichos valores.
;RESULTADOS: Se consigue unos valores precisos, pero el motor no es muy sensible
;con valores del  factor bajo y este se puede alterar en +5ms aprox.

; ZONA DE DATOS ******************************************************************

      __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _HS_OSC
      LIST      P=16F877A
      INCLUDE  <P16F877A.INC>


      CBLOCK  0x20
      Registro100micros               ; Registro auxiliar para conseguir una
      ENDC                            ; temporización fina.

CARGA_fbms          EQU    d'189'     ; Factor bajo
CARGA_fams          EQU    d'11'      ; Factor alto
TMR0_Carga100micros EQU    d'64'      ; Valor obtenido con stopwatch
#DEFINE             MOTOR  PORTB,0    ; para contar cada 100micros


; ZONA DE CÓDIGOS ****************************************************************

      ORG    0
      goto   Inicio
      ORG    4                        ; Vector de interrupción.
      goto   Timer0_Interrupcion
Inicio
      bsf          STATUS,RP0          ; Acceso al Banco 1.
      bcf          MOTOR               ; Línea del MOTOR configurada como salida.
      movlw  b'00001000'
      movwf  OPTION_REG                ; Sin Prescaler, se le asigna al WDT
      bcf          STATUS,RP0          ; Acceso al Banco 0.
      movlw  TMR0_Carga100micros       ; Carga el TMR0 con valor 100 micros
      movwf  TMR0
      movlw  CARGA_fbms
      movwf  Registro100micros
    bcf              MOTOR             ; ponemos a 0 la señal de control del
servomotor
      movlw  b'10100000'                ; durante 18.9 ms
      movwf  INTCON                    ; Autoriza interrupción del TMR0 (T0IE)
                                       ; y la GIE.
Principal                              ; No puede pasar a modo bajo consumo porque
      goto   $                         ; el Timer 0 se detendría.
```

```
; Subrutina "Timer0_Interrupcion" ------------------------------------------------
------
;
; el PIC trabaja a una frecuencia de 8 MHz, el TMR0 evoluciona cada 0,5
microsegundos.


Timer0_Interrupcion
       movlw  TMR0_Carga100micros
       movwf  TMR0                          ; Recarga el TMR0.
       decfsz Registro100micros,F           ; Decrementa el contador.
       goto   FinInterrupcion
       btfsc  MOTOR                         ; Testea el último estado del MOTOR.
       goto   EstabaEncendido
EstabaApagado

       bsf          MOTOR                  ; Estaba apagado y lo enciende.
       movlw  CARGA_fams                    ; Repone el contador nuevamente para que
       goto   CargaRegistro100micros        ; esté fams encendido.
EstabaEncendido

       bcf          MOTOR                   ; Estaba encendido y lo apaga.
       movlw  CARGA_fbms                    ; Repone el contador nuevamente

CargaRegistro100micros
       movwf  Registro100micros
FinInterrupcion
       bcf          INTCON,T0IF              ; Repone flag del TMR0.
       retfie
                                            ; Retorno de interrupción.

       END
```

8.4.2. Aplicación cambio de velocidad y sentido

1. **Nombre del fichero:** controlhitec.asm

2. **Descripción**

Programa que permite cambiar la velocidad y el sentido del motor , pulsando uno de los botones de la fila RC de la placa EasyPic4

3. **Características principales del programa:**

Este programa se basa en el anterior salvaguardando la diferencia que ahora si se puede cambiar la velocidad y sentido del giro del servomotor . Para ello solo hace falta pulsar unos de los botones comentados.

Hay que destacar la creación de una nueva variable llamada Contador cuya finalidad no es otra que la de refrescar el valor de la entrada. Posee un pequeño fallo y es que cada vez que pasa por ORG 4 se produce 0.5us de retraso ,pero no es un fallo grave dado que afecta a la tercera o cuarta cifra decimal significativa del parámetro τ y por tanto no afecta al control del motor .

Se procedió a realizar una tabla de verdad para relacionar el botón que se está pulsando con la velocidad y giro deseado del servomotor. La tabla de verdad se resume de la siguiente forma:

Aprovechando la disposición de la siguiente hilera de botones como se muestra en la figura 8.5,que posee la EasyPIC4.

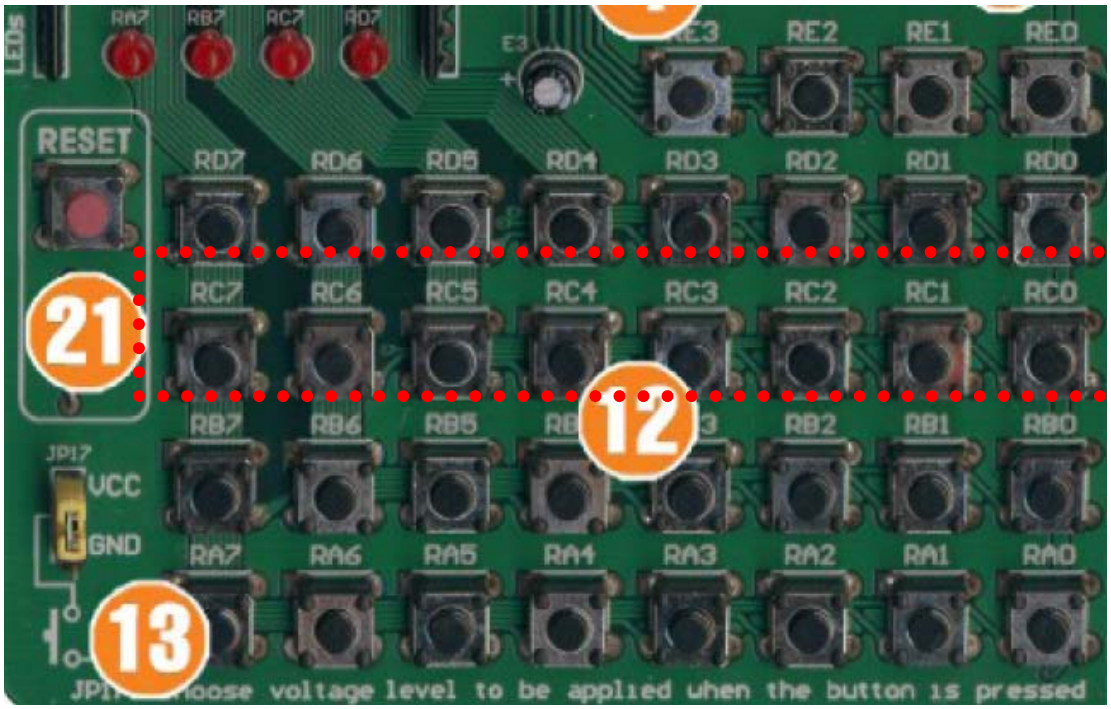


Figura 8. 5 : Fila de botones RD

Y teniendo en cuenta que sólo se puede pulsar un botón a la vez por ende generamos la tabla de verdad tal que :

Valor Bin fila RC	Valor Dec. Fila RC	Valor cte	Valor τ	velocidad	Sentido del giro
RC0 = 00000001	1	0	1.9 ms	**más**	
RC1 = 00000010	2	1	1.8 ms		
RC2 = 00000100	4	2	1.7 ms		
RC3 = 00001000	8	7	1.6 ms	**menos**	
RC4 = 00010000	16	15	1.4 ms	**menos**	
RC5 = 00100000	32	31	1.3 ms		
RC6 = 01000000	64	63	1.2 ms		

RC7 = 10000000	128	127	1.1 ms	**más**

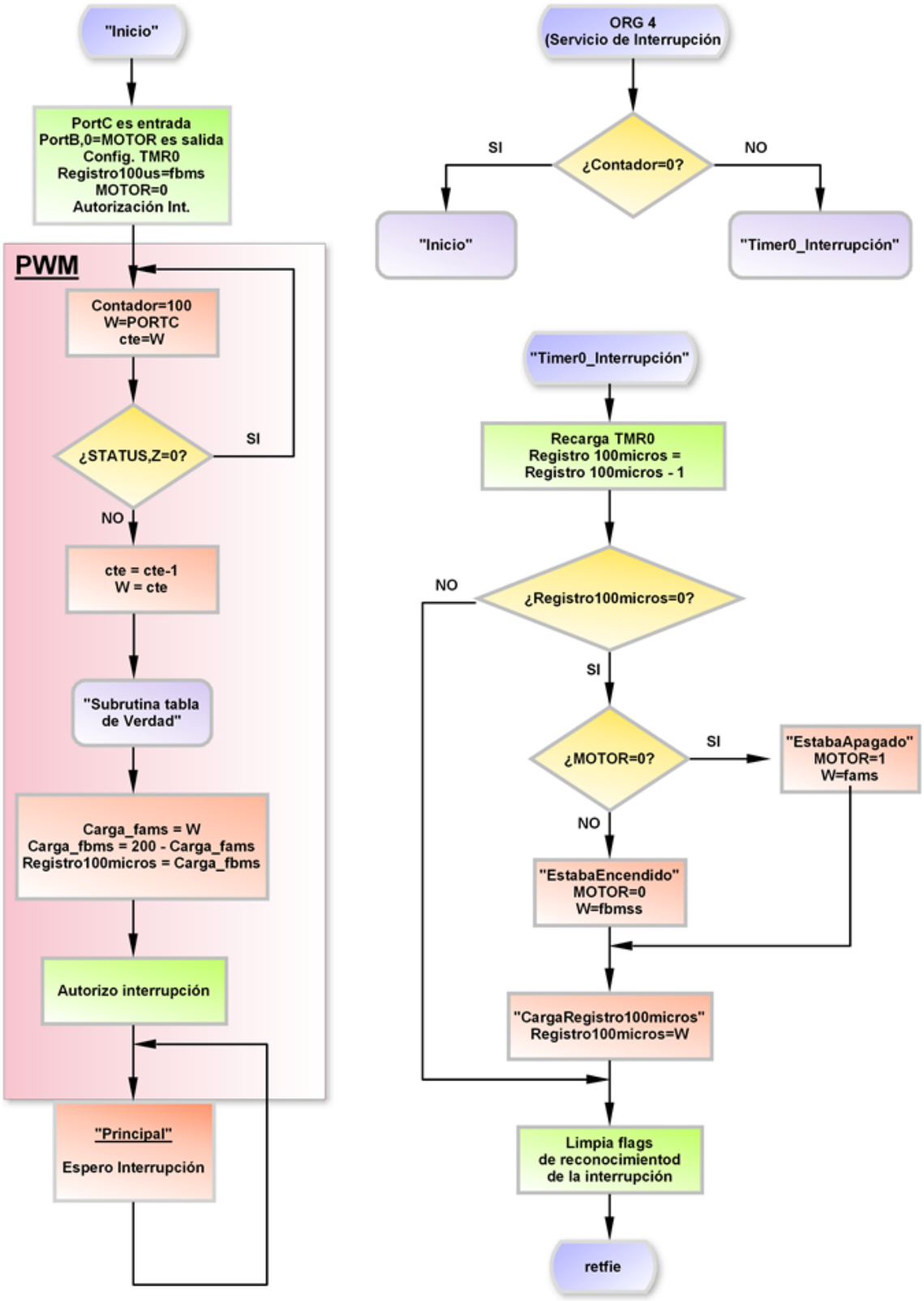
Tabla 8. 2

Cte es una constante que sirve para obtener el valor de τ puesto que ,observando el código, se empieza a contar en la tabla de verdad a partir del valor numérico 0 no del 1, de ahí que disminuya en una unidad.

Se observa que no se utiliza el valor de ancho de pulso 1.5 ms, como se comentó anteriormente, si queremos que esté parado basta con no enviarle ninguna señal.

Por último indicar que el factor bajo (fbms) se calcula a partir de la operación 200-fams.

4. Diagrama de flujo



5. Código

```
;////////////////////////////////////////////////////////////////////////////
;//                                                                        //
;//                            controlhitec.asm                            //
;//                                                                        //
;//                     PFC DIEGO P. GONZALEZ GONZALEZ                     //
;//                                                                        //
;////////////////////////////////////////////////////////////////////////////

;DESCRIPCIÓN DEL PROGRAMA: Partiendo del archivo PWM.asm se ha configurado una
entrada
;para dar diferentes valores de factor alto y bajo,dicha entrada es la C y esta
codificada
;de la siguiente manera:  RC7   RC6   RC5   RC4  |   RC3   RC2   RC1   RC0
;                         1.1   1.2   1.3   1.4  |   1.6   1.7   1.8   1.9
;                          SENTIDOR ANTIHORARIO        SENTIDO HORARIO
;
;Para ello se utiliza una tabla de verdad y mientras se este pulsando un pulsador ,el
motor
;girara en el sentido y velocidad que corresponda a dicho pulsador.
;RESULTADOS: Se produce un desfase en la parte baja de la señal de +5ms aprox, en
principio
; parece que no afecta a la velocidad deseada(COMPROBAR). Se ha advertido que de RC6
a RC7
; y de RC1 A RC0 no hay aparentemente una variación significativa de la
velocidad(COMPROBAR)
; ZONA DE DATOS
*****************************************************************

      __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _HS_OSC
      LIST      P=16F877A
      INCLUDE  <P16F877A.INC>


      CBLOCK  0x40                        ;CUIDADO!!! ANTES POSICIÓN 0XC = FALLO!!
      Registro100micros                             ; Registro auxiliar para
conseguir una                   ; temporización fina.
    CARGA_fbms   ;Factor bajo ej: 18.9ms--->189
    CARGA_fams        ;Factor alto ej: 1.1ms--->11
    cte
    Contador
    ENDC

TMR0_Carga100micros EQU    d'64'; Valor obtenido con stopwatch para contar cada
100micros

#DEFINE             MOTOR  PORTB,0



; ZONA DE CÓDIGOS
*****************************************************************
```

```
; ZONA DE CÓDIGOS
*****************************************************************

       ORG    0
       goto   Inicio
       ORG    4                                          ; Vector de
interrupción.
    decfsz Contador                 ; CUIDADO:Se produce un desfase e 0.5us en la
señal,sirve para refrescar
    goto     Timer0_Interrupcion     ; el valor de la entrada


Inicio
       bsf           STATUS,RP0                       ; Acceso al Banco 1.
       bcf           MOTOR                                  ; Línea del MOTOR
configurada como salida.
    movlw  b'11111111'
    movwf     TRISC                   ; Las lineas de C como entrada
       movlw  b'00001000'
       movwf  OPTION_REG                       ; Sin Prescaler, se le asigna al WDT
       bcf           STATUS,RP0                        ; Acceso al Banco 0.
       movlw  TMR0_Carga100micros       ; Carga el TMR0 con valor 100 micros
       movwf  TMR0
    movwf  Registro100micros

PWM

    movlw   .100               ; Valor del contador
    movf    Contador
       movf   PORTC,W                          ; Lee el valor de las variables de
entrada.
    movwf   cte
    btfsc   STATUS,Z;          ;Si no tengo nada contectado vuelve a Principal,
Z=0-->PWM
    goto   PWM
    movwf   cte                ;utilizamos una constante para drecrementar en 1 su
valor
    decf    cte                ;para despues acceder al valor de la tabla de
verdad
    movf    cte,W
       call   TablaVerdad              ; Obtiene la configuración de salida
       movwf CARGA_fams
    sublw   .200 ;antes 192?¿
    movwf  Registro100micros
    movwf   CARGA_fbms
    movlw    b'10100000'
       movwf  INTCON                           ; Autoriza interrupción del TMR0
(T0IE) y la GIE.

Principal
       goto   $                   ; No puede pasar a modo bajo consumo porque no
funcionaria el tmro
; Subrutina "TablaVerdad" ---------------------------------------------------------
------
```

```
; Subrutina "TablaVerdad" ---------------------------------------------------
------
;
TablaVerdad ; Solo puede haber un pulsador pulsado, esta configurado para la linea
RC

       addwf  PCL,F
       DT     d'19',d'18',.0,d'17',.0,.0,.0,.16,.0,.0,.0,.0,.0,.0,.0,d'14'
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,d'13'
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,d'12'
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0
    DT .0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,.0,d'11'


; Subrutina "Timer0_Interrupcion" -------------------------------------------------
------
;
; el PIC trabaja a una frecuencia de 8 MHz, el TMR0 evoluciona cada 0,5
microsegundo.


Timer0_Interrupcion

       movlw  TMR0_Carga100micros
       movwf  TMR0                                ; Recarga el TMR0.
       decfsz Registro100micros,F              ; Decrementa el contador
Registro100micros*(fa ó fb)
       goto   FinInterrupcion
       btfsc  MOTOR                                    ; Testea el último estado del
MOTOR
       goto   EstabaEncendido
EstabaApagado

       bsf          MOTOR                                   ; Estaba apagado y lo
enciende.
       movf   CARGA_fams,W                    ; Repone el contador nuevamente para
que esté
       goto   CargaRegistro100micros          ; 800 ms encendido.
EstabaEncendido

       bcf          MOTOR                             ; Estaba encendido y lo apaga.
       movf   CARGA_fbms,W                    ; Repone el contador nuevamente para
que esté

CargaRegistro100micros
       movwf  Registro100micros
FinInterrupcion
       bcf          INTCON,T0IF                       ; Repone flag del TMR0.
    retfie


       END
```

8.5. LIBRERIA PARA CONTROLAR MOTORES HITEC HSR 1422 CR EN PROTOBOARD OLIMEX.

Quizás antes de narrar como se ha desarrollado la librería en la protoboard OLIMEX , hay que destacar que se probó con otra placa denominada tarjeta de control MSE-F87X, tal y como se comento en el capítulo 4.

Esta tarjeta capaz de soportar los microcontroladores PIC16F873/F876 de 28 patillas o el PIC16F874/F877 de 40patillas. Dichos microcontroladores pueden venir pre grabados de fábrica con el programa monitor PICMOS desarrollado por Ingeniería de Microsistemas Programados S.L. que, junto con el programa Real_PIC para PC, facilita la edición, el ensamblado, la grabación y depuración de los programas de aplicación del usuario.

El principal problema de ese programa era que el servicio de interrupciones quedaba vetado solo para dicho programa. Se procedió a utilizar un PIC 16f877A con programa propio pero con resultado nulo al grabar el programa con el programa Real_PIC en reiteradas ocasiones, esto fue así porque la placa poseía un fallo de fábrica.

Por ello se pasó a la protoboard OLIMEX (ver figura 8.6) , descrita en el mismo capítulo 4.

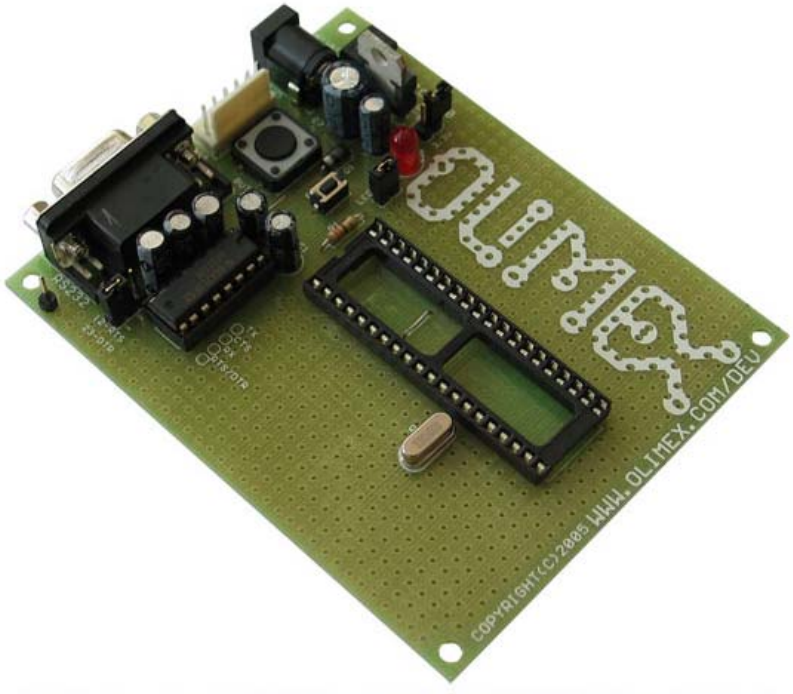


Figura 8. 6: Tarjeta de control OLIMEX

Se ha modificado la placa para poder conectar los diversos componentes que forman el micro robot. La alimentación se realiza mediante una batería.

En este apartado del capítulo todo lo que se desarrolle estará orientado en el control del movimiento del micro robot “MYBOT”. Se ha diseñado una librería capaz de reproducir todos los movimientos descritos en el capítulo 6.

La estructura del programa principal es muy simple dado que se pretende que el control del movimiento del micro robot también sea muy simple. Para realizar dicho cometido se procede a la creación de unas MACROS (FUNCIONES).

Estas MACROS se pasan unos valores y realizan la función que se les ha encomendado. La configuración general de las entradas y las salidas , variable etc. se encuentran en la subrutina INI .Se advierte que se han creado variables nuevas, modificado el nombre de algunas existentes y eliminando otras.

Un aspecto importante destacable es que la librería desarrollada (a partir de ahora la referiremos como Mlibreria) hay que incluirla en el programa principal al principio debido a, que antes de utilizar las MACROS hay que definirlas. Por tanto en la memoria de programa se escribe primero la librería y luego las instrucciones que posee el programa principal

En Mlibreria (ver AnexoC) , todos los cálculos que se realicen se tomará como referencia el motor derecho . En la imagen siguiente, suponiendo que es una vista superior sin estructura, se indica donde se ha conectado los servomotores y que referencia hay que tomar para realizar los cálculos de factores altos y bajos.

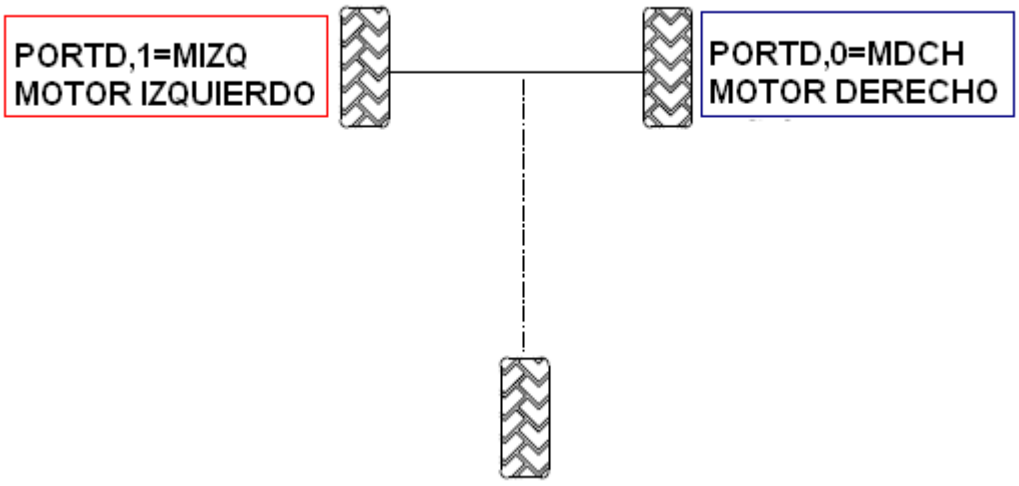


Figura 8. 7 : “MYBOT” sin estructura

8.5.1. MACRO GIRACENTRO

1. **Nombre del fichero:** MLIBRERIA.asm

2. **Descripción :**

Esta función pretende que el micro robot realice un giro completo sobre su eje. Para realizar dicha acción tendremos que proporcionar a los dos servomotores la misma señal. En la figura 8.8 y 8.9 se puede observar los movimientos que se realizarán.

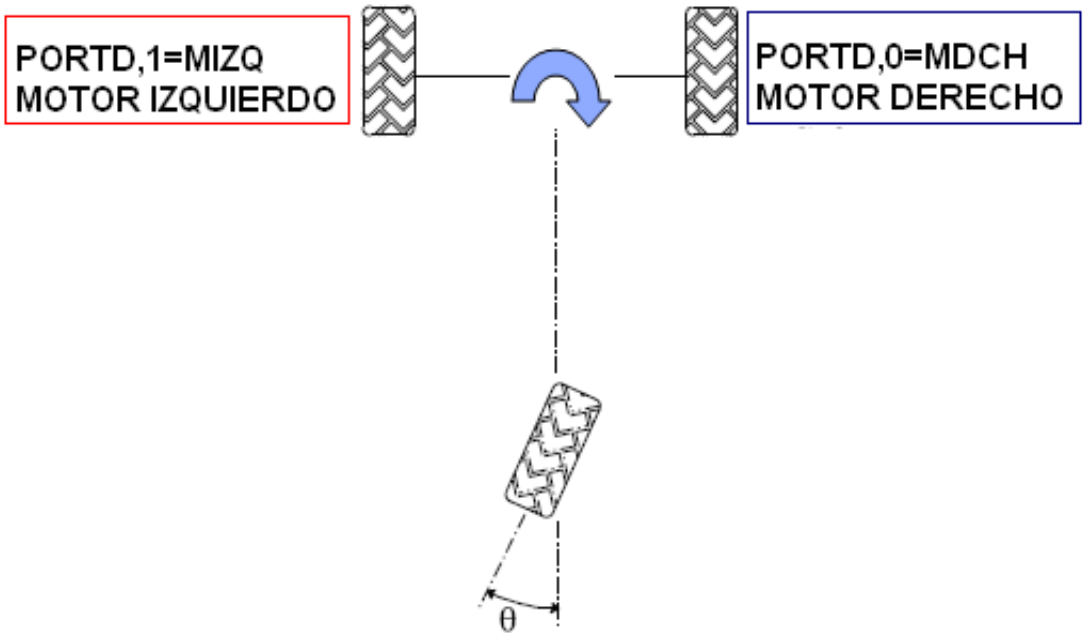


Figura 8. 8 : Giro sobre su propio eje en sentido horario

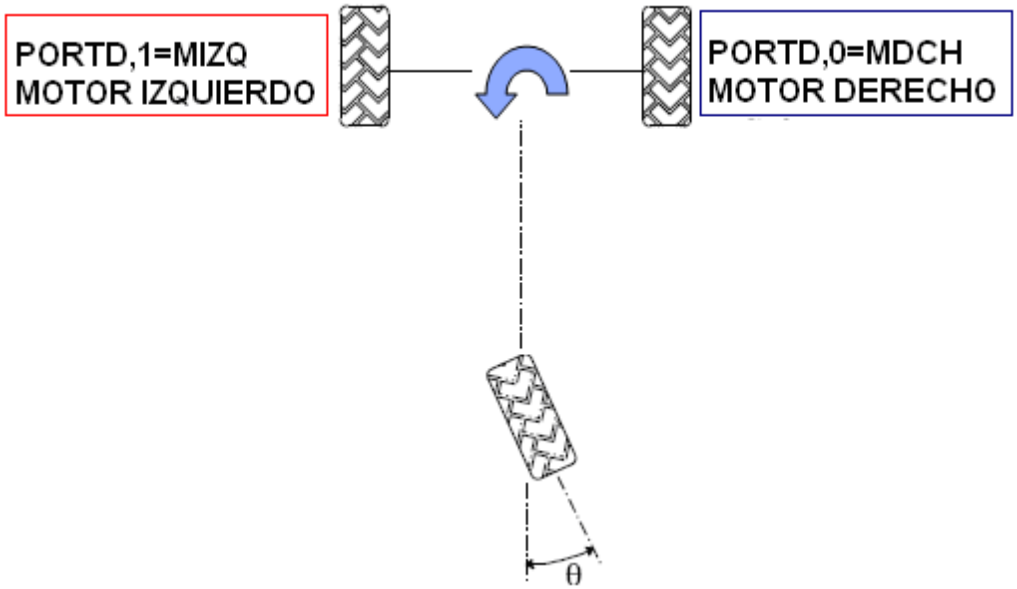


Figura 8. 9: Giro sobre su propio eje en sentido anti horario

3. Características principales del programa:

Lo primero que hay que matizar de esta función son los valores que se le pasan a la función MACRO GIRACENTRO:

- **VelSent** : variable que indica la velocidad y el sentido del robot
- **Tiempo:** el tiempo que deseamos que este realizando esa tarea

Estas variables están codificadas para poder operar con el PIC 16F877A.

La variable **VelSent** toma valores entre el 0 y el 7 y se divide de la siguiente forma:

- del 0 al 3 sentido micro robot anti horario
- del 4 al 7 sentido micro robot horario

La siguiente tabla recoge todos los datos necesarios:

VelSent	Valor τ	velocidad	Sentido del giro Servomotor derecho	Sentido del giro micro robot
0	1.9 ms	más		
1	1.8 ms			
2	1.7 ms			
3	1.6 ms	menos		
4	1.1 ms	más		
5	1.2 ms			
6	1.3 ms			
7	1.4 ms	menos		

Tabla 8. 3

Tal y como hemos indicado al principio vamos a tener en cuenta como referencia siempre al motor derecho para realizar los cálculos pertinentes.

Dado que en los dos servomotores se introduce la misma señal, es decir el mismo factor alto ,el sentido de giro de los servos será el mismo . Esto provocará que el micro robot tenga un par de rotación de sentido opuesto al del servomotor derecho.

En la figura 8.10 podemos observar el sentido de giro de los servomotores si los estuviéramos mirando de frente. Las flechas rojas indican el sentido de las fuerzas que nos proporciona los servomotores. En este caso particular como el motor derecho tiene sentido anti horario, si nos vamos a la tabla 8.2 observamos que el “MYBOT” poseerá un sentido horario.

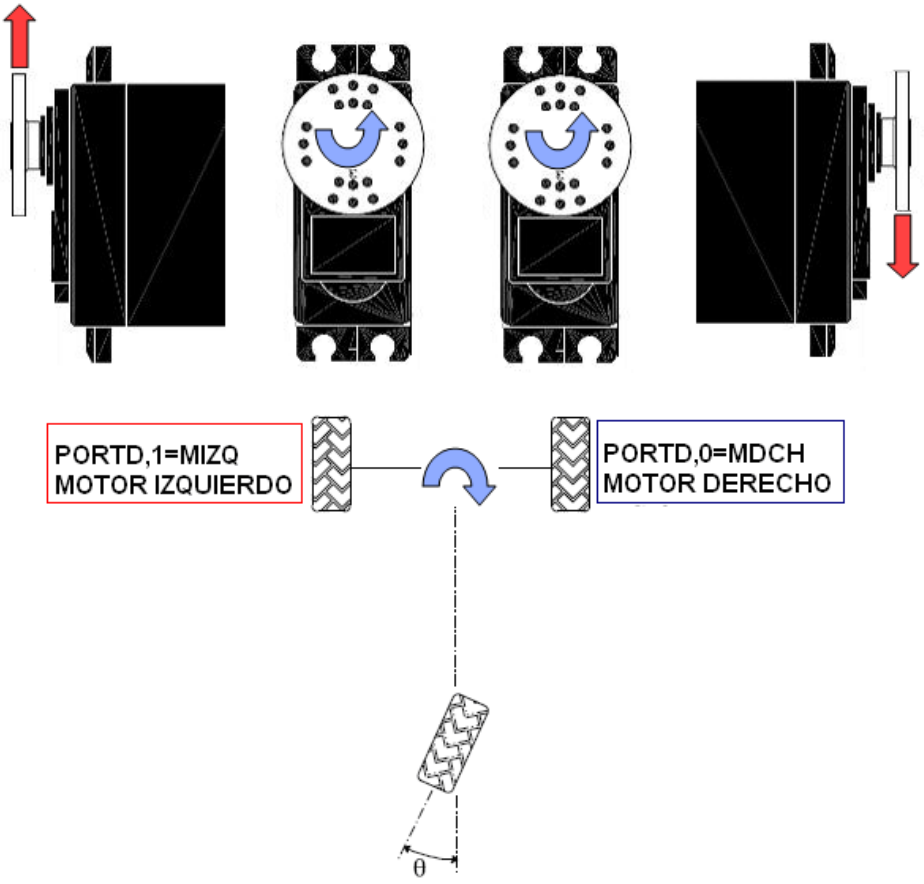


Figura 8. 10

La variable **Tiempo** esta codificada de la siguiente forma:

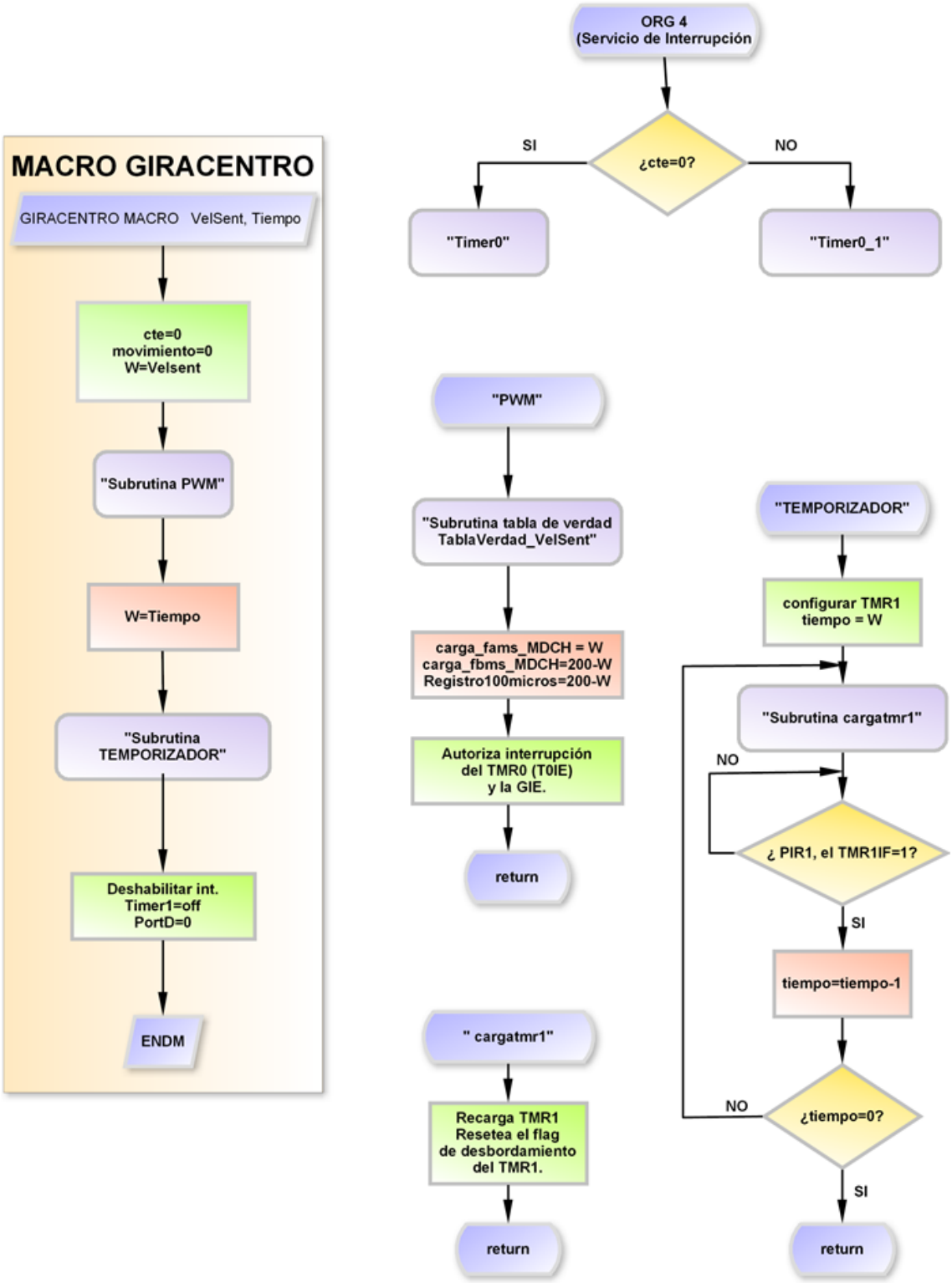
- 1→100ms;
- 10→1s,
- 100→10s etc.
- hasta un valor máximo de 256→ 25.6s

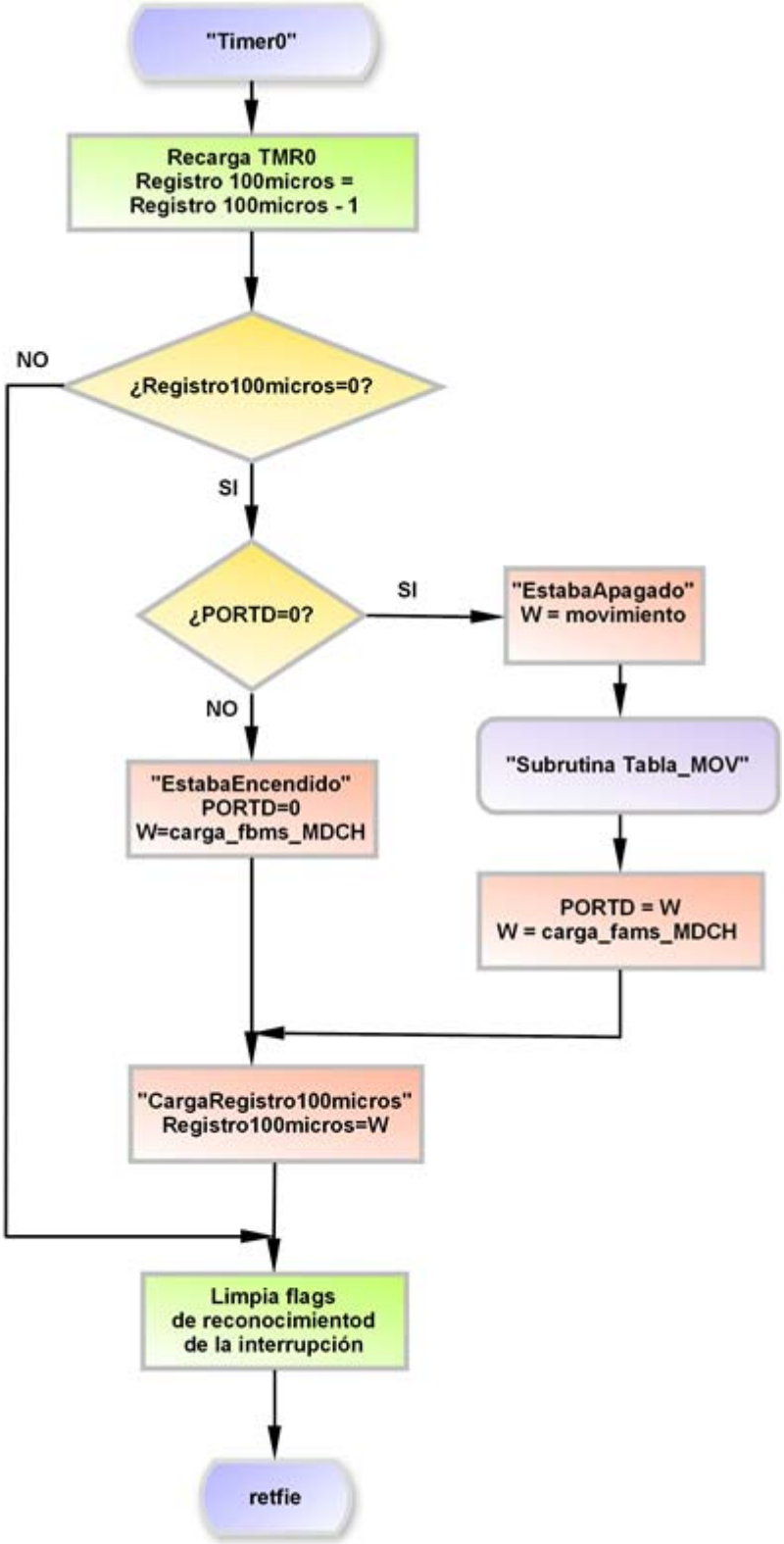
Para esta macro no se hará uso de la subrutina Timer0_1 . La subrutina encargada de generar los movimientos de giro de los servomotores, es decir hacer la señal PWM , es la subrutina de interrupción Timer0, para ello se utiliza la constante cte.

El cálculo del factor alto y bajo se hace uso de la la subrutina PWM y la variable movimiento sirve para indicar que servomotor se desea que este funcionando:

- Movimiento =0 → 2 servomotores encendido
- Movimiento =1 → 2 MIZQ encendido y MDCH apagado
- Movimiento =2 → 2 MDCH encendido y MIZQ apagado

4. <u>Diagrama de flujo</u>





5. **Código:** Ver anexo C

8.5.2. MACRO GiraT

1. **Nombre del fichero:** MLIBRERIA.asm

2. **Descripción :**

Función que permite girar a la izquierda o a la derecha, hacia delante o hacia atrás. Para realizar dicha acción tendremos que proporcionar a un servomotores la señal de control PWM y al otro servomotor no entregarle nada. En la figura 8.11 se puede observar el movimiento que realizará.

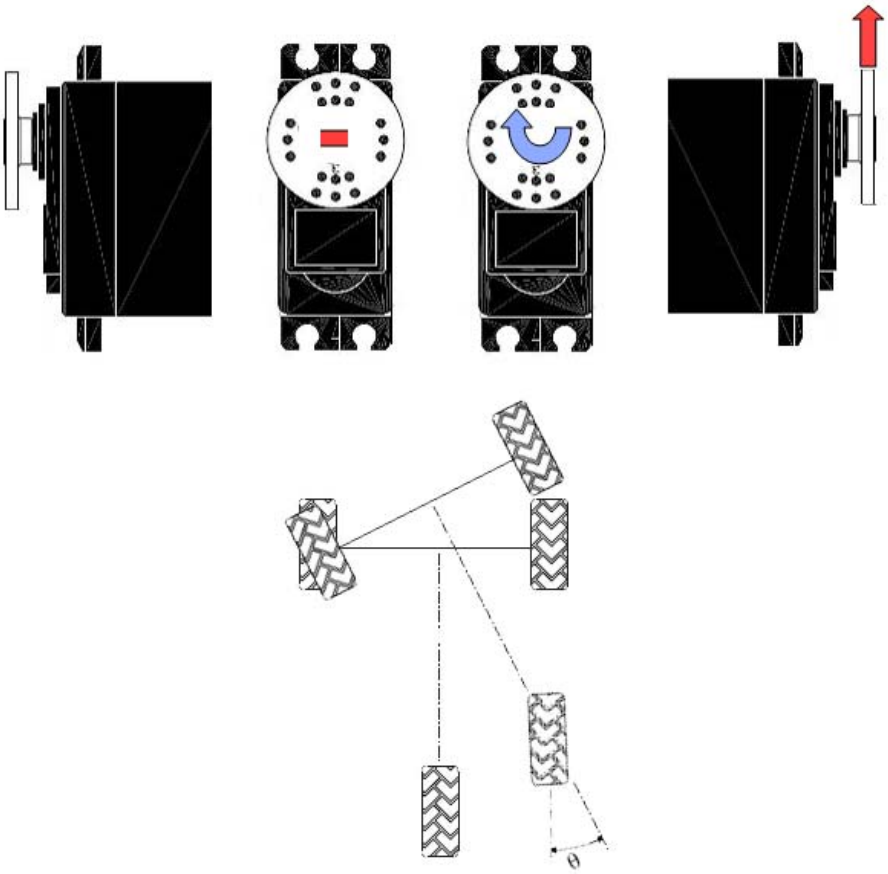


Figura 8. 11

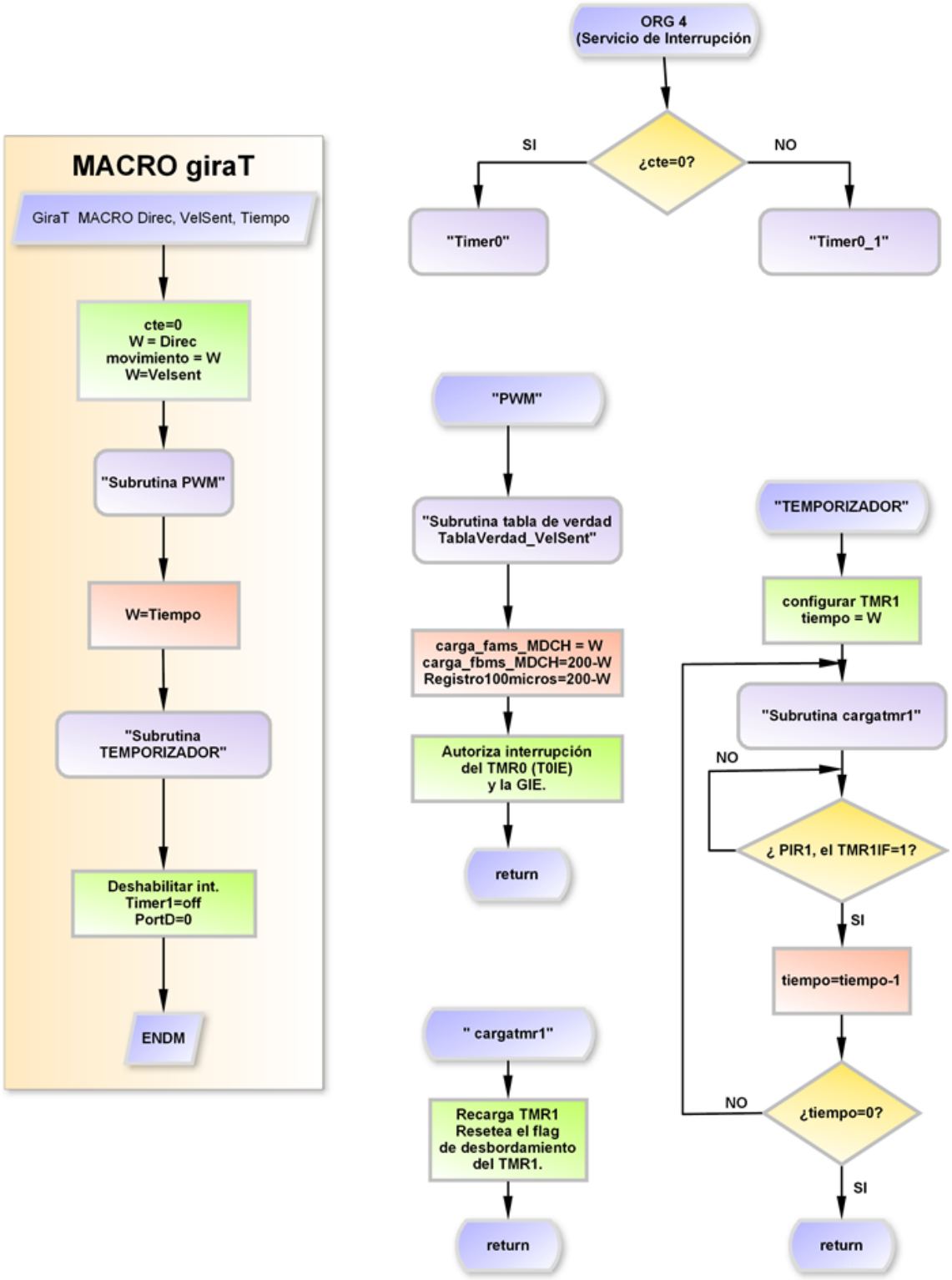
3. Características principales del programa:

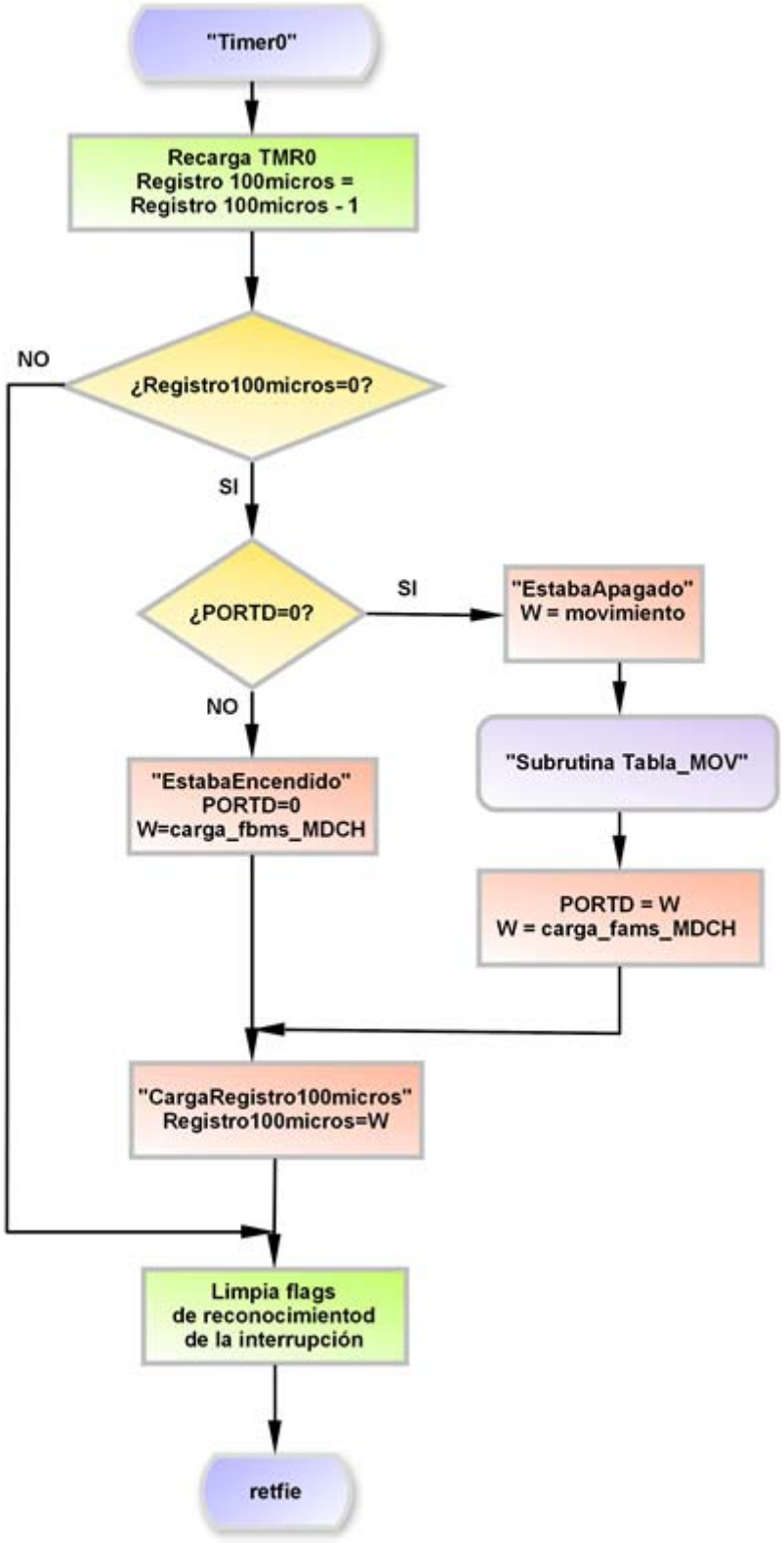
Esta macro es prácticamente idéntica en cuestión de código, las únicas diferencias que posee son que en esta función solo hay un servo al que se le manda la señal de control mientras que al otro servomotor no se le envía nada, por ello se utiliza una variable más denominada Direc ,que no deja de ser la variable movimiento. La otra diferencia es el sentido de codificación de la variable VelSent . El resto es igual. En la siguiente tabla se muestra como se codifica

Valor Direc	VelSent	Valor τ	velocidad	Sentido giro micro robot	Sentido del giro del servomotor
1	0	1.9 ms	más	derecha	
1	1	1.8 ms			
1	2	1.7 ms			
1	3	1.6 ms	menos	atrás	
1	4	1.1 ms	más	derecha	
1	5	1.2 ms			
1	6	1.3 ms			
1	7	1.4 ms	menos	adelante	
2	0	1.9 ms	más	izquierda	
2	1	1.8 ms			
2	2	1.7 ms			
2	3	1.6 ms	menos	adelante	
2	4	1.1 ms	más	izquierda	
2	5	1.2 ms			
2	6	1.3 ms			
2	7	1.4 ms	menos	atrás	

Tabla 8. 4

4. Diagrama de flujo





5. **Código:** Ver anexo C

8.5.3. MueveRectoT

1. **Descripción :**

Función que permite realizar un movimiento rectilíneo, hacia delante o hacia atrás. Como se puede observar en la figura 8.12 los servomotores poseen sentidos opuestos.

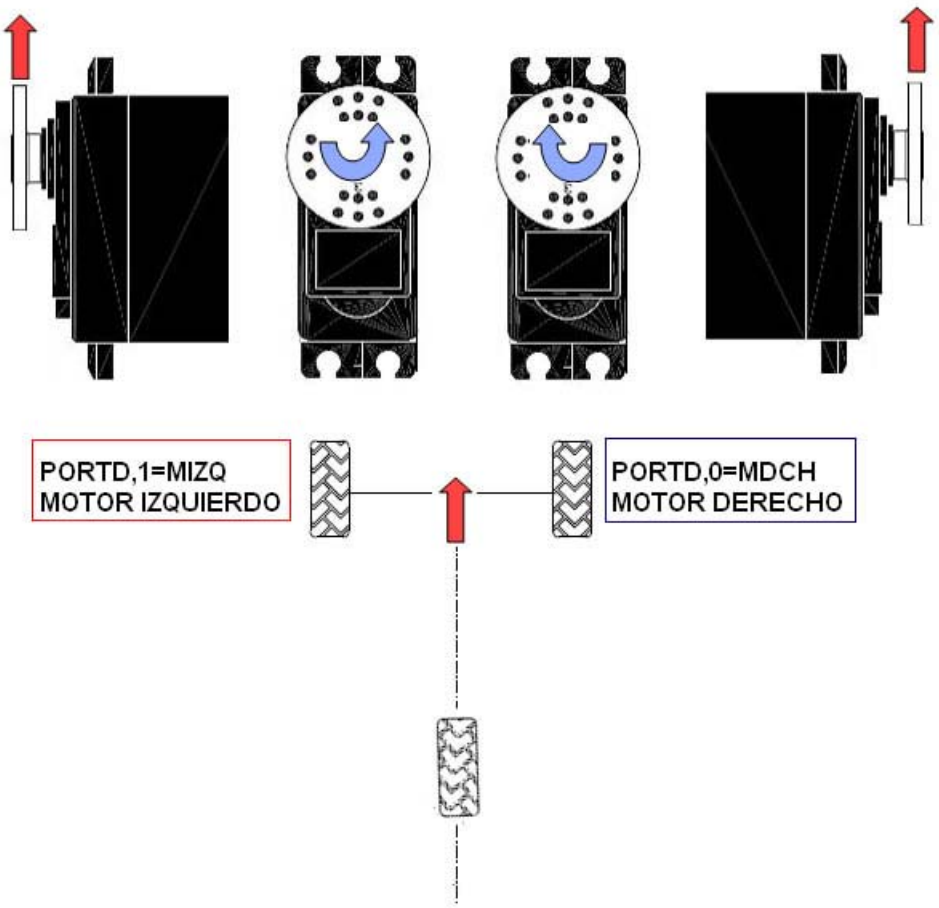


Figura 8. 12

2. Características principales del programa:

Esta macro es algo distinta pero conserva la forma de las otras dos. Lo primero que hay que destacar es que ya no tenemos dos señales idénticas como en GIRACENTRO , para que el micro robot posea una dirección rectilínea debemos enviar dos señales PWM con distinta TAU pero equivalentes . Es decir que los servomotores posean distintos sentidos del giro pero que posean también la misma velocidad. En la tabla 8.1 se dedujo lo expuesto.

La principal diferencia de esta macro es el uso de la subrutina PWM2 , encargada de calcular los factores altos y bajos del motor derecho y el factor alto del motor izquierdo.

Así pues, se elije una Velocidad con un sentido (VelSent) y llamamos a dicha subrutina, esta llamará a otra , encargada de retornarnos el valor del factor alto del servomotor derecho para esa VelSent. Acto seguido le restamos 200 para obtener el factor bajo para dicho servomotor. Y después se calcula el valor del factor alto del servomotor izquierdo restando 30 a factor alto del servomotor derecho. No se calcula el valor del factor bajo de servomotor derecho debido a que ya se hace en un cálculo en la subrutina Timer0_1. En este punto hay que recalcar que se utiliza un formato especial para las variables factor altor y bajo, ya descrito en el apartado 4.1. Otra tarea que realiza esta subrutina es analizar si el micro robot tiene el sentido hacia delante o hacia atrás, se hace uso para ello de la constante cte2.

La otra diferencia es el uso de la subrutina de interrupción Timer0_1(cte=1). En gran parte es equivalente a la Timer0 , pero tiene la diferencia de que ahora trabaja con dos señales distintas , influyendo en el sentido de los servomotores. En la siguiente figura se expone cual es la estrategia de control.

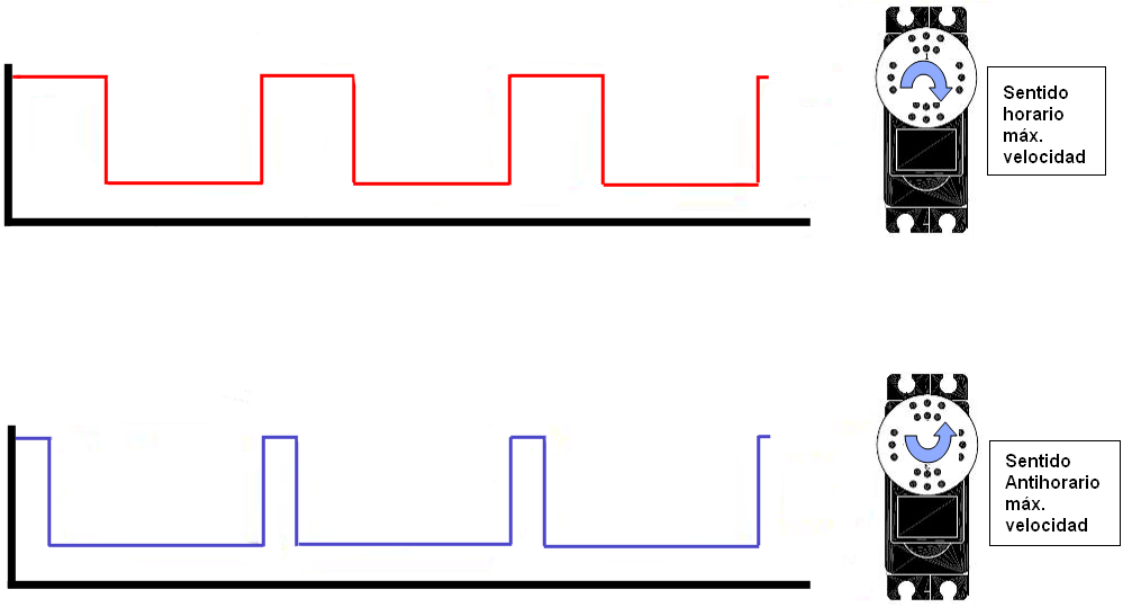
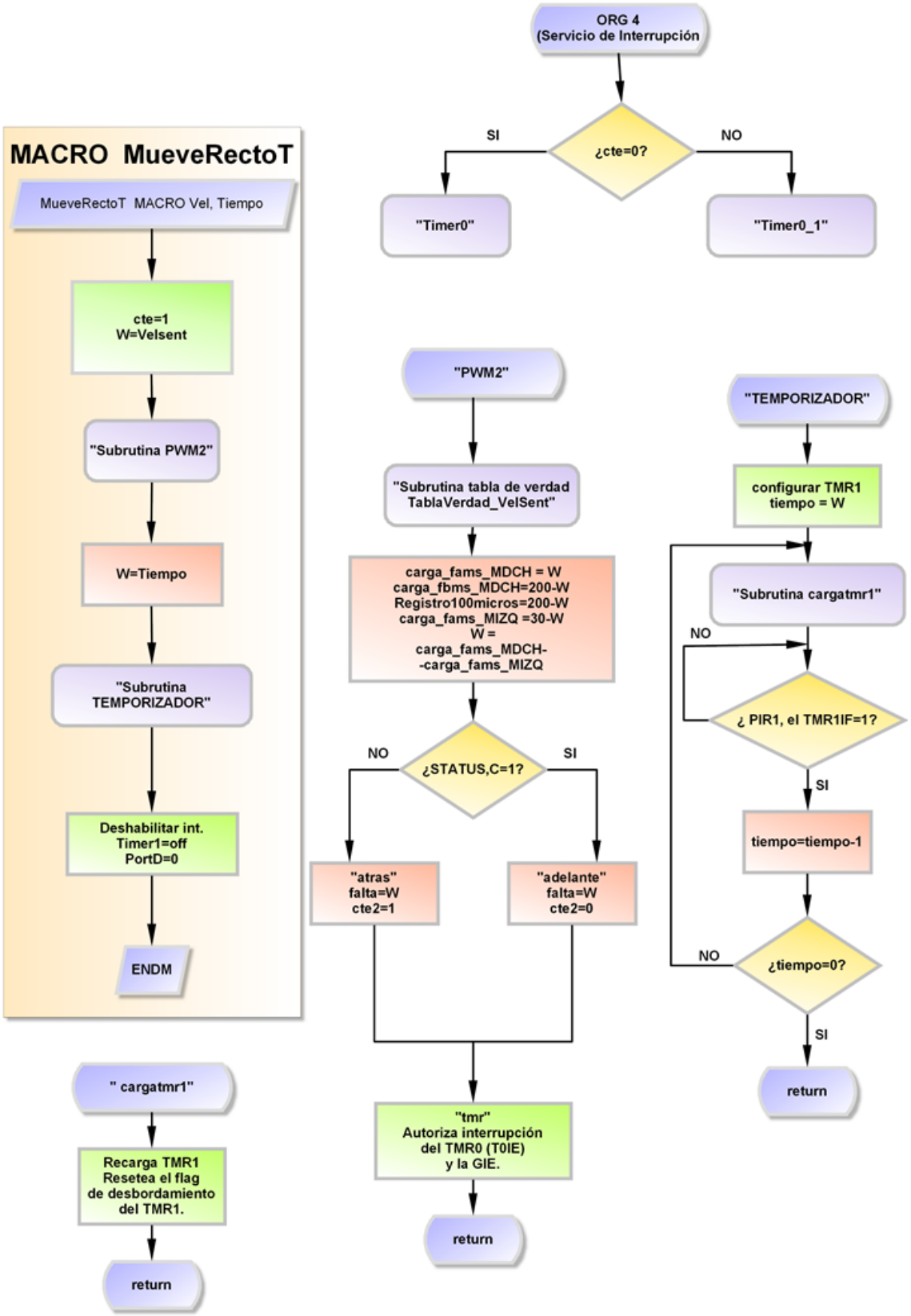
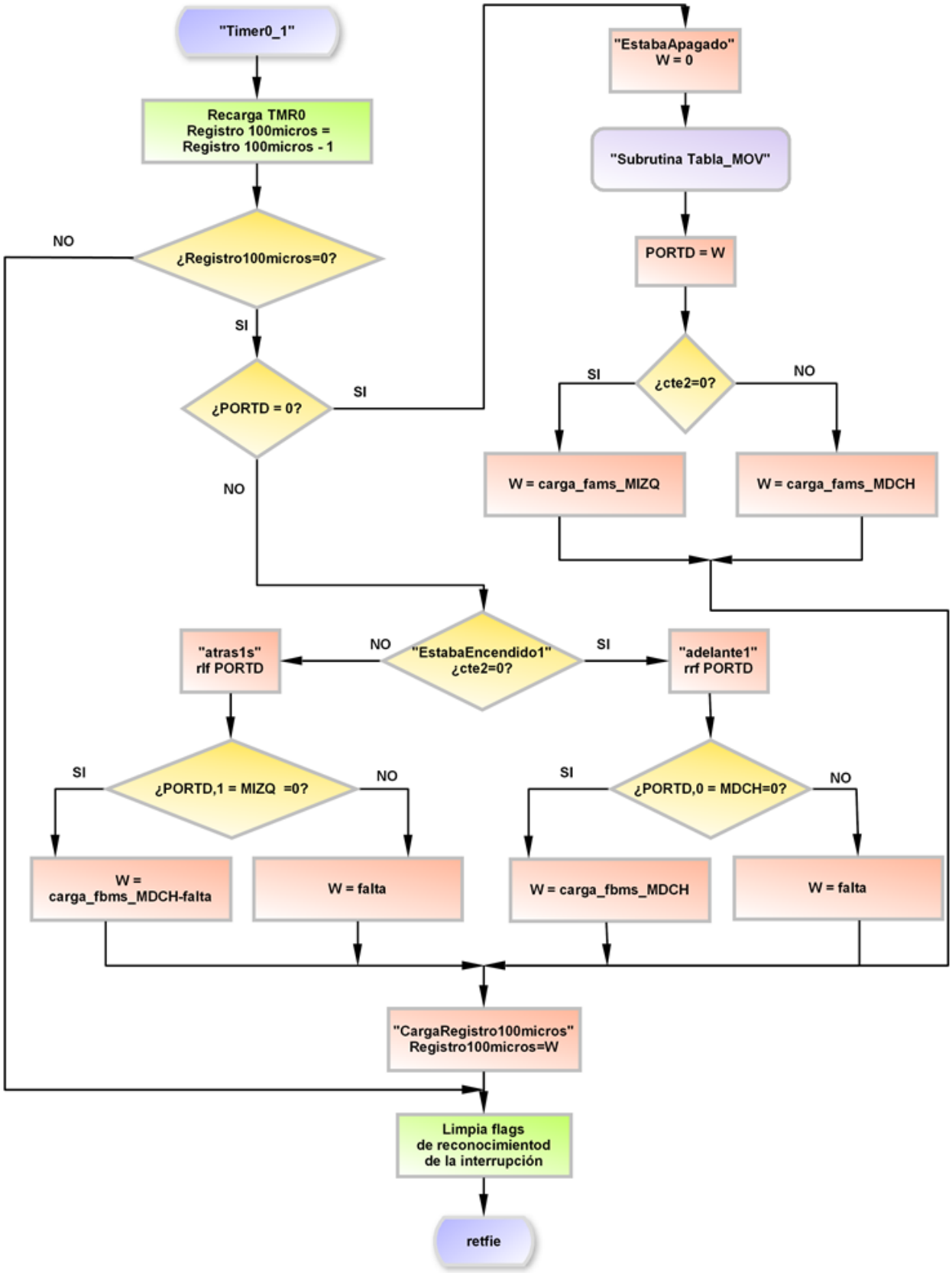


Figura 8. 13: Señales PWM para los servomotores

Si suponemos que el servo que tiene el sentido horario es el servomotor derecho y el otro el izquierdo el resultado sería que el micro robot tendría un movimiento rectilíneo hacia delante. Lo primero que se hace es cargar el valor de factor alto mas pequeño , en este caso el servo izquierdo, una vez terminado cargamos el valor que falta para completar el factor alto del servo derecho (restamos factor derecho menos el izquierdo para ese tiempo que falta) y a continuación cargamos el valor factor bajo más pequeño , en este caso el del servo derecho, para completar el período de 20ms.

3. Diagrama de flujo




"Timer0_1"
Recarga TMR0
Registro 100micros =
Registro 100micros - 1
¿Registro100micros=0?
NO
SI
¿PORTD = 0?
SI
NO
"EstabaApagado"
W = 0
"Subrutina Tabla_MOV"
PORTD = W
¿cte2=0?
SI
NO
W = carga_fams_MIZQ
W = carga_fams_MDCH
"EstabaEncendido1"
¿cte2=0?
NO
SI
"atras1s"
rlf PORTD
"adelante1"
rrf PORTD
¿PORTD,1 = MIZQ =0?
SI
NO
¿PORTD,0 = MDCH=0?
SI
NO
W =
carga_fbms_MDCH-falta
W = falta
W = carga_fbms_MDCH
W = falta
"CargaRegistro100micros"
Registro100micros=W
Limpia flags
de reconocimientod
de la interrupción
retfie

8.6. LIBRERIA PARA CONTROLAR MOTORES HITEC HSR 1422 CR CON UN PIC32

8.6.1. INTRODUCCIÓN

En este apartado se van a describir distintos aspectos a tener en cuenta a la hora de programar el PIC32 que no se han comentado en ninguno de los capítulos del proyecto.

En primer lugar se resaltaran las dos grandes diferencias con respecto el PIC 16F877A , que hacen que este microcontrolador sea idóneo para su uso en el campo de la micro robótica.

En segundo lugar se resaltará los aspectos técnicos a la hora de desarrollar la librería del micro robot MyBot en C, explicando cuales son las diferencias con la librería desarrollada para el PIC16F877A.

8.6.2. VARIABLES

En este apartado vamos a ver la importancia que tiene el usar distintas variables para la ejecución de los programas. A la hora de programar mediante el MPLAB C32, podemos elegir entre 10 tipos de datos enteros diferentes, char(8), short(16), int(32) long(32) y long long(64) con sus correspondientes variantes sin signo (“unsigned”), ver la siguiente tabla:

Tipo	Bits	Min	Max
char, signed char	8	-128	127
unsigned char	8	0	255
short, signed short	16	-32768	32767
unsigned short	16	0	65535
int, signed int, long, signed long	32	-2^{31}	$2^{31}-1$
unsigned int, unsigned long	32	0	$2^{32}-1$
long long, signed long long	64	-2^{63}	$2^{63}-1$
unsigned long long	64	0	$2^{64}-1$

Tabla 8. 5

Tal y como podemos observar en la tabla anterior, cuando el valor es con signo se va a dedicar un bit para evaluar el signo. También podemos comprobar que int y long son sinónimos pues ambos ocupan 32 bits (4 bytes).

Las operaciones con valores 8, 16 bits o 32 bits se procesan en el mismo tiempo por la ALU, sin embargo, cuando las definamos tipo long el tiempo en procesar una operación va a aumentar considerablemente así como la cantidad de RAM ocupada por cada uno de ellos.

En variables tipo long long, el código generado es más grande que para las variables tipo int, ya que para realizar la operación de multiplicación se requiere el uso de más instrucciones para llevarla a cabo. Esto es debido a que, la ALU puede ejecutar operaciones de 32bits de una sola vez. Sin embargo, para operaciones con datos de 64 bits, en realidad, estas se ejecutan como una secuencia de multiplicaciones y sumas de 32 bits.

Por otra parte, para el caso de una operación de división obtendríamos el mismo resultado. El código para las variables enteras de tipo char, short e int, sería el mismo, mientras que para la variable entera tipo long long, el espacio requerido aumenta considerablemente respecto a las anteriores, pues para realizar dicha operación es necesario llamar a la subrutina jal, la cual está dentro de la librería.

En cuanto a las variables fraccionales existen tres tipos:

Tipo	Bits
Float	32
Double	64
Long double	64

Tabla 8. 6

No existe ninguna diferencia entre estas dos últimas, sin embargo, hay que tener cuidado al declarar una variable float, ya que en el PIC32 no existe una unidad especial que trate los datos tipo float y por tanto tiene que ser compilada usando librerías aritméticas, aumentando el código a usar. Esto se traduce en más memoria necesaria para ejecutar el programa.

En la tabla 8.7 podemos ver de forma más detalla el análisis temporal realizado para una frecuencia de simulación de 64MHz. Las dos últimas columnas nos muestran la diferencia relativa respecto a la variable int o float. Como podemos observar, las operaciones hasta 32 bits de las variables enteras son hasta 3 veces más rápidas que cuando usamos el entero tipo long long. Evidentemente las operaciones que usan las variables tipo float requieren de más tiempo para ejecutarse, llegando a duplicar el tiempo necesario respecto a esta última para ejecutar una operación con formato long double. Por tanto podemos decir que la elección de las variables nos va a afectar tanto al tamaño del código del programa como a la velocidad de ejecución de este.

Tipo	Bits	Cycle Count	Tiempo (µs)	Respecto a int	Respecto a float
char	8	6	0.09375	1	-
Short	16	6	0.09375	1	-
Int, long	32	6	0.09375	1	-
Long long	64	21	0.328125	3.5	-
Float	32	51	0.796875	8.5	1
Long double	64	97	1.515625	16.5	2

Tabla 8. 7

Teniendo en cuenta todo lo expuesto , observamos como primera ventaja que a la hora de introducir el valor de la señal PWM podremos realizarlo en milisegundos sin tener que recurrir a la argucia utilizada en el subapartado 8.4.1 en adelante. Otra ventaja adicional es que podremos realizar operaciones aritméticas como la multiplicación y la división con mucha facilidad .

8.6.3. INTERRUPCIONES

Tal y como hemos estado observando a lo largo de este capítulo las interrupciones proporcionan un mecanismo para el control en tiempo real, permitiendo a las aplicaciones tratar con eventos externos asíncronos, las cuales requieren de una atención rápida por parte de la CPU.

El PIC32 proporciona 64 recursos distintos de interrupciones .De tal forma, que cada recurso de interrupción puede tener un trozo de código único, llamando a la rutina de servicio de interrupción (ISR) asociada a él, proporcionando de esta manera la respuesta requerida. Por tanto, las interrupciones pueden ser ejecutadas desde cualquier punto en un orden impredecible, cambiando la instrucción actual temporalmente para ejecutarse un procedimiento especial.

Las interrupciones tienen que ser capaces de salvar el contexto del procesador antes de tomar cualquier acción y cargarla después, tal y como estaba antes de que ocurriese la misma. Estas acciones las realiza el compilador MPLAB C32, sin embargo existen una serie de limitaciones que hay que tener en cuenta:

- Las funciones de servicio de la interrupción no devuelven ningún valor, son de tipo void.
- No se pueden pasar parámetros a la función.
- No pueden ser directamente llamadas por otras funciones.
- No deberían llamar a otras funciones, como recomendación para una mayor eficiencia del programa

Como hemos comentado, existen 64 recursos de interrupción los cuales a su vez pueden generar distintos eventos de interrupción, de tal forma que en total se dispone de 96 eventos diferentes que pueden ser controlados por el PIC32.

Evidentemente, cuando varios recursos de interrupción están habilitados, es necesario que la ISR identifique cual ha sido la interrupción que ha ocurrido para que se ejecute el trozo de código correcto. Para lo cual se van a utilizar varios flags en distintos registros.

Cada recurso de interrupción tiene 7 bits de control asociados, agrupados en varios registros especiales, estos son:

- Bit de habilitación de la interrupción, (“Interrupt Enable”,IE). Cuando está a 1, se podrá procesar la interrupción.

- Bit de interrupción (“Interrupt flag”,IF), se activa una vez ocurre el evento esperado por la interrupción. Debe ser puesto a 0 por el usuario antes de salir de la ISR.

- Nivel de prioridad (“Group Priority level”, IP). Existen 7 niveles de prioridad (desde ipl1 a ipl7, 3 bits) de tal forma que si dos interrupciones ocurren al mismo tiempo se ejecutará la que tenga un mayor nivel de prioridad.

- Nivel de subprioridad. Existen 4 niveles más de prioridad (2 bits) para un grupo de prioridad en concreto. De tal forma que si dos eventos ocurren simultáneamente y estos tienen la misma prioridad, el que tenga una subprioridad más alta será el seleccionado. En el caso de no configurar las prioridades existen unas por defecto (consultar el “*Data Sheet*”).

Para configurar y controlar las interrupciones vamos a usar las siguientes librerías, *plib.h* y *int.h.* Entre las principales funciones de estas librerías podemos encontrar:

- **INTEnableSystemSingleVectoredInt();** Esta función sigue una secuencia de inicialización del modulo de control de la interrupción para habilitar el manejo de las interrupciones del PIC32.

- **mXXSEetIntPriority(x);** Asigna un nivel de prioridad al recurso de interrupción seleccionado en XX.

- **mXXClearIntFlag();** permite poner a cero el flag IF, bandera que nos detecta que recurso de interrupción ha sido habilitado.

A la hora de programar hay que tener en cuenta dos cosas:

- Antes de habilitar la interrupción hay que declararla completamente (Ej: mT3IntEnable(1))

- La prioridad asignada a la interrupción debe coincidir con la sintaxis en la declaración de esta (Ej: mT3SetIntPriority(1))

En el caso de no realizar correctamente los dos puntos anteriores nuestro programa no funcionaría correctamente.

La prioridad decidirá cuál de las dos interrupciones será atendida primero si ocurren de forma simultánea. Sin embargo, cuando una de ellas se esté ejecutando, la otra tendrá que esperar a que esta termine para que se pueda ejecutar.

No obstante, puede darse el caso que se esté ejecutando una interrupción de un nivel de prioridad bajo, pero que una interrupción con una prioridad superior requiera una atención inmediata por parte del programa, interrumpiendo la ejecución de la primera interrupción. Una vez finalizada la ejecución de la interrupción con un nivel de prioridad más alto volverá a la interrupción de nivel más bajo para terminarla, esto es lo que se conoce como nesting

El mecanismo de funcionamiento del servicio de interrupción visto hasta el momento, es muy similar al desarrollado en la librería para el PIC16F877A. Hemos utilizado una función de interrupción para ejecutar una rutina de servicio de la interrupción en función de la que haya sido habilitada. Esto se conoce con el nombre de “Single mode vector”.

Sin embargo, el PIC32 ofrece la posibilidad de usar “vectored interrupts” y múltiples registros de interrupción, para dotar de una respuesta más rápida a las interrupciones y evitar sobrecargas.

En particular existen 64 vectores distintos de interrupción y 2 registros de 32 bits. Los 96 recursos de interrupciones que habíamos comentado que disponía la arquitectura del PIC32 están agrupados en los distintos vectores (consultar el “*Data Sheet*”).

Esta nueva posibilidad de asignar un vector separado para cada grupo de recursos de interrupción elimina la necesidad de testear secuencialmente todos los posibles recursos de interrupción para encontrar el que necesita ejecutarse. Es decir:

- Single mode vector: Todas las respuestas de interrupción son ejecutadas desde el mismo vector de interrupción, vector 0.
- Multi-vector mode: Las respuestas de interrupción son ejecutadas desde el vector de interrupción correspondiente.

Por tanto, el prologo de una interrupción (secuencia de comandos que hay que ejecutar antes de pasar a la interrupción propiamente), se ve reducida con esta forma de controlar las interrupciones. Además, podremos seguir usando el nesting para ejecutar las interrupciones de alta prioridad de una forma más rápida.

Para programar las interrupciones usando este nuevo modo, tenemos que tener en cuenta que cada interrupción tiene su propia función, en la cual tendremos que colocar el número del vector, este se puede consultar en el "Data Sheet".

Además, como ahora ya no hay que comprobar que interrupción ha sido la que ha incrementado su bandera, ya que se realiza implícitamente, cada función será llamada cuando se haya desbordado su bandera correspondiente. Y el último cambio es que para inicializar el "Multi-vector mode" tendremos que usar otra función de inicialización.

Por tanto teniendo en cuenta todo lo expuesto , observamos que el uso de "Multi-vector mode" lo hace altamente recomendable para el gobierno del micro robot "MyBot". Usando este método las ventajas son evidentes:

- No hace falta comprobar que interrupción ha sido la que ha incrementado su bandera.
- Se conseguirá una programación más flexible.
- Separación de funciones.
- Agregar prioridades.

8.6.4. MLIBRERIA PARA PIC32

Una vez vistas las ventajas que hacían al microcontrolador de 32 bits candidato futuro para su uso en el campo de la micro robotica, se procedió a su programación en C [29].La librería (ver anexo D) posee prácticamente la misma estructura que en la desarrollada para el microcontrolador de 8 bits . La filosofia de mantener un programa principal lo más sencillo posible con una serie de funciones que gobernaran el movimiento del micro robt sigue vigente.

En primer lugar hay que destacar que las macros GIRACENTRO, GIRAT y MUEVERECTO ahora son funciones tipo void y al igual que sus predecesoras se les tiene que pasar unos parámetros:

- **void GIRACENTRO (int VelSent1, int Tiempo1)**
- **void GIRAT (int Direc, int VelSent2, int Tiempo2)**
- **void MUEVERECTO (int VelSent3, int Tiempo3)**

Para no cambiar de nuevo toda la codificación narrada en los apartados anteriores, se les pasará los parámetros de la misma forma que en la librería desarrollada para el microcontrolador PIC16F877A.

La estrategia de control del servomotor sigue siendo la misma que en la librería anterior. Las diferencias son el uso de variables no enteras tipo float para poder realizar los cálculos de los parámetros de la función PWM y por supuesto el uso "Multi-vector mode".

Como su predecesora posee una función llamada PWM encargada de calcular los parámetros de la función y habilitar las interrupciones para generarla . Esta función se la llama cuando se hace uso de las funciones GIRACENTRO Y GIRAT. La diferencia fundamental con la subrutina desarrollada para el PIC16F877A es la asignación a del servicio de interrupción de Timer2 usando la filosofía "Multi-vector mode". Le hemos asignado una prioridad de 1.

Usando este nuevo modo, tenemos que tener en cuenta que cada interrupción tiene su propia función, por tanto para el timer 2 usaremos " _TIMER2_VECTOR" según el "DataSheet" . Esta función se activará cuando se haya desbordado el temporizador. La función la encontraremos bajo el nombre de :

void __ISR(_TIMER_2_VECTOR, ipl1) T2InterruptHandler(void)

Y esta función sustituirá a la subrutina Timer0, que es rediirgira desde la posición ORG4, de la librería desarrollada para el PIC16F877A.

Desde función MUEVERECTO se llama a una subrutina PWM2, encargada también de calcular los parámetros de la función y habilitar las interrupciones para generarla. . La diferencia fundamental con la subrutina de la librería para el microcontrolador de 8 bits es la asignación a del servicio de interrupción de Timer4 usando la filosofía "Multi-vector mode". Le hemos asignado una prioridad de 1.

Para el timer 4 usaremos " _TIMER4_VECTOR" según el "DataSheet". Esta función se activará cuando se haya desbordado el temporizador. La función la encontraremos bajo el nombre de :

void __ISR(_TIMER_4_VECTOR, ipl1) T4InterruptHandler(void)

Y esta función sustituirá a la subrutina Timer0_1, que también es redirigira desde la posición ORG4, de la librería desarrollada para el PIC16F877A

Por último en las funciones GIRACENTRO, GIRAT y MUEVERECTO como en su predecesora se llama a una subrutina denominada TEMPORIZADOR cuyo objetivo no es otro que el gestionar el tiempo que está en movimiento el micro robot. En caso de que haya cumplido el tiempo estipulado deshabilitara todas las interrupciones y reseteará las variables utilizadas.

Al igual que las otras subrutinas antes mencionadas se le ha asginado servicio de interrupción de Timer3 usando la filosofia “Multi-vector mode”.Le hemos asignado una prioridad de 7 dado que si finaliza el tiempo debe parar obligatoriamente.

Para el timer3 usaremos ” _TIMER3_VECTOR” según el “DataSheet” . Esta función se activará cuando se haya desbordado el temporizador. La función la encontraremos bajo el nombre de :

void __ISR(_TIMER_3_VECTOR, ipl7) T3InterruptHandler(void)

Y esta función presenta una grandísima ventaja con respecto su predecesora, ya que no tiene que permanecer continuamente testeando si se ha desbordado o no el temporizador. Al asignarle una interrupción , cosa que en la anterior librería no se hizo dado que complicaba mas el código, podemos gestionar ese tiempo útil para otros objetivos.

Debido a esta ventaja las funciones principales GIRACENTRO, GIRAT y MUEVERECTO después de configurar el PWM y Temporizador se hace uso de una variable denominada ocupado para indicar que se están usando las interrupciones pertinentes. Esta variable se usa en el programa principal main en un bucle while para que ,mientras el robot este en movimiento , se pueda tomar datos, comprobar sensores , hacer cálculos etc. Aportando una nueva ventaja más a la librería.

CAPÍTULO 9 SOFTWARE DESARROLLADO PARA EL CONTROL DE MYBOT MEDIANTE SENSORES

CAPÍTULO 9. SOFTWARE DESARROLLADO PARA EL CONTROL DE MYBOT MEDIANTE SENSORES

9.1. Introducción

En este capítulo detallaremos a continuación las aplicaciones desarrolladas para el micro robot utilizando la placa de control OLIMEX PIC-P40 con un PIC16F877A. Por tanto para finalizar con el desarrollo del mismo , nos centraremos en la zona de control de la Torrebot, descrita en el capítulo 6. Se detallará en cada apartado la forma de instalar las placas al micro robot.

9.2. LDR

9.2.1. INSTALACIÓN DE LA PLACA

En primer lugar antes de realizar un programa hay que establecer la estrategia que debe seguir el micro robot, es decir que comportamiento queremos que tenga . Nuestro micro robot “MyBot” ,en este caso , alterará su recorrido en función de la señal que el sensor LDR le suministre. Como se observa en la figura siguiente la placa desarrollada en el capítulo 7 se coloca en el piso superior para que pueda incidir la luz sobre ella.

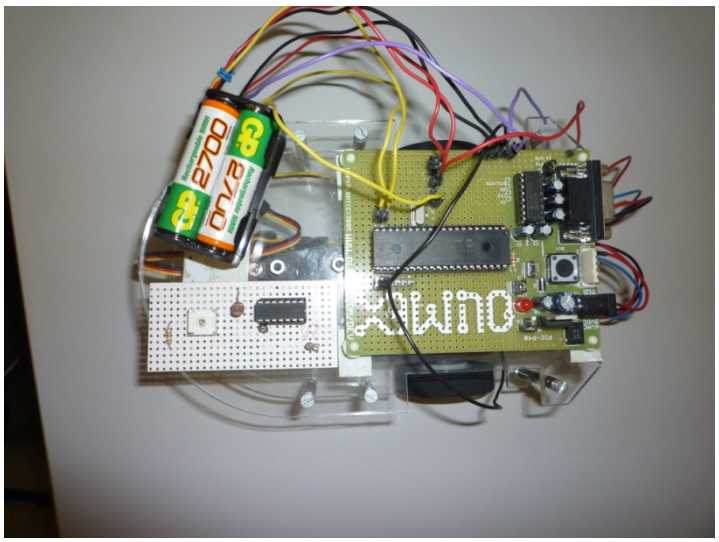


Figura 9. 1: Situación de la placa en el micro robot

La instalación de la placa es extremadamente sencilla. En primer lugar conectaremos los cables como se muestra en la tabla siguiente , teniendo en cuenta la figura 9.2.

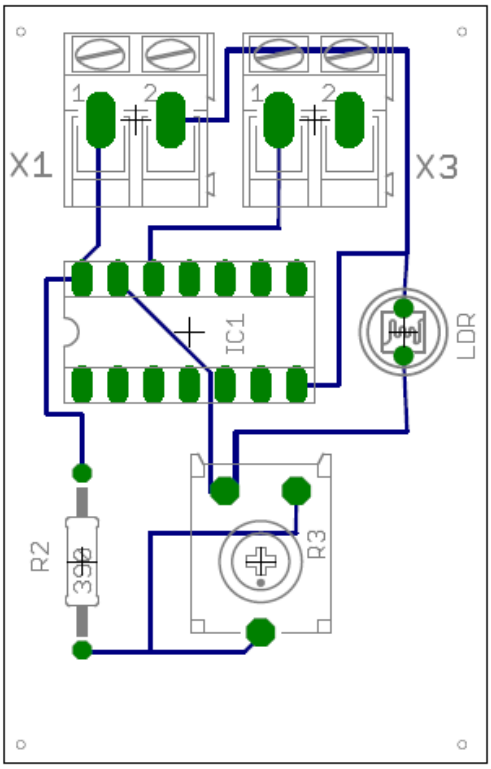


Figura 9. 2 : **Placa LDR**

SIMBOLO	VALOR
X1-1	V+
X1-2	V-
X3-1	Salida sensor LDR

Tabla 9. 1

9.2.2. PRUEBA

Como se puede comprobar las conexiones son extremadamente sencillas pero por si acaso generamos un programa de prueba para observar si esta todo correcto. Vamos a utilizar la librería desarrollada en el capítulo 8 para el PIC16F877A. Las salidas de la placa LDR se conectarán en el siguiente puerto del microcontrolador PIC16F877A que se encuentra en la placa de control del micro robot OLIMEX PIC-P40:

SALIDA PLACA LDR	VALOR	PUERTO PIC16F877A
X3-1	Salida en digital del sensor LDR	PORTD,2

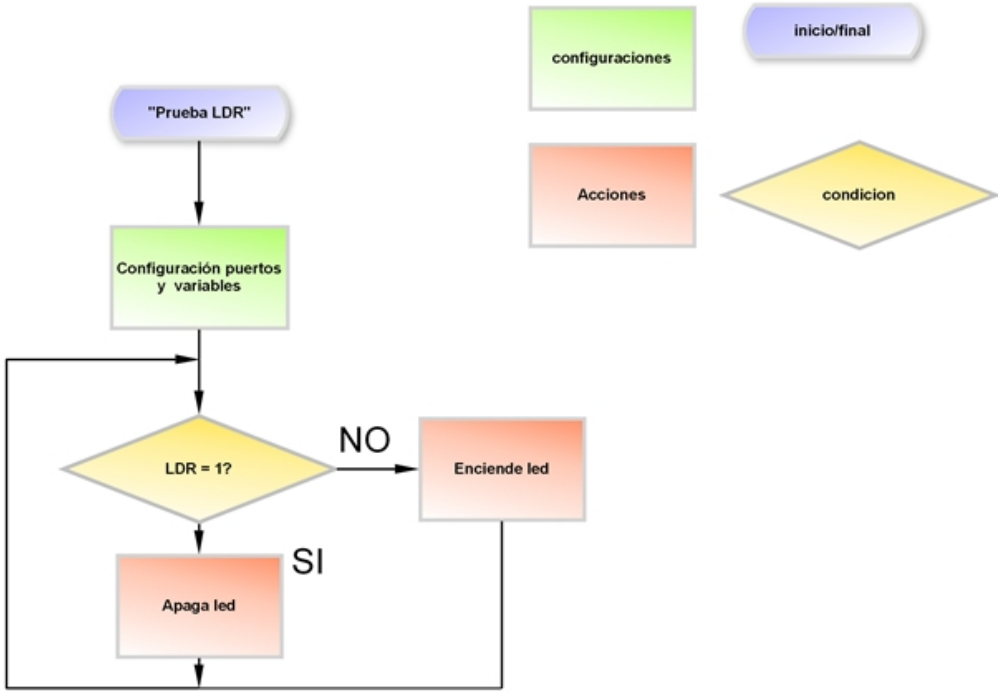
Tabla 9. 2

1. **Nombre del fichero : prueba.asm**

2. **Descripción :**

El objetivo del programa es comprobar la funcionalidad de la placa LDR desarrollada. Para ello se procede a encender un led que posee la placa de control OLIMEX PIC-P40 cuando no incide luz sobre el sensor LDR. Se incorpora además la librería desarrollada en el capítulo 8 para el PIC16F877A para comprobar que no existe incompatibilidad alguna . La variable LDR se declara en la misma librería

3. **Diagrama de flujo**



4. <u>Código:</u>

```
;////////////////////////////////////////////////////////////////////////////
;//                                                                        //
;//                              prueba.asm                                //
;//                                                                        //
;//                    PFC DIEGO P. GONZALEZ GONZALEZ                      //
;//                                                                        //
;////////////////////////////////////////////////////////////////////////////

; ZONA DE DATOS
*********************************************************************

      __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _HS_OSC
      LIST     P=16F877A
      INCLUDE  <P16F877A.INC>


      CBLOCK 0x40
    ENDC


; ZONA DE CÓDIGOS
*********************************************************************

      ORG   0
      goto  Inicio

                                          ; salvamos vector interrupcion
INCLUDE<Mlibreria.INC>
Inicio
call INI

Principal

   btfsc LDR ;LDR=0 no hay luz // LDR =1 Hay luz
   goto Apaga
   goto Enciende

Apaga
   bcf LED ; Apagamos el Led de la placa
   goto Principal

Enciende
   bsf LED ; Encendemos el Led de la placa
   goto Principal



;Resetear para empezar


;INCLUDE<CMOTORES.INC> ERROR
END
```

9.2.3. ALGORITMO FINAL

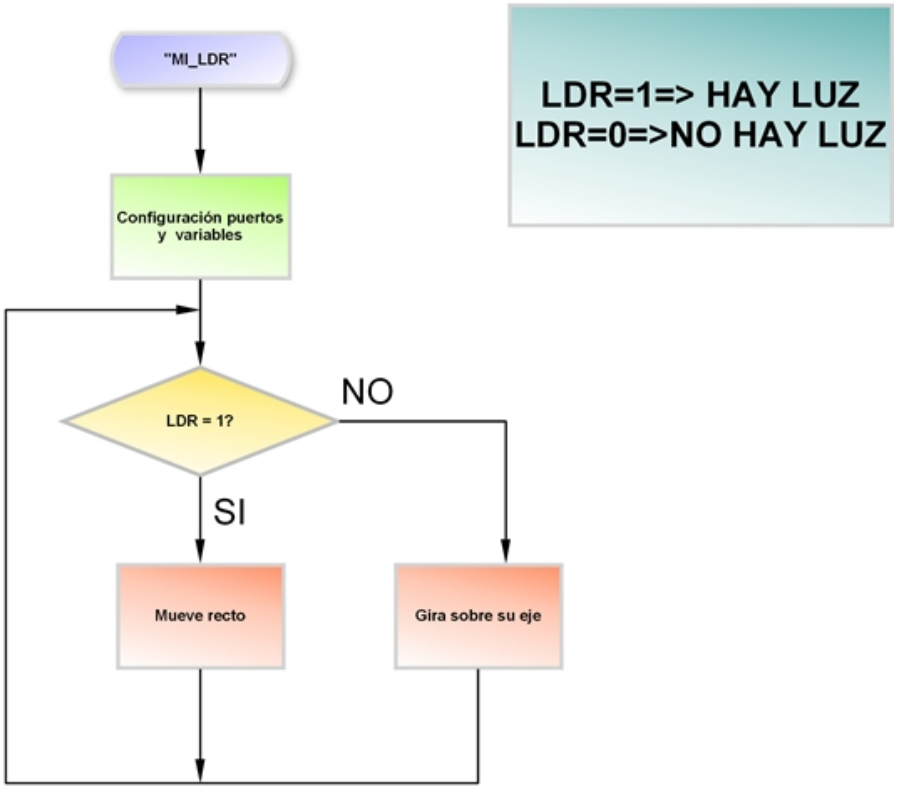
El algoritmo que se propone es de una sencillez extrema. Y sirve para resaltar el objetivo cumplido de mantener el programa principal lo mas sencillo posible para el control del micro robot.

1. **Nombre del fichero : LDR.asm**

2. **Descripción :**

Posee la misma estructura que el programa anterior. La particularidad des este programa es la utilización de las macros para controlar el micro robot. Así pues si se incide luz sobre el sensor, el micro robot seguirá recto. Pero cuando aplicamos una sombra el robot empieza a girar sobre sí mismo.

3. **Diagrama de flujo**



4. Código:

```
;////////////////////////////////////////////////////////////////////////////////
;//                                                                            //
;//                              LDR.asm                                     //
;//                                                                            //
;//                    PFC DIEGO P. GONZALEZ GONZALEZ                          //
;//                                                                            //
;////////////////////////////////////////////////////////////////////////////////

; ZONA DE DATOS
*********************************************************************

      __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _HS_OSC
      LIST     P=16F877A
      INCLUDE  <P16F877A.INC>


      CBLOCK 0x40
    ENDC


; ZONA DE CÓDIGOS
*********************************************************************

      ORG   0
      goto  Inicio

                                          ; salvamos vector interrupcion
    INCLUDE<Mlibreria.INC>
    Inicio
    call INI

    Principal
    bsf LED

    btfss LDR
    goto Giro
    goto Sigo_recto

    Giro

    GIRACENTRO .0,.20;GIRA ANTIHORARIO, MAX VELOCIDAD Y 2 S
    bcf LED
    goto Principal


    Sigo_recto

    MueveRectoT  .0, .20;MUEVETE RECTO ,SENTIDO DELANTE, MAX VELOCIDAD Y 2S
    bcf LED
    goto Principal



;INCLUDE<CMOTORES.INC> ERROR
END
```

9.3. Cny70

9.3.1. INSTALACIÓN DE LA PLACA

Antes de realizar un programa hay que establecer la estrategia que debe seguir el micro robot, ya se para un comportamiento como un robot rastreador o para cualquier otra función. Nuestro micro robot "MyBot" realizará una función de rastreador mediante los sensores CNY70 instalados como se muestra en la figura 9.3.Como se observa se han colocado unos separadores para aproximarlos lo máximo posible al suelo. Hay que destacar que la sensibilidad de estos sensores se ve reducida enormemente cuando se alejan lo suficiente de la superficie a rastrear. Normalmente se colocan a una distancia de 2 o 3 mm por encima del suelo.

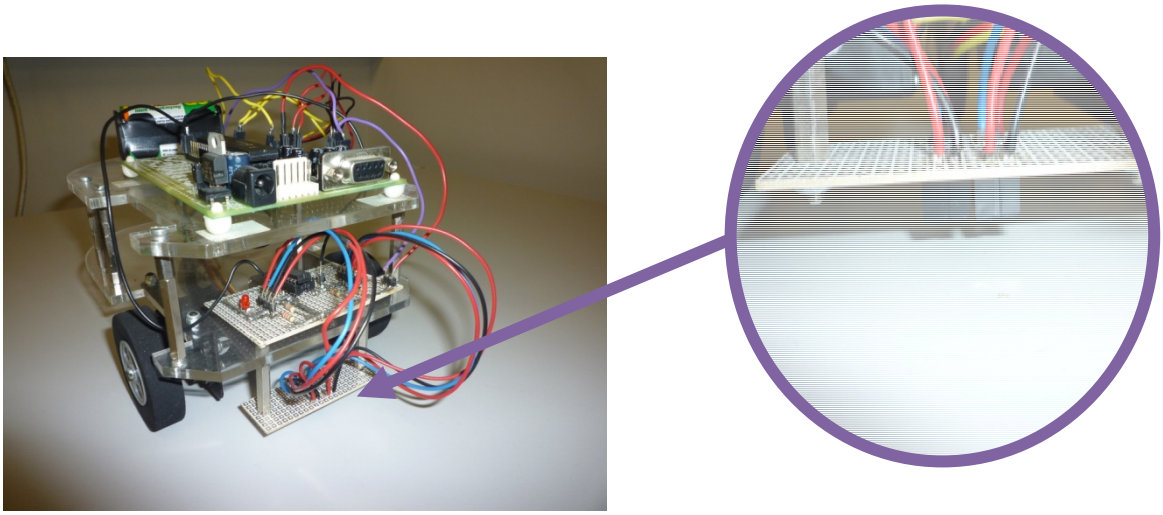


Figura 9. 3

Se han usado una serie de combinación de colores en los cables para saber que pin corresponde al colector,emisor, anodo y catodo:

PIN	Cable
Colector	**Rojo-Negro**
Emisor	**Azul**
Ánodo	**Rojo**
Cátodo	**Negro**

Tabla 9. 3

Se conectarán estos cables a la placa desarrollada en el capítulo 7(ver figura 9.4). Como se citó en dicho capítulo en los bornes con los símbolos X2 y X4

se conectarán los sensores CNY70 izquierdo y derecho respectivamente. Teniendo en cuenta la siguiente tabla:

SIMBOLO	VALOR	CABLE
X2-1	Ánodo del sensor izquierdo	**Rojo**
X2-2	Cátodo del sensor izquierdo	**Negro**
X2-3	Emisor del sensor izquierdo	**Azul**
X2-4	Colector del sensor izquierdo	**Rojo-Negro**
X4-1	Colector del sensor derecho	**Rojo-Negro**
X4-2	Emisor del sensor derecho	**Azul**
X4-3	Cátodo del sensor derecho	**Negro**
X4-4	Anodo del sensor derecho	**Rojo**

Tabla 9. 4

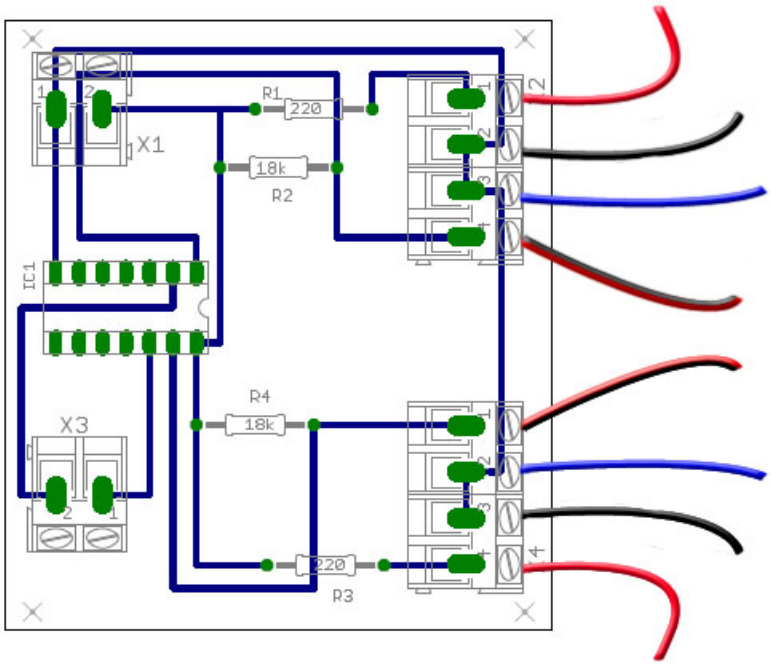


Figura 9. 4 : Placa CNY70

En el borne X1 suministraremos la corriente continua de 5V. Teniendo en cuenta que en :

SIMBOLO	VALOR
X1-1	V-
X1-2	V+

Tabla 9. 5

Y para finalizar conectaremos al borne X3 los cables para obtener la salida de los sensores CNY70 en digital:

SIMBOLO	VALOR
X3-1	Salida en digital del derecho (X4)
X3-2	Salida en digital del izquierdo(X2)

Tabla 9. 6

9.3.2. PRUEBA

Una vez realizadas las conexiones pertinentes es hora de probar si funciona nuestra placa y si hemos hecho bien las conexiones. Para ello se ha realizado un algoritmo extremadamente sencillo. Vamos a utilizar la librería desarrollada en el capítulo 8 para el PIC16F877A. Las salidas de la placa CNY70 se conectarán en los siguientes puertos del microcontrolador PIC16F877A que se encuentra en la placa de control del micro robot OLIMEX PIC-P40:

SALIDA PLACA CNY70	VALOR	PUERTO PIC16F877A
X3-1	Salida en digital del sensor derecho (X4)	PORTD,3
X3-2	Salida en digital del sensor izquierdo(X2)	PORTD,2

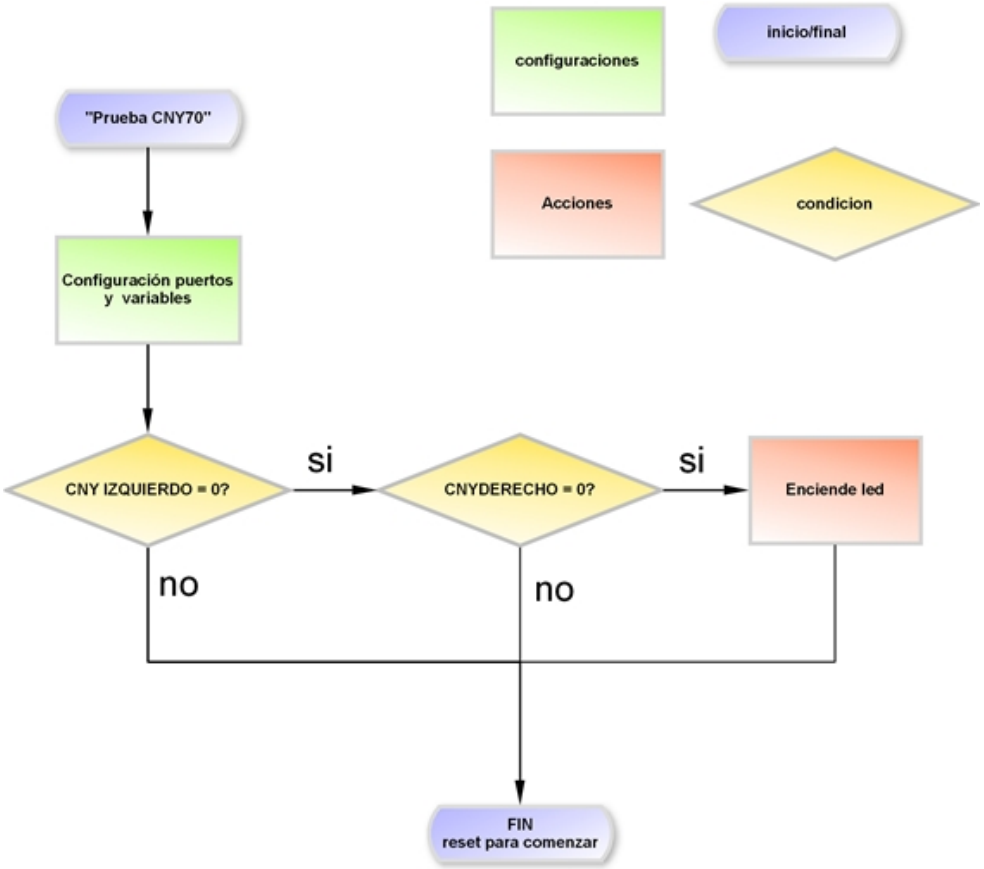
Tabla 9. 7

1. **Nombre del fichero : prueba.asm**

2. **Descripción :**

Como en el caso anterior el objetivo del programa es comprobar la funcionalidad de la placa CNY70 desarrollada. Para ello se procede a encender un led que posee la placa de control OLIMEX PIC-P40 cuando se detecta negro con ambos sensores (bastaría con taparlos). Se incorpora además la librería desarrollada en el capítulo 8 para el PIC16F877A para comprobar que no existe incompatibilidad alguna . Las variables CNY70 izquierdo y derecho se declara en la misma librería. Para volver a comenzar con el programa basta con pulsar el botón reset.

3. **Diagrama de flujo**



4. <u>Código:</u>

```
;////////////////////////////////////////////////////////////////////////////////
;//                                                                            //
;//                              prueba2.asm
//
;//                                                                            //
;//                    PFC DIEGO P. GONZALEZ GONZALEZ                          //
;//                                                                            //
;////////////////////////////////////////////////////////////////////////////////

; ZONA DE DATOS
*********************************************************************

      __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _HS_OSC
      LIST     P=16F877A
      INCLUDE  <P16F877A.INC>


      CBLOCK 0x40
    ENDC


; ZONA DE CÓDIGOS
*********************************************************************

      ORG   0
      goto  Inicio

                                          ; salvamos vector interrupcion
INCLUDE<Mlibreria.INC>
Inicio
call INI

Principal

   bcf LED
   btfsc cnyiz
   goto $
   btfss cnydc
   bsf LED
   goto $

;Resetear para empezar


;INCLUDE<CMOTORES.INC> ERROR
END
```

En principio parece razonable pensar en cualquiera de las siguientes estrategias a la hora de diseñar un algoritmo capaz de seguir una línea:

Algoritmo para seguir el centro de línea negra. Dependiendo de la posición de los sensores podemos hacer que el micro robot tome las decisiones mostradas en la figura 9.5. En esta figura los cuadrados de la derecha e izquierda representan la lectura del sensor correspondiente: si es blanco indica que detecta fondo blanco y negro que se encuentra encima de la línea negra.

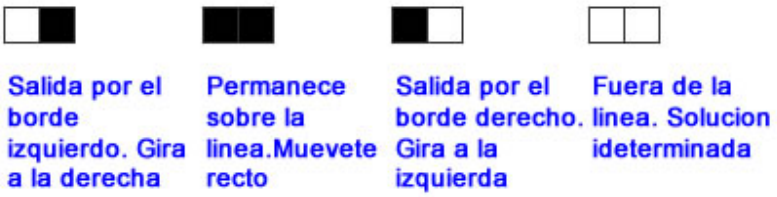


Figura 9. 5: Algoritmo incorrecto

Si observamos este algoritmo con detenimiento podemos comprobar que no cumple realmente el objetivo que se le ha asignado. Esto es así debido a que depende directamente de la anchura de la pista y por tanto producirá una imprecisión suficientemente grande a la hora de seguimiento de la misma, ocasionando retrasos en el recorrido , cabeceos no deseados o incluso la desviación total de la pista.

Algoritmo para seguir el borde de la línea negra. En este caso se soluciona el problema anterior antes descrito. En primer lugar tendremos que seleccionar uno de los bordes de la línea. En nuestro caso elegiremos el derecho, es decir, la detección de negro-blanco respectivamente por los sensores colocados a la derecha y a la izquierda tal y como se muestra en la figura9.6.

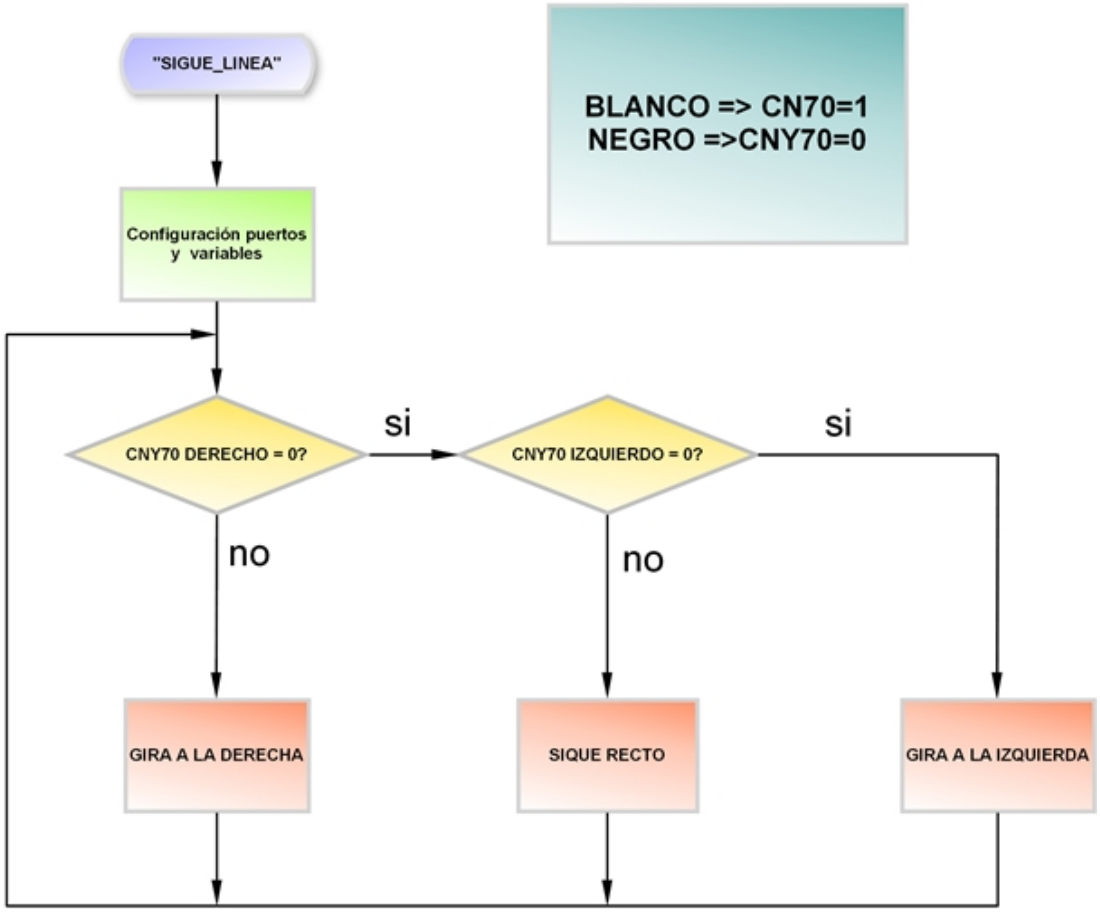
Figura 9. 6: Algoritmo correcto

1. **<u>Nombre del fichero</u> : CHL.asm**

2. **<u>Descripción</u> :**

Sigue el recorrido de una línea utilizando el último algoritmo descrito hay que tener en cuenta que se ha seleccionado el borde derecho para el control. Por tanto inicialmente el sensor izquierdo debe detectar negro y el derecho debe detectar blanco.

3. **<u>Diagrama de flujo</u>**



4. <u>Código:</u>

```
;////////////////////////////////////////////////////////////////////////////////
;//                                                                            //
;//                              CHL.asm                                     //
;//                                                                            //
;//                    PFC DIEGO P. GONZALEZ GONZALEZ                          //
;//                                                                            //
;////////////////////////////////////////////////////////////////////////////////

; ZONA DE DATOS
*********************************************************************

      __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _HS_OSC
      LIST    P=16F877A
      INCLUDE  <P16F877A.INC>


      CBLOCK 0x40
    ENDC


; ZONA DE CÓDIGOS
*********************************************************************

      ORG  0
      goto  Inicio

                                        ; salvamos vector interrupcion
    INCLUDE<Mlibreria.INC>
    Inicio
    call INI

    Principal
    bsf LED

    btfss cnydc
    goto Giro_a_la_derecha
    btfss cnyiz
    goto Sigo_recto

    Giro_a_la_izquierda

    GiraT .2,.0,.5;GIRA IZQ ,SENTIDO DELANTE , MAX VELOCIDAD Y 500 m
    bcf LED
    goto Principal

    Giro_a_la_derecha

    GiraT .1,.4,.5;GIRA DHA ,SENTIDO DELANTE , MAX VELOCIDAD Y 500 ms
    bcf LED
    goto Principal
```

```
    Sigo_recto

    MueveRectoT  .0, .5;MUEVETE RECTO ,SENTIDO DELANTE, MAX
VELOCIDAD Y 500MS
    bcf LED
    goto Principal



;INCLUDE<CMOTORES.INC> ERROR
END
```

CAPÍTULO 10
CONCLUSIONES

CAPÍTULO 10. CONCLUSIONES Y TRABAJOS FUTUROS

10.1. Introducción

El presente Proyecto Fin de Carrera se ha centrado en el desarrollo de aplicaciones prácticas orientadas en el campo de la micro robótica mediante la programación de microcontroladores PIC de gama baja y alta.

Este proyecto presenta los equipos, herramientas de Microchip, programas y documentos necesarios para el diseño, construcción y programación de un micro robot. Además, hemos resaltado la viabilidad de realizar todo el control del micro robot con un microcontrolador.

Por lo expresado anteriormente la elaboración del presente proyecto sirve para comprender un poco más la teoría de los microcontroladores y su aplicación en el campo de la micro robótica .

Como trabajo futuro o continuación del presente proyecto y tras el resultado bastante satisfactorio de las pruebas realizadas, se abre la puerta para la realización de aplicaciones más sofisticadas.

En concreto se ha pensado en el desarrollo de un mando inalámbrico capaz de detectar los movimientos del sujeto que lo controla y enviar dicha información codificada ,ya sea por radiofrecuencia o por infrarrojos, al micro robot. Se recomienda para esta aplicación el uso del micro controlador PIC32MX460F512L para el control del micro robot dado que es un candidato esplendido para esta aplicación. Los movimientos se recogerían mediante el acelerómetro descrito en el capítulo 7. (figura 10.1)

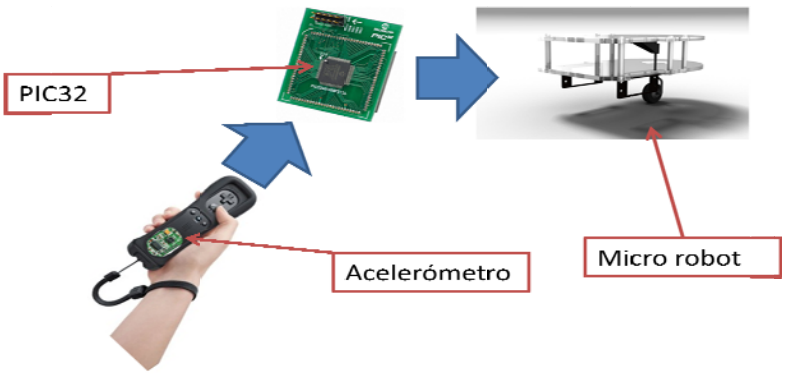


Figura 10. 1

Otra aplicación interesante es el control del micro robot mediante el PC por puerto COM a una placa que envíe y transmita las señales , por infrarrojos o por radiofrecuencia, al micro robot. Se adjunta en el CD del presente proyecto diversas pruebas realizadas con la EASY PIC para enviar información por dicho protocolo. (figura 10.2)

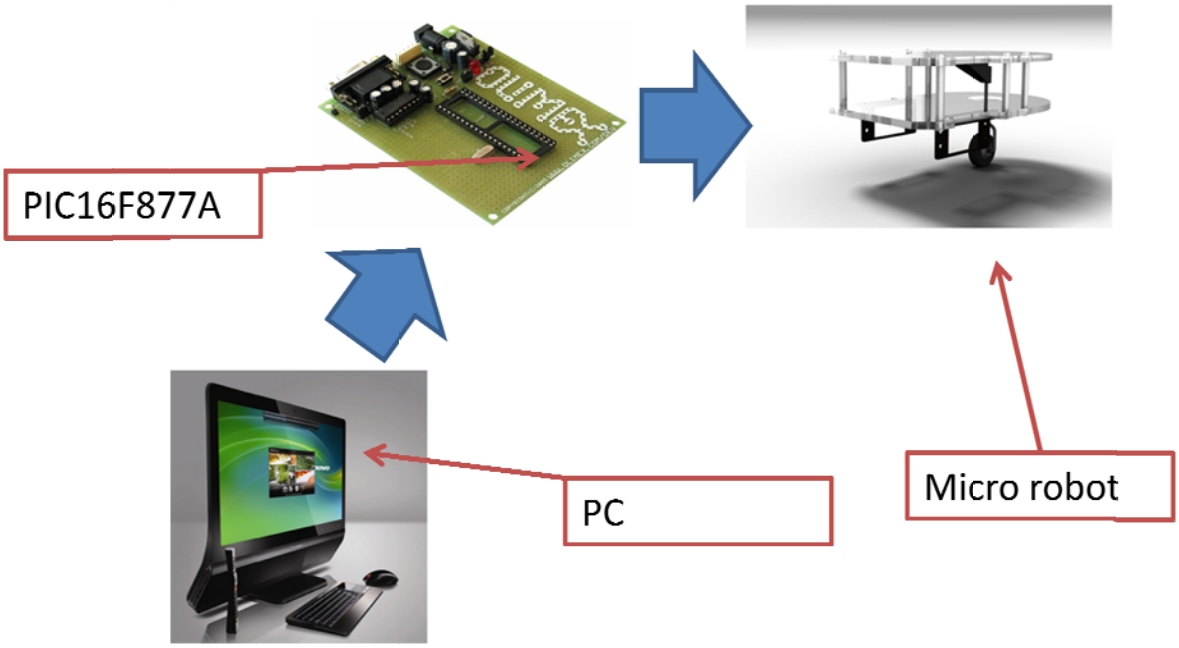


Figura 10. 2

Por último , utilizando la placa descrita en el párrafo anterior, sería interesante mandar la posición , velocidad y aceleración del micro robot usando el acelerómetro al PC y desde él mostrar el resultado en gráficas.(figura 10.3). Se adjunta algoritmo desarrollado por la compañía freescale mediante el uso de otro microcontrolador

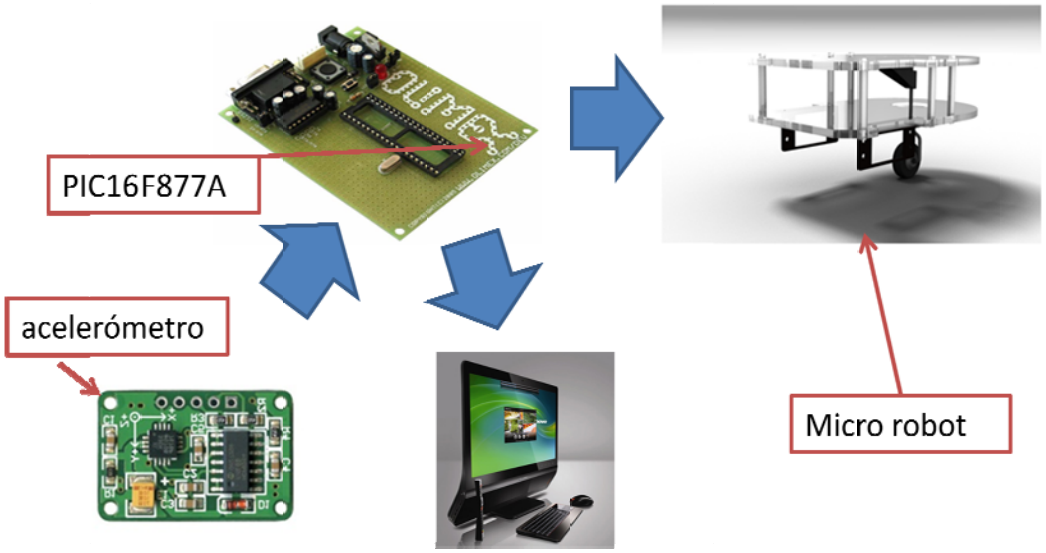


Figura 10. 3

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

ANEXO A

ANEXO A. CONFIGURACIÓN CONVERSOR AD EN PIC16F877A

A.1. Conversor Analógico Digital

A.1.1. Introducción

El microcontrolador PIC16F877A de Microchip puede desempeñar muchas funciones pero en la que nos vamos a centrar en este subapartado es la de su conversor analógico-digital.

Antes de meternos de pleno en su funcionamiento vamos a comentar los conceptos básicos de una conversión de una señal analógica a digital.

Tanto nuestra voz como muchas de las señales que se envían a través de un medio guiado como un cable o no guiado como es el aire son de tipo continuo y pueden tomar infinitos valores a lo largo del tiempo. Por ejemplo podemos decir que la señal eléctrica que se transmite de la tarjeta de sonido al altavoz es continua y puede tomar cualquier tensión entre los dos hilo

El interés en digitalizar una señal puede surgir por varios motivos: el hecho de querer almacenarla en un soporte digital o transmitirla digitalmente para poder reconstruirla, poder tratar con programas los valores analógicos que dé un sensor, etc.

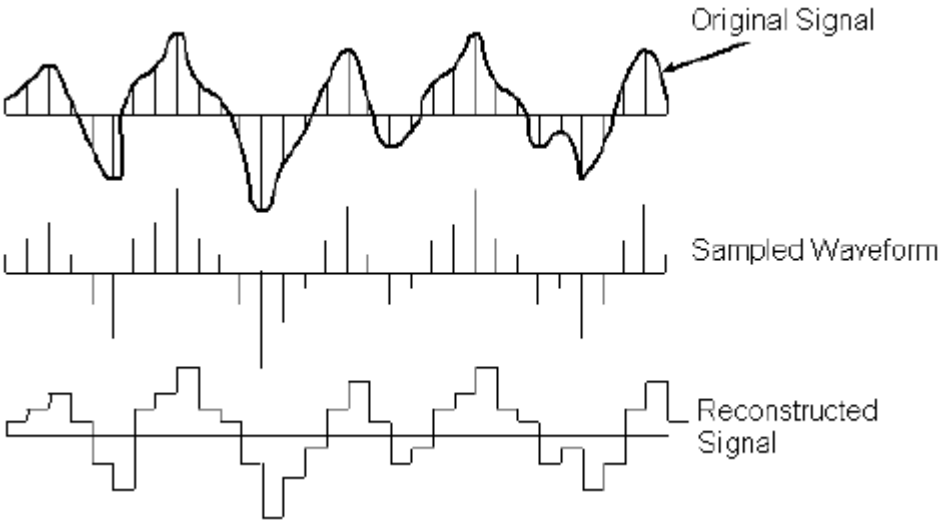


Figura A. 1

Acotando la señal en un intervalo de tiempo y unos valores mínimos y máximos de tensión tenemos que tener en cuenta dos factores fundamentales a la hora de almacenar dicha señal en un formato adecuado que pueda almacenarse digitalmente (con ceros y unos): se tiene que **muestrear** y **cuantificar.**

El **muestreo** implica que tenemos que coger una muestra de la señal cada T segundos ya que no hay memoria suficiente capaz de almacenar los infinitos puntos de una señal en un intervalo cualquiera de tiempo. En el ejemplo de la figura A.2 se puede observar como se muestrea una señal; se ha acotado un segundo de tiempo y 5 V de tensión de entrada analógica del PIC

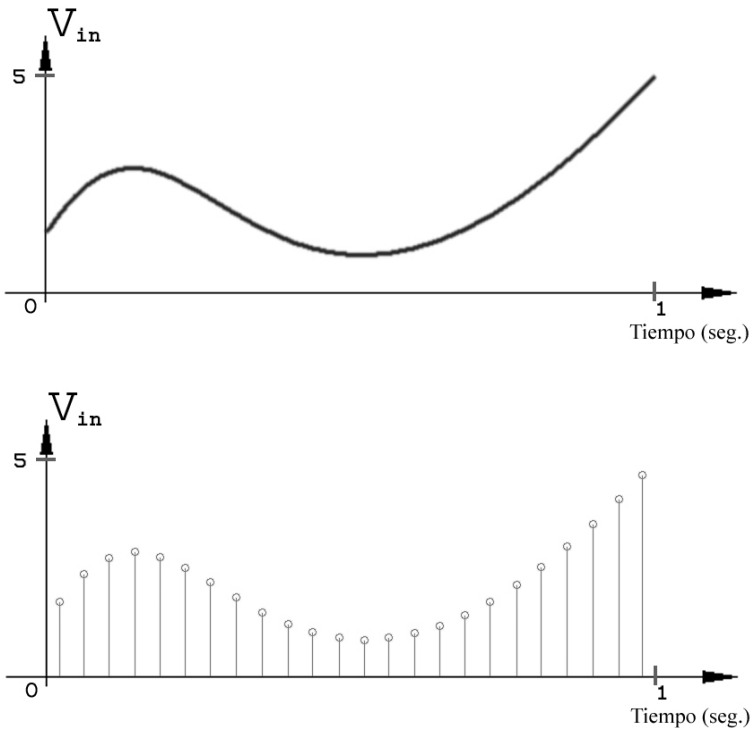


Figura A. 2

La **cuantificación** surge por el mismo motivo que el muestreo pero para el eje de ordenadas: una vez tenemos una muestra su amplitud puede tomar infinitos valores, debemos redondear entre unos valores fijos a lo largo de ese eje. Estos valores van a depender del número de bits que vayamos a almacenar para cada muestra, por ejemplo, en la imagen se cogen 4 bits y con ellos se pueden formar 16 combinaciones y por lo tanto 16 distintos niveles en los que se puede dividir el eje. El PIC cuantifica con 10 bits luego son 1023 niveles.

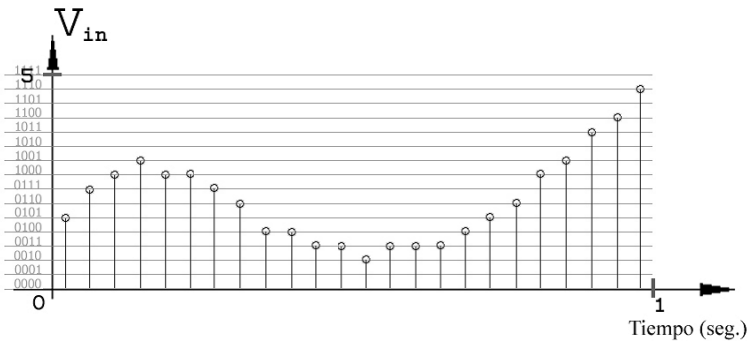


Figura A. 3

A.3.1. Error en la cuantificación

Como se puede observar tanto en la cuantificación como en el muestreo se produce una pérdida de la forma de la señal original y eso implica que se va a cometer un error de aproximación cuya magnitud se puede calcular .

A.1.2. Error cometido en el muestreo

El parámetro fundamental del muestreo digital es el intervalo de muestreo ▲seg., o su equivalente frecuencia de muestreo 1/▲ Hz. Lógicamente, cuanto menor sea ▲, mayor número de valores obtendremos de la señal, y viceversa. El resultado de dicho muestreo es la obtención de una serie discreta ordenada {xr} = {x0, x1, x3,..., xr,...}, en la que el índice r indica la posición de orden temporal del valor xr. Así, el valor de la señal original, en el tiempo t = ▲r, x (t), se representa por xr.

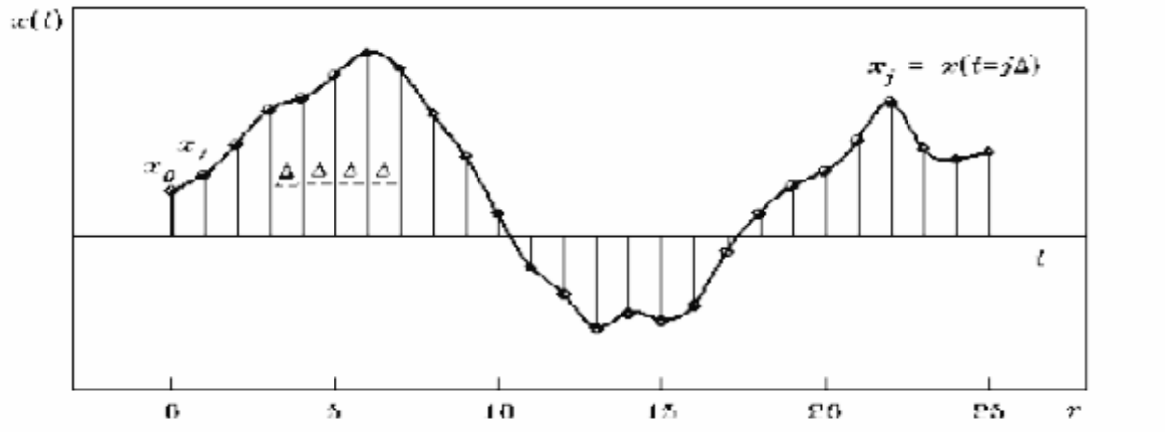


Figura A. 4

Si aumentamos el número de muestras por unidad de tiempo, la señal muestreada se parecerá más a la señal continua. Respecto a esto, el criterio de Nyquist asegura que para que la señal muestreada contenga la misma información que la continua, la separación mínima entre dos instantes de muestreo debe ser 1/(2 W), siendo W el ancho de banda de la señal. Dicho de otra forma, que la frecuencia de muestreo debe ser mayor o igual que 2 W.

A.1.3. Error cometido en la cuantificación

Se conoce como error de cuantificación (o *ruido,eq(n)*), a la diferencia entre la señal de entrada(sin cuantificar, x(n)) y la señal de salida (ya cuantificada,xq(n)), interesa que el ruido sea lo más bajo posible.

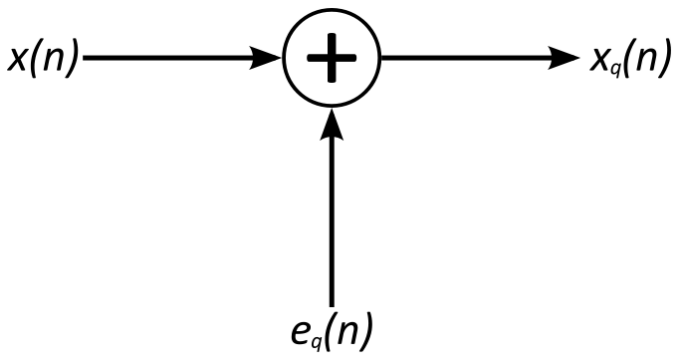


Figura A. 5

de modo que se cumple: $e_q(n) = x_q(n) - x(n)$

En la *Figura A.5* es posible verificar que el error de cuantificación $e_q(n)$ está siempre en el rango -Δ/2 a Δ/2 mientras la señal analógica de entrada se encuentre dentro del rango del cuantificador:

$$-\frac{\Delta}{2} < e_q(n) < \frac{\Delta}{2}$$

donde Δ es el tamaño del escalón de cuantificación que viene dado por: $\Delta = \frac{R}{L}$

donde R es el rango del cuantificador y L el número de niveles de cuantificación.

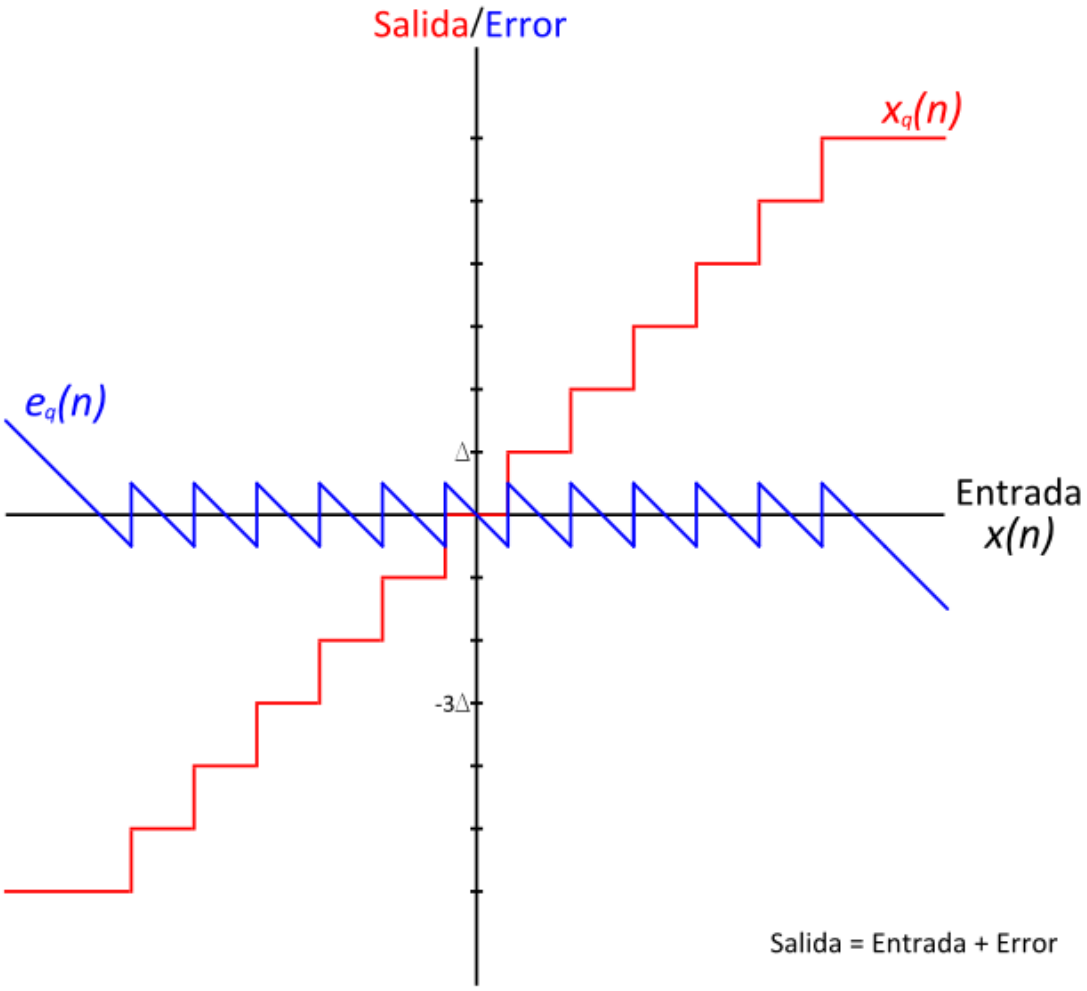


Figura A. 6

Para minimizar los efectos negativos del error de cuantificación, se utilizan las distintas técnicas de cuantificación que a continucación se describen:

La **cuantificación uniforme** o **lineal** es el proceso de **cuantificación** más simple. Se utiliza un bit rate constante A cada muestra se le asigna el valor inferior más próximo, independientemente de lo que ocurra con las muestras adyacentes.

A cada muestra de amplitud de la señal eléctrica se le asigna un valor de entre los que el bit rate del códec permite. Si no existe el equivalente, se toma el inferior más próximo.

Este procedimiento se sigue en todas y cada una de las muestras, independientemente de lo que ocurran con las muestras adyacentes.

La cuantificación uniforme tiene el inconveniente de que es la menos *eficaz* de entre las existentes, pues la probabilidad de ruido de cuantificación es proporcional al incremento de la amplitud de la señal.

La **cuantificación no uniforme** o **no lineal** se aplica cuando se procesan señales no homogéneas que se sabe que van a ser más sensibles en una determinada banda concreta de frecuencias.

Se estudia la propia entropía de la señal analógica y se asignan niveles de cuantificación de manera no uniforme (bit rate variable), de tal modo que se asigne un mayor número de niveles para aquellos márgenes en que la amplitud de la tensión cambia más rápidamente.

En este caso, lo que se hace es estudiar la propia entropía de la señal y asignar niveles de cuantificación de manera no uniforme (utilizando un bit rate variable), de tal modo que se asigne un mayor número de niveles para aquellos márgenes en que la amplitud cambia más rápidamente (contienen mayor *densidad* de información).

Cuando durante la digitalización se ha usado una cuantificación no uniforme, se debe utilizar el mismo circuito no lineal durante la decodificación, para poder recomponer la señal de forma correcta.

La **cuantificación logarítmica** o **escalar** es un tipo de **cuantificación digital** en el que se utiliza una tasa de datos constante, pero se diferencia de la cuantificación uniforme en que como paso previo a la cuantificación se hace pasar la señal por un compresor logarítmico.

Se hace pasar la señal por un compresor logarítmico antes de la cuantificación. Como en la señal resultante la amplitud del voltaje sufre variaciones menos abruptas, la posibilidad de que se produzca un ruido de cuantificación grande disminuye. Antes de reproducir la señal digital, ésta tendrá que pasar por un expansor.

En esta cuantificación tendremos pequeños pasos de cuantificación para los valores pequeños de amplitud y pasos de cuantificación grandes para los valores grandes de amplitud, lo que proporciona mayor resolución en señales débiles al compararse con una cuantifificación uniforme de igual bit rate, pero menor resolución en señales de gran amplitud.

A la salida del sistema, la señal digital ha de pasar por un expansor, que realiza la función inversa al compresor logarítmico. El procedimiento conjunto de compresión y expansión se denomina **companding**.

Los algoritmos Ley Mu y Ley A sirven como ejemplo de cuantificadores logarítmicos.

La **cuantificación vectorial**, un tipo de **cuantificación digital**, en el proceso puede ser idéntico a la cuantificación uniforme (utiliza un bit rate constante) o no constante (utiliza un bit rate variable). La particularidad radica, en que, en lugar de **cuantificar** las muestras retenidas individualmente, se cuantifican por bloques de muestras. Con ello, se logra una cuantificación más eficaz.

En lugar de cuantificar las muestras obtenidas individualmente, se cuantifica por bloques de muestras.

Cada bloque de muestras será tratado como si se tratara de un vector; de ahí el nombre de esta tipología.

La cuantificación vectorial es la más eficiente de todas las modalidades de cuantificación en lo referente al error de cuantificación. No obstante, está más predispuesta a verse afectada por errores de transmisión. Otro inconveniente, es que los procesos informáticos para lograr esta codificación resultan muy complejos.

A.2. Registros del modulo A/D

El módulo de A/D tiene cuatro registros. Estos registros son:

ADRESH : Parte alta del resultado de la conversión

ADRESL: Parte baja del resultado de la conversión

ADCON0: Registro de Control 0 ;control del funcionamiento del conversor

ADCON1, Registro de Control 1; configuración de los pines del puerto

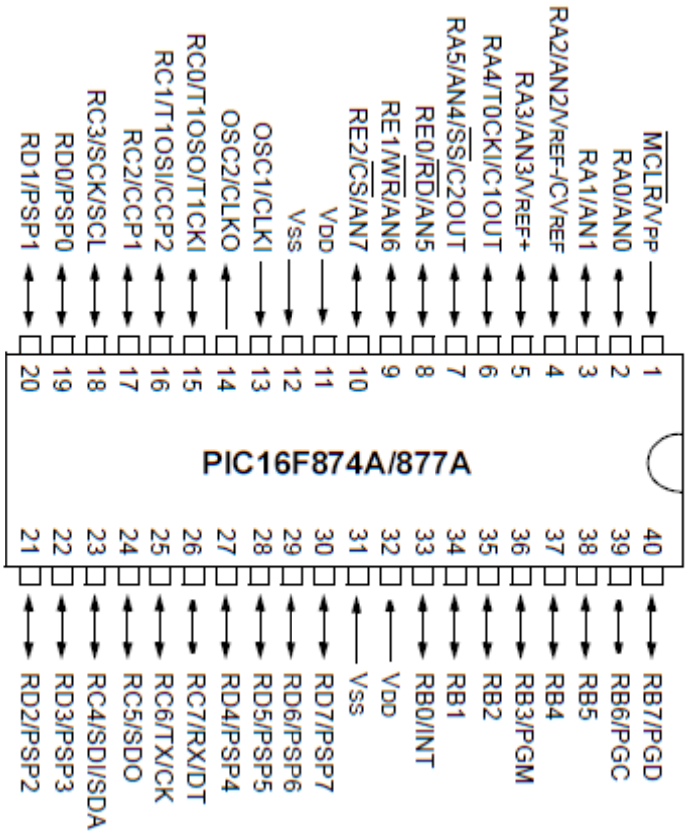


Figura A. 7

A.2.1. El registro ADCON0 (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

Figura A. 8

- bit 7-6: **ADCS1:ADCS0**: En estos dos bits se hace la selección de la frecuencia de reloj para el Convertidor A/D. Se tendra que tener en cuenta el bit ADCS2 que esta en el registro ADCON1

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

Figura A. 9

- bit 5-3: **CH2:CH0:** Aquí se selecciona el canal analógico por donde entrará la señal a digitalizar. En este microcontrolador tenemos 8 canales de entrada al Conversor A/D

000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

Figura A. 10

- bit 2: **GO/#DONE**. bit de estado de la conversión A/D

Si ADON=1

1= La conversión A/D está en marcha (mientras está a 1 se está realizando la conversión)

0 = La conversión ha finalizado. (el bit se pone a cero automáticamente por hardware cuando laconversión A/D finaliza) el resultado de la conversión aparece en ADRESH:ADRESL

- bit 1: **No implementado**: Se lee como “0”
- bit 0: **ADON**: bit de puesta en marcha

1 = El convertidor A/D está operativo

0 = El convertidor A/D está apagado y no consume corriente.

A.2.2. EL REGISTRO ADCON1

El registro ADCON1 es uno de los registros del convertidor A/D del PIC16F877, se trata de un registro de configuración de los pines del puerto, este registro se compone de 8 bits, los cuales describamos su función a continuación:

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Figura A. 11

- bit 7: ADFM Selecciona el formato del resultado de la conversión A/D

1 = Pone en el registro **ARDESH** los 6 bits de mayor peso a “0”

0 = Pone los 6 bits de menor peso del registro **ADRESL** a “0”

- bit 6: Es el otro bit necesario para la selección de la frecuencia de reloj para el Convertidor A/D

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

Figura A. 12

- bits 5-4: No implementados: Se leen como cero

- Bit 3-0: **PCFG3:PCFG0:** bits de configuración de los canales de entrada del convertidor A/D. Se utilizan para configurar las patillas como E/S digital o como entrada analógica de acuerdo con la siguiente tabla:

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

Figura A. 13

A.2.3. LOS REGISTROS ADRESH Y ADRESL

El par de registros **ADRESH:ADRESL** se carga con el resultado de 10 bits de la conversión A/D. Este par de registros se extienden hasta 16 bits. El módulo A/D tiene la

posibilidad de justificar el resultado de 10 bits dentro de los 16 bits de la pareja de registros. La selección del formato de justificación a la izquierda o derecha se realiza con el bit **ADFM**(**ADCON1**). Los bits restantes (a los 10 de la conversión) se llenan con ceros.

Estos dos registros cuando el convertidor A/D está en OFF y no se utiliza, pueden utilizar se como dos registros de 8 bits de propósito general.

Cuando se completa la conversión A/D, el resultado se guarda en los registros y se pone acero el bit **GO/DONE.**

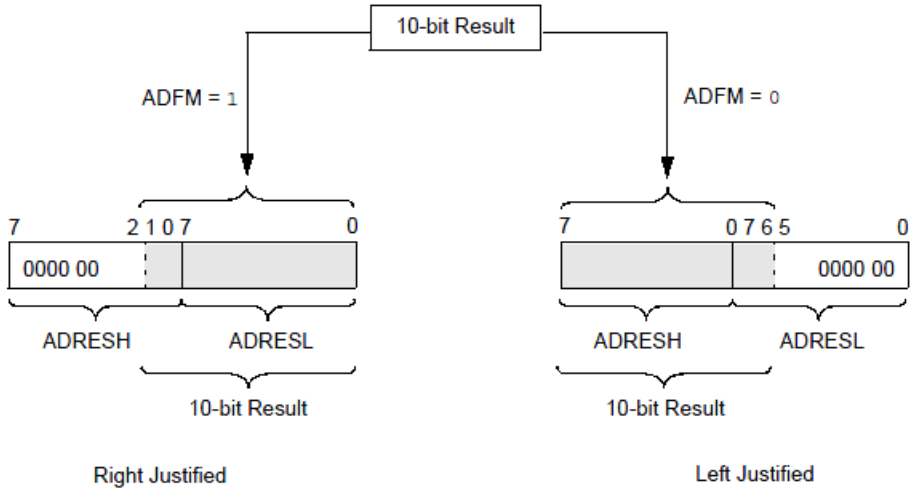


Figura A. 14

Por lo tanto, los 16 bits que forman el registro ARDESH-ARDESL con ADFM=1 tiene los 6 bits de mayor peso a cero y con ADFM=0 los 6 bit de menor peso están a cero, en los 10 bits restantes se almacena el resultado de la conversión.

Hay que tener en cuenta que a la hora de conocer el resultado de ADRESL tenemos que acceder al banco 1 antes de pasar el valor al registro de trabajo (W).

A.3. TEMPORIZACIÓN

Para introducirnos vamos a llamar a 'Tad' como el tiempo de conversión por bit.En la figura A.15 tenemos un esquema de lo que sería el proceso medido en tiempo para estar seguros de que se ha realizado la conversión.

Este comienza a funcionar en el tiempo de adquisición cuando activamos el **bitGO/#DONE(ADCON0).** El tiempo de adquisición es el tiempo que el modulo A/D esta conectado a un voltaje externo.

Pero tenemos que tener en cuenta que para una nueva conversión podemos preparar la configuración del módulo conversor A/D y si se desea realizar una interrupción del modulo conversor A/D, se exige un mínimo de 2*TAD para realizar una nueva conversión.

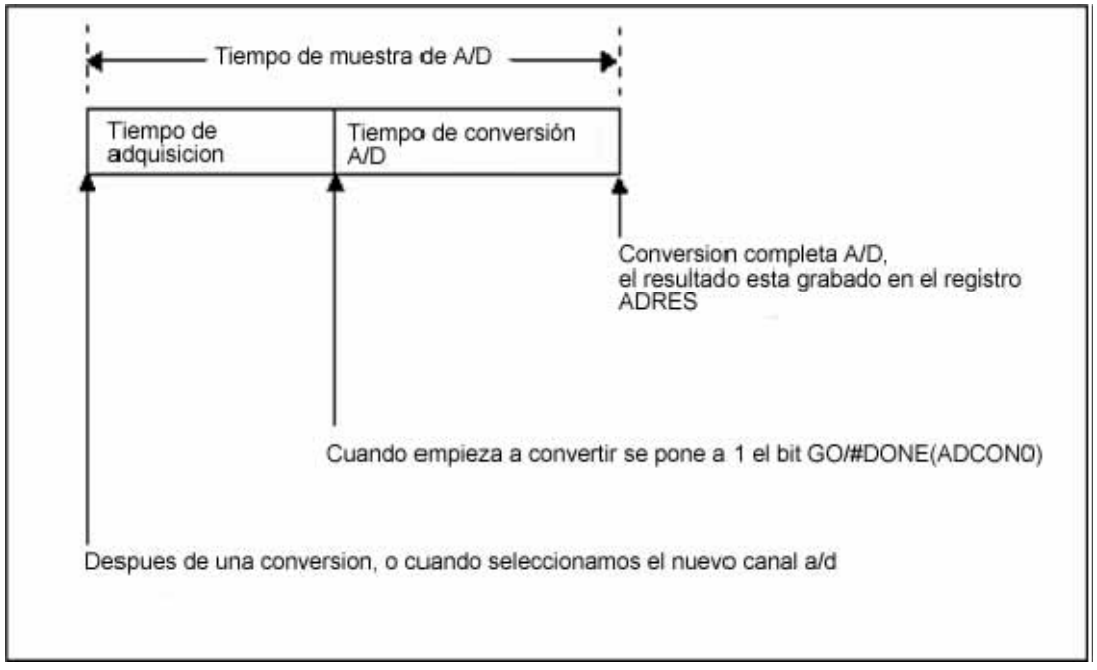


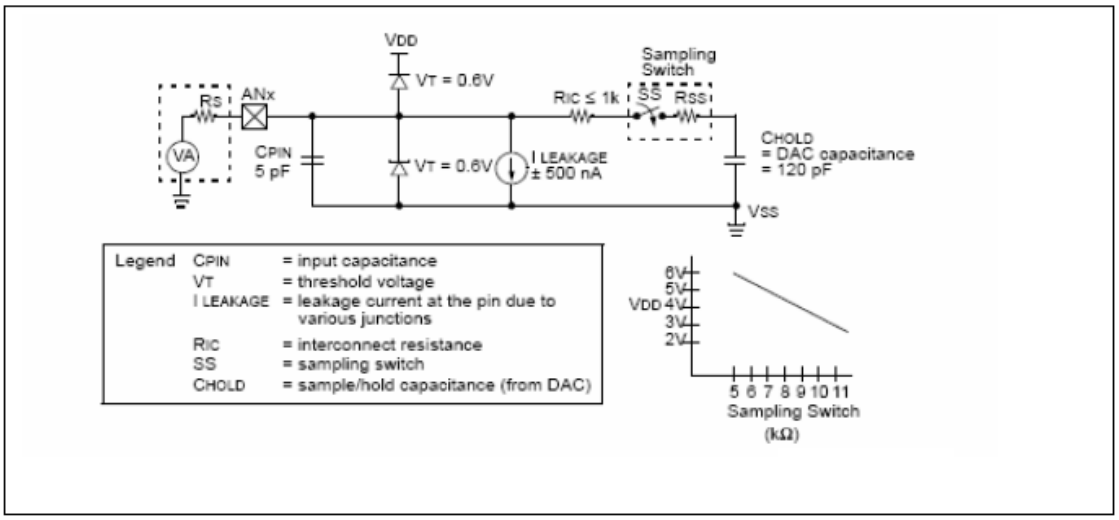
Figura A. 15

En la figura A.16 tenemos una ecuación que nos resolverá el tiempo de adquisición asumiendo un error explicado anteriormente, para 1024 pasos del convertidor A/D. Estos cálculos han sido basados en el esquema de uso de la figura 3.21. TACQ sería el tiempo de respuesta del amplificador, TC seria el tiempo de carga

del condensador (figura 3.22) que guarda el dato y TCOFF sería el coeficiente de temperatura (que este solo se utilizaría para temperaturas > 25º)

TACQ = Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient

= TAMP + TC + TCOFF
= 2 µs + TC + [(Temperature – 25°C)(0.05 µs/°C)]
TC = CHOLD (RIC + RSS + RS) In(1/2047)
= - 120 pF (1 kΩ + 7 kΩ + 10 kΩ) In(0.0004885)
= 16.47 µs
TACQ = 2 µs + 16.47 µs + [(50°C – 25°C)(0.05 µs/°C)
= 19.72 µs



CHOLD	=	120 pF
Rs	=	10 kΩ
Conversion Error	≤	1/2 LSb
VDD	=	5V → Rss = 7 kΩ
Temperature	=	50°C (system max.)
VHOLD	=	0V @ time = 0

Figura A. 16

A.3.2. SELECCIÓN DEL RELOJ DEL CONVERTIDOR A/D

El convertidor A/D requiere un mínimo de 12 TAD para la conversión de los 10 bits, La señal de reloj para la conversión A/D se selecciona por software mediante los bits **ADCS1:ADCS0** y **ADCS2.** Las siete posibles opciones son :

- 2 TOSC
- 4 TOSC
- 8 TOSC
- 16 TOSC
- 32 TOSC
- 64 TOSC
- Oscilador Interno RC (2-6 µs)

Para realizar conversiones correctas el reloj del convertidor A/D debe seleccionarse para asegurar un tiempo mínimo de TAD de 1,6 mS. La figura A.17 siguiente muestra los tiempos de TAD dependiendo de la señal de reloj del micro.

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS2:ADCS1:ADCS0	
2 TOSC	000	1.25 MHz
4 TOSC	100	2.5 MHz
8 TOSC	001	5 MHz
16 TOSC	101	10 MHz
32 TOSC	010	20 MHz
64 TOSC	110	20 MHz
RC(1, 2, 3)	x11	**(Note 1)**

Note 1: The RC source has a typical TAD time of 4 µs but can vary between 2-6 µs.
2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for Sleep operation.
3: For extended voltage devices (LF), please refer to **Section 17.0 "Electrical Characteristics"**.

Figura A. 17

A.3.3. TIEMPOS DE FUNCIONAMIENTO

Si se pone a cero el bit GO/#DONE durante la conversión, se aborta la conversión actual.El par de registros no se modificarán parcialmente con los bits que se hayan completado hasta el momento. Es decir, los registros ADRESH:ADRESL seguirán conteniendo el valor de la última conversión completa (o el último valor que se haya escrito en ADRESH:ADRESL)después de abortar la conversión A/D, es requerido el TAD de espera para realizar la próxima adquisición comience. Después de 2 TAD de espera, la adquisición en cauce se comienza automáticamente.

En la Figura A.18 después de poner el bit GO a uno, la primera vez el segmento tiene un TCY mínimo y un TAD máximo.

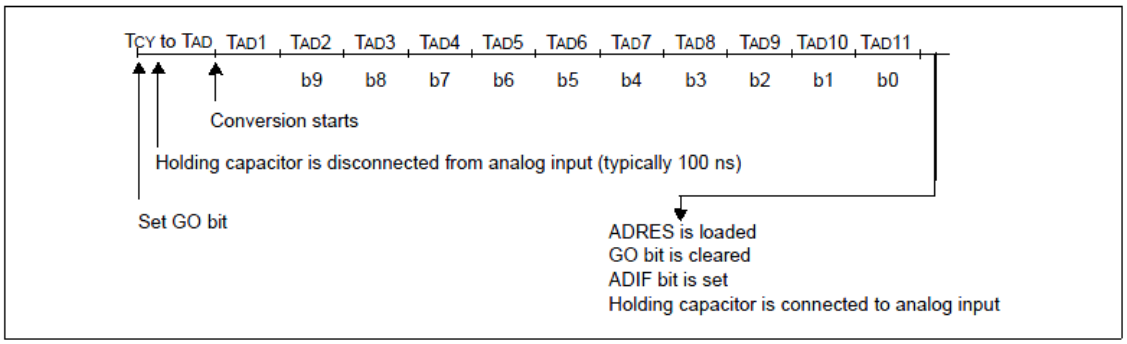


Figura A. 18

Para concluir con este apartado decir que el bit GO/#DONE no debe ponerse a uno en la misma instrucción que se pone en ON el convertidor A/D.

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

ANEXO B

ANEXO B. INVERSOR TRIGGER SCHMITT 40106

Con el fin de profundizar más en el entendimiento del desarrollo de las placas diseñadas se considera interesante realizar un análisis del inversor Trigger Schmitt para el posterior comprendimiento del funcionamiento de las placas CNY70 y LDR.

En el presente anexo se describe el componente utilizado en el capitulo 7.

B.1. INVERSOR TRIGGER SCHMITT

Dado que algunos sensores no proporcionan señales digitales puras es necesario conformar dicha señal antes de aplicarla al microcontrolador , como en el ejemplo que se muestra en la figura B.4.

Una forma sencilla de conformar una señal en digital es mediante puertas Trigger Schmitt (ver figura B.1 derecha), como las que tiene el circuito integrado 40106. Este dispositivo contiene 6 inversores Trigger Schmitt distribuidos según la figura B.1 izquierda.

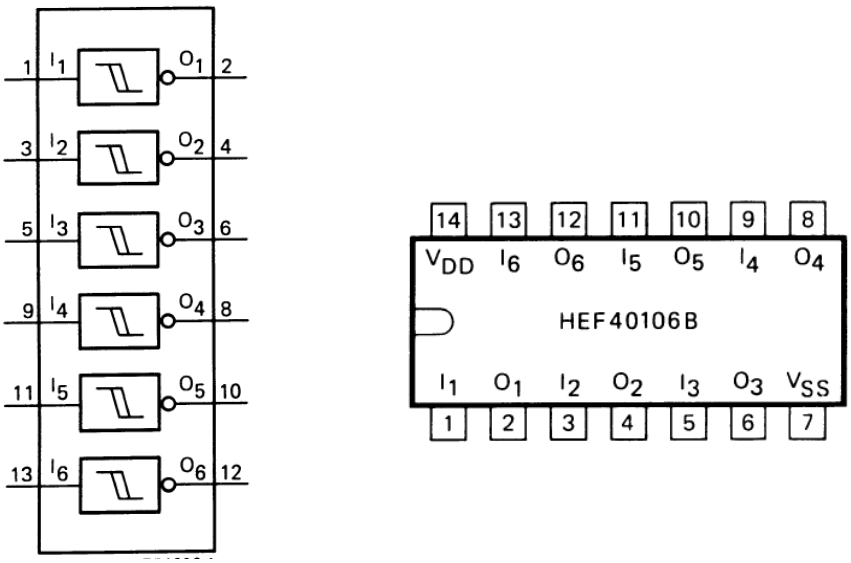


Figura B. 1 Distribución y encapsulado del Trigger Schmitt

Como se puede comprobar en la figura B.2 los dispositivos tienen una caracteristicas de transferencia tal que si la tensión de entrada aumenta desde 0V hasta 5V la transición se produce siguiente la curva A y conmuta para el valor Vp, de nominado umbral superior. Por el contrario, si la entrada está a un nivel alto y disminuye hasta 0 V, la transición se produce siguiendo la cuerba B cuando se alcanza el denominado umbral inferio Vn.

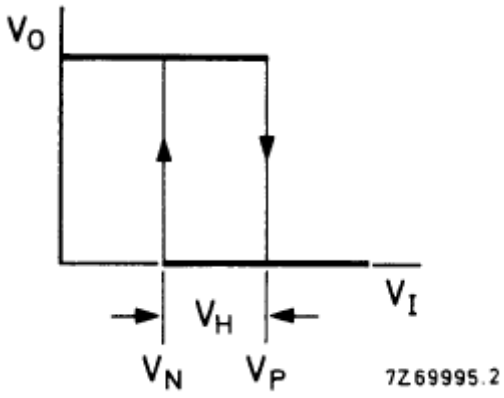


Figura B. 2 : Características de transferencia

Estos valores, Vp y Vn, para el caso del Trigger Schmitt 40106 dependen de la tensión de alimentación . En la curva de transferencia del inversor es importante observar que la transción de salida Alto a Bajo es distinta que la de Bajo a Alto . A este fenómeno se le denomina Histéresis y viene determinada por la variable Vh. Estas variables pueden tomar los valores de la tabla B.1.

DC CHARACTERISTICS

V_{SS} = 0 V; T_{amb} = 25 °C

	V_{DD} V	SYMBOL	MIN.	TYP.	MAX.	
Hysteresis	5		0,5	0,8		V
voltage	10	V_H	0,7	1,3		V
	15		0,9	1,8		V
Switching levels	5		2	3,0	3,5	V
positive-going	10	V_P	3,7	5,8	7	V
input voltage	15		4,9	8,3	11	V
negative-going	5		1,5	2,2	3	V
input voltage	10	V_N	3	4,5	6,3	V
	15		4	6,5	10,1	V

Tabla B. 1 : Valores de Vp, Vn y Vh

El inversor Trigger Schmitt, y todos los dispositivos con este tipo de funcionamiento, utiliza el símobolo de la figura B.3 . Para indicar que pueden responder de manera fiable ante señales que cambian con lentitud . Esta simbología se basa en añadir al dibujo de la gráfica de hitéresis en la entrada correspondiente.

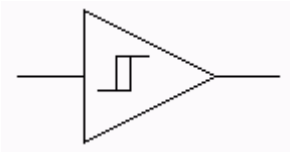


Figura B. 3: Simbolo Trigger Schmitt

Estos circuitos son de gran utilidad cuando se desea controlar un circuito digital con señales que no lo son o señales digitales con una señal de ruido . En la figura B.4 se muestra como actúa un circuito inversor frente a una señal que no es puramente digital. Cuando la señal V1 alcanza al valor Vp la salida Vo bascula aun nivel bajo y no vuelve a tomar un nivel alto hasta que la entrada no llegue a Vn.

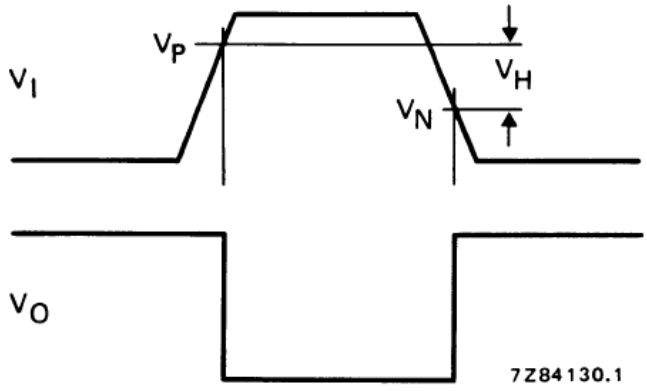


Figura B. 4 : Ejemplo de señal digitalizada

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

ANEXO C

ANEXO C. LIBRERÍA DESARROLLADA PARA EL PIC16F877A

En el presente anexo, se describe la librería desarrollada para el control del micro robot “MyBot”

C.1. Mlibreria.inc

```
ORG 4
    btfss cte,0
    goto Timer0
    goto Timer0_1
;Eliminamos Warning
ERRORLEVEL -302
ERRORLEVEL -203
ERRORLEVEL -205
ERRORLEVEL -207
ERRORLEVEL -204
;LIBRERIA CONTROL MOTOR HITEC HSR-1422CR CON UN RELOJ DE 20 MHZ

;*************************************;
;*    DEFINICIONES Y  VARIABLES FIJAS   *
;*************************************;
      CBLOCK
      Registro100micros ; Registro auxiliar para conseguir una temporización fina.
      carga_fbms_MDCH   ;Factor bajo para servo derecho ej: 18.9ms--->189

    carga_fams_MDCH   ;Factor alto para servo izquierdo ej: 1.1ms--->11
    carga_fams_MIZQ ;Factor alto para servo izquierdo
    falta;tiempo que falta a fams para completar el periodo
    cte      ; constante para el uso de subrutinas de interrupción
    cte2 ; constante para diferenciar el sentido de la dirección rectilinea
    tiempo;variable para indicar el tiempo de ejecucion de cada macro
    movimiento; variable para indicar el tipo de movimiento que se desea
    ENDC


     TMR0_Carga100micros  EQU   d'11';Si 20 Mhz preescaler 4 y valor 133 |
preescaler 2 valor 10--este
    ;El Valor obtenido con stopwatch para contar cada 100micros para 8mhz= 63
    Carga equ .3036;TMR1=40536-->100ms para 8 mhz, 3036 para 20mhz con el mismo
preescaler


    #DEFINE         MIZQ   PORTD,1  ; Se definen los motores y los puertos en donde
se contectarán
    #DEFINE     MDCH    PORTD,0
    #DEFINE     LED     PORTA,0
    #DEFINE     cnyiz   PORTD,2
    #DEFINE     cnydc   PORTD,3
```

```
;************
;*     INICIO   *
;************
INI

      bsf          STATUS,RP0                      ; Acceso al Banco 1.
    MOVLW 0x06                    ; Se configura el PORTA
    MOVWF ADCON1                  ; como digital
    bcf MIZQ
    bcf MDCH
    bcf LED
    bsf cnyiz
    bsf cnydc
      movlw  b'00000000'
      movwf  OPTION_REG                     ; Sin Prescaler, se le asigna al WDT
      bcf          STATUS,RP0                      ; Acceso al Banco 0.
      movlw  TMR0_Carga100micros       ; Carga el TMR0 con valor 100 micros
      movwf  TMR0
    return

;************
;*     CÓDIGO *
;************


;----FUNCIONES CON TIEMPO: 3 funciones, 6 movimientos en 2 direcciones

;En función de la disposición de los motores en el microbot se definen
;las siguientes funciones
;*************************************************
;*    GIRACENTRO   VelSent1, Tiempo                    *
;*************************************************
;*Descripción: Comando para cambiar la velocidad ,   *
;*el sentido y la direccion del robot,en resumen para*
;*girar sobre centro del eje de las ruedas.           *
;*El movimiento es circular    y los parámetros son  *
;VelSent(Velocidad y Sentido) del 0 al 7             *
;ver TablaVerdad_VelSent:                            *
;del 0 al 3 sentido robot antihorario                *
;del 4 al 7 sentido robot horario                    *
;y Tiempo 1-->100ms, 10--->1s,100--->10s             *
;*************************************************
GIRACENTRO MACRO   VelSent, Tiempo
    clrf cte;uso de subrutina CONTROL MOTORES
    movlw .0
    movwf movimiento;movimiento circular
    movlw VelSent; Velocidad y sentido deseado
    call PWM; A partir de VelSent deducimos el fams y fbms

;TEMPORIZADOR1
    movlw Tiempo ; Tiempo que deseamos que este operando esta macro
    call TEMPORIZADOR

;ANULACION_DE_TIMERS1
    movlw    b'00000000'
    movwf    INTCON ; Desautoriza la interrupcion del timer0
    movlw b'00000000' ; Ponemos en off el timer1
    movwf T1CON
    clrf PORTD  ; Limpiamos el puertoD

ENDM
```

```
;*************************************************
;*    GIRA  Servo, VelSent, Tiempo                *
;*************************************************
;*Descripción: Comando para cambiar la velocidad ,   *
;*el sentido y la direccion del robot,en resumen para*
;*girar a la izquierda o a la derecha.               *
;*El movimiento es circular    y los parámetros son  *
;*Direc : 1= GIRA IZQ, 2 = GIRA DHA                 *
;VelSent(Velocidad y Sentido)                        *
;si Direc :1 y  VelSent  0 al 3 gira a la izq hacia delante *
;si Direc :1 y  VelSent  4 al 7 gira a la izq hacia atrás*
;si Direc :2 y  VelSent  0 al 3 gira a la dcha  hacia atrás*
;si Direc :2 y  VelSent  4 al 7 gira a la dcha hacia delante*
; y Tiempo 1-->100ms, 10--->1s,100--->10s
;*************************************************
GiraT  MACRO Direc, VelSent, Tiempo

    clrf cte ;uso de subrutina CONTROL MOTORES
    movlw Direc
    movwf movimiento ;movimiento de giro
    movlw VelSent ; Velocidad y sentido deseado
    call PWM ; A partir de VelSent deducimos el fams y fbms

;TEMPORIZADOR2
    movlw Tiempo ; Tiempo que deseamos que este operando esta macro
    call TEMPORIZADOR


;ANULACION_DE_TIMERS2
    movlw    b'00000000'
    movwf    INTCON ; Desautoriza la interrupcion del timer0
    movlw b'00000000' ; Ponemos en off el timer1
    movwf T1CON
    clrf PORTD  ; Limpiamos el puertoD

ENDM
```

```
;*****************************************************
;*    MueveRectoT      VelSent, Tiempo          *
;*****************************************************
;*Descripción: Comando para cambiar la velocidad y   *
;*el sentido de los dos servomotores.                *
;*El movimiento es rectilinieo y los parámetros son  *
;VelSent(Velocidad y Sentido) del 0 al 3 DELANTE
;y del 4 al 7 ATRÁS
; y Tiempo 1-->100ms, 10--->1s,100--->10s
;*****************************************************

 MueveRectoT  MACRO VelSent, Tiempo
    movlw .1
    movwf cte;USO DE LA SUBRUTINA CONTROL MOTORES2
    movlw VelSent
    call PWM2; sacamos tambien el factor alto equivalente para el otro servo

;TEMPORIZADOR3
    movlw Tiempo ; Tiempo que deseamos que este operando esta macro
    call TEMPORIZADOR

;ANULACION_DE_TIMERS3

    movlw    b'00000000'
    movwf    INTCON ; Desautoriza la interrupcion del timer0
    movlw b'00000000' ; Ponemos en off el timer1
    movwf T1CON
    clrf PORTD  ; Limpiamos el puertoD


ENDM
;---------------------****-------------------

;*************************************;
;*     TABLA DE VERDAD  :VelSent           *
;*************************************************************************************
;del 0 al 3 Sentido HORARIO DEL SERVO DERECHO y velocidad decreciente(0 max vel y 3
min vel)
;del 4 al 7 Sentido ANTIHORARIO DEL SERVO DERECHO y velocidad decreciente(4 max vel
y 7 min vel)
;*************************************************************************************

TablaVerdad_VelSent ;

       addwf  PCL,F
       DT     d'19',d'18',d'17',d'16',d'11',d'12',d'13',d'14'

;*************************************;
;*     MOVIMIENTOS   (direccion)           *
;*************************************;

Tabla_MOV;

       addwf  PCL,F
       DT     b'00000011',b'00000010',b'00000001'
;2motores funcionando,motor izquierdo func., motor derecho func.


;*****************************SUBRUTINAS TIMERS y otras
*****************************
```

```
;*************************************;
;*     PWM  :velocidad y sentido            *
;*************************************;
PWM

    call TablaVerdad_VelSent; cálculo de fams y fbms
    movwf carga_fams_MDCH
    sublw   .200
    movwf  Registro100micros
    movwf   carga_fbms_MDCH
    movlw    b'10100000'
    movwf    INTCON ; Autoriza interrupción del TMR0 (T0IE) y la GIE
    return


;*************************************;
;*     PWM2  :velocidad y sentido           *
;   para linea recta                    *
;*************************************;


PWM2 ;calculos referidos desde el motor derecho

    call TablaVerdad_VelSent; cálculo de fams y fbms
    movwf carga_fams_MDCH
    sublw   .200
    movwf  Registro100micros
    movwf   carga_fbms_MDCH
    movfw  carga_fams_MDCH
    sublw  .30

Sentido_de_la_marcha
    movwf  carga_fams_MIZQ ;factor alto equivalente(SENTIDO OPUESTO MISMA VEL) a
carga_fams_MDCH ,para mov rectilineo
    subwf carga_fams_MDCH,W ; carga_fams_MDCH - carga_fams_MIZQ
    btfss STATUS,C ; si C=0 (resultado negativo)atras si C=1(resultado positivo)
adelante
    goto atras
    goto adelante

atras
    movlw .1
    movwf cte2; 0 adelante, 1 atras
    movf  carga_fams_MDCH,W
    subwf carga_fams_MIZQ,W
    movwf falta ;variable para cumplir el tiempo de carga_fams_MIZQ
    goto tmr
adelante
    movwf falta ;variable para cumplir el tiempo de carga_fams_MDCH
    movlw .0
    movwf cte2; 0 adelante, 1 atras
tmr
    movlw    b'10100000'
    movwf    INTCON                         ; Autoriza interrupción del TMR0
(T0IE) y la GIE.

    return
```

```
;**************************************;
;*     TEMPORIZADOR                     *
;**************************************;
TEMPORIZADOR

    movwf tiempo
    movlw b'00110001' ; se configura el Timer1 con un prescaler de 8
    movwf T1CON
    call cargatmr1 ;
    btfss    PIR1,TMR1IF  ; ¿Se ha producido desbordamiento?
    goto     $-1    ; Todavía no. Repite.
    decfsz  tiempo,F
    goto $-4
    return
;**************************************;
;*     SUBRUTINA CARGA TIMER1           *
;**************************************;
cargatmr1

       movlw  low Carga
       movwf  TMR1L
       movlw  high Carga
       movwf  TMR1H
       bcf          PIR1,TMR1IF         ; Resetea el flag de desbordamiento del
TMR1.
    return

;**************************************;
;*     CONTROL MOTORES                  *
;**************************************;

Timer0

       movlw  TMR0_Carga100micros
       movwf  TMR0                           ; Recarga el TMR0.
       decfsz Registro100micros,F            ; Decrementa el contador
Registro100micros*(fa ó fb)
       goto   FinInterrupcion
    movlw .0
    xorwf PORTD,W
    btfss    STATUS,Z                                  ; Testea el último
estado de los motores
       goto   EstabaEncendido
EstabaApagado

    movfw movimiento
    call Tabla_MOV
    movwf  PORTD
       movf   carga_fams_MDCH,W                    ; Repone el contador
       goto   CargaRegistro100micros          ;
EstabaEncendido
    nop; Para conseguir un ajuste fino del Timer0
    nop
    nop
    nop
    nop
    nop
    clrf PORTD
       movf   carga_fbms_MDCH,W                    ; Repone el contador

CargaRegistro100micros
       movwf  Registro100micros
FinInterrupcion
       bcf          INTCON,T0IF                   ; Repone flag del TMR0.
    retfie
```

```
;**************************************;
;*     CONTROL MOTORES2                    *
;**************************************;

Timer0_1

       movlw  TMR0_Carga100micros
       movwf  TMR0                          ; Recarga el TMR0.
       decfsz Registro100micros,F           ; Decrementa el contador
Registro100micros*(fa ó fb)
       goto   FinInterrupcion
    movlw .0
    xorwf PORTD,W
    btfss    STATUS,Z                                  ; Testea el último
estado de los motores
       goto   EstabaEncendido1
EstabaApagado1

    movlw .0
    call Tabla_MOV
    movwf  PORTD
    btfsc cte2,0 ;0 adelante , 1 atras
    goto $+3
       movf   carga_fams_MIZQ,W                  ; Repone el contador con el
valor + bajo para ser el primero en apagarse
       goto   CargaRegistro100micros          ;
    movf     carga_fams_MDCH,W                  ; Repone el contador con el
valor + bajo para ser el primero en apagarse
       goto   CargaRegistro100micros          ;
EstabaEncendido1
    bcf STATUS,C
    btfsc cte2,0
    goto atras1
adelante1
    bcf STATUS,C
    rrf PORTD,F
    btfsc PORTD,0 ; ¿ Ha cumplido el MDCH su tiempo de fams?
    goto $+3  ; No , completa el tiempo que falta
    movf     carga_fbms_MDCH,W                  ; Repone el contador nuevamente
    goto     CargaRegistro100micros
    movf falta,W ; seguimos a 1 hasta que cumpla carga_fams_MDCH
    goto   CargaRegistro100micros

atras1
    bcf STATUS,C
    rlf PORTD,F
    btfsc PORTD,1
    goto $+4
    movf     falta,W                       ; Repone el contador nuevamente
    subwf carga_fbms_MDCH,W ; W= carga_fbms_MDCHMIZQ , para no crear una nueva
variable
    goto     CargaRegistro100micros
    movf falta,W ; seguimos a 1 hasta que cumpla carga_fams_MDCH
    goto   CargaRegistro100micros
```

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

ANEXO D

ANEXO D. LIBRERÍA DESARROLLADA PARA EL PIC32XXX

En el presente anexo, se describe la librería desarrollada para el control del micro robot “MyBot” para microcontroladore de gama alta PIC32

D.1. Mlib.h

```
//************************************************************
//DEF VARIABLES Y CTES
//************************************************************
#define             MIZQ   PORTD,1;
#define     MDCH    PORTD,0 ;
float VS[8]={1.9,1.8,1.7,1.6,1.1,1.2,1.3,1.4};
int movimientos[3]={0x3,0x2,0x1};
float fa;
float fa2;
float fb;
float fb2;
float falta;
int mov;
int recto;
int temp;
float count;
int ocupado;
//************************************************************
//DEF funciones
//************************************************************
void GIRACENTRO (int VelSent, int Tiempo);
void GIRAT (int Direc, int VelSent2, int Tiempo2);
void MUEVERECTO (int VelSent, int Tiempo);
void PWM (float factoralto);
void PWM2(float VelSent);
void Temporizador (int tiempor);
void AnulInte(void);
void INIM(void);

//************************************************************
//FUNCIONES CONTROL MYBOT
//************************************************************
void GIRACENTRO (int VelSent1, int Tiempo1)
{

   PORTD = 0x0;

   fa= VS[VelSent1];
   mov = movimientos[0];
//Calculamos el factor bajo y activamos interrupciones para motores
   PWM(fa);
   Temporizador(Tiempo1);
   ocupado=1;
   return;
}
```

```
//************************************************************
void GIRAT (int Direc, int VelSent2, int Tiempo2)
{
   PORTD = 0x0;
   mov = movimientos[Direc];
   fa= VS[VelSent2];
//Calculamos el factor bajo y activamos interrupciones para motores
   PWM(fa);
   Temporizador(Tiempo2);
   ocupado=1;

   return;
}
//************************************************************
void MUEVERECTO (int VelSent3, int Tiempo3)
{

   PORTD = 0x0;
   mov = movimientos[0];
   fa= VS[VelSent3];
   PWM2 (fa);
   Temporizador(Tiempo3);
   ocupado=1;
   return;
}
//************************************************************
//Servicio de interrupciones multivectorial
//************************************************************
void __ISR( _TIMER_3_VECTOR, ipl7) T3InterruptHandler( void)
{
    // 1. T3 handler is responsible for incrementing count

count--;
if (count==0)
{
AnulInte();
mT3ClearIntFlag();
mT3IntEnable(0);
//mINTDisableSystemMultiVectoredInt();
//return;

}
else
 // 3. re-enable interrupts immediately (nesting)
{
//asm("ei");
// 2. clear the flag and exit
mT3ClearIntFlag();
return;
}
    //return;
} // T3 Interrupt Handler
```

```
//************************************************************
void __ISR( _TIMER_2_VECTOR, ipl1) T2InterruptHandler( void)
{
    // 3. re-enable interrupts immediately (nesting)
   asm("ei");

    // 4. T2 handler code here
// estaba apagado

   if ( (PORTD)==0)
       {
         PORTD = mov;
         temp= ((fa*1000)*36)/16; // BreakPoint
         PR2 = (int)temp;//pasamos temp a numero entero
       }
//estaba encendido
   else
       {
         PORTD = 0x0;
         temp= ((fb*1000)*36)/16; //BreakPoint
         PR2 = (int)temp;//pasamos temp a numero entero
       }


    // 5. clear the flag and exit
    mT2ClearIntFlag();

} // T2 Interrupt Handler
//************************************************************
void __ISR( _TIMER_4_VECTOR, ipl1) T4InterruptHandler( void)
{
    // 3. re-enable interrupts immediately (nesting)
   asm("ei");

    // 4. T2 handler code here
// estaba apagado

   if ( PORTD ==0) //¿estaba apagado?
       {
         PORTD = mov;
         if(recto==1)//adelante
         {
         temp= ((fa2*1000)*36)/16; // cargamos el tiempo mas pequeño
         PR4 = (int)temp;//pasamos temp a numero entero
         mT4ClearIntFlag();
         return;
         }
         else//atras
         {
         temp= ((fa*1000)*36)/16; // cargamos el tiempo mas pequeño
         PR4 = (int)temp;//pasamos temp a numero entero
          mT4ClearIntFlag();
          return;
         }
       }
```

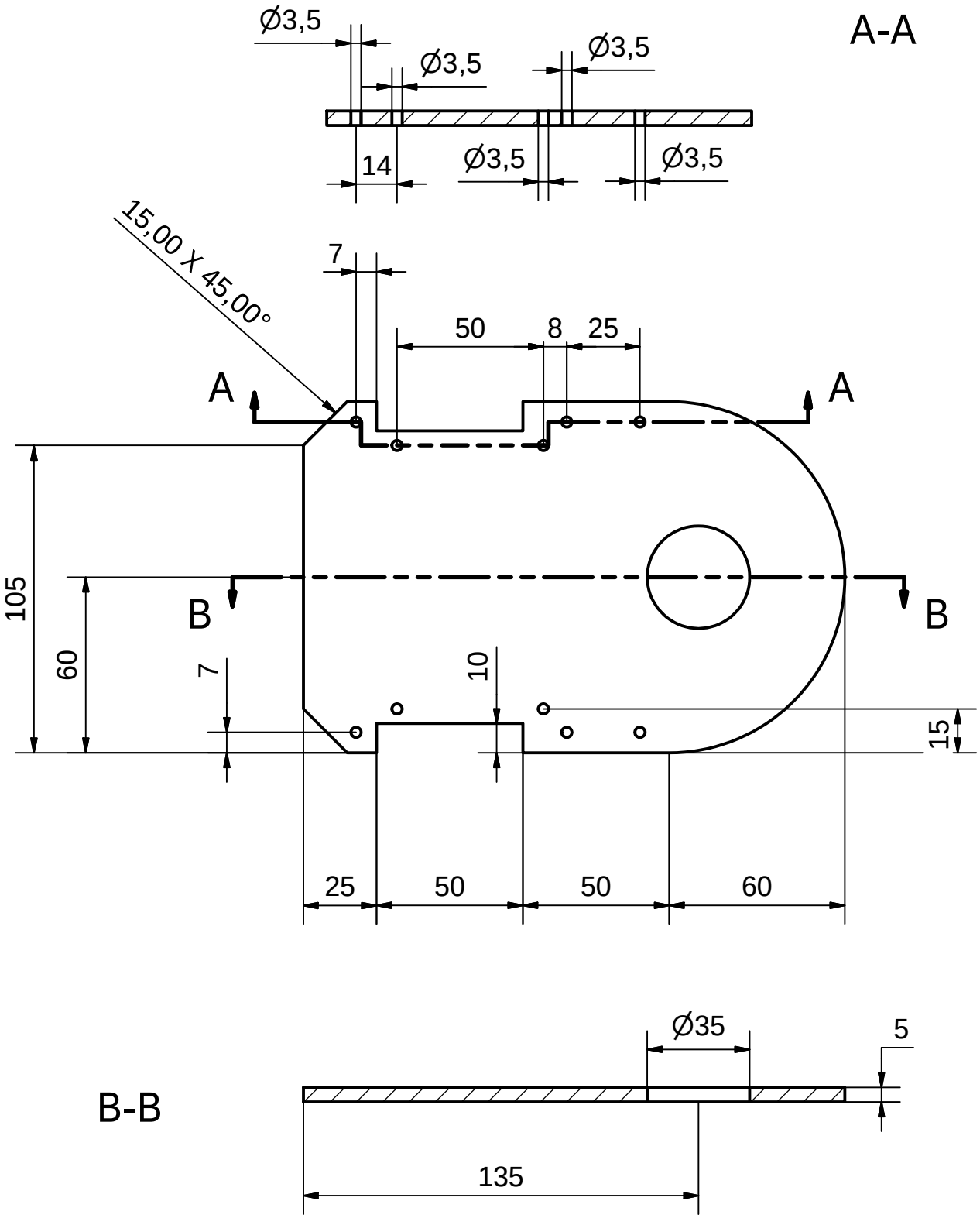
```
//estaba encendido
 //  if ( PORTD^0!=0)
   //    {
         if(PORTD==3) //se completa el tiempo que falta
          {
             if(recto==1)//adelante
             {
                    PORTD = 0x1;// esta encendido solo el motor derecho
                    temp= ((falta*1000)*36)/16;
                    PR4 = (int)temp;//pasamos temp a numero entero
                    mT4ClearIntFlag();
                    return;
              }
           else //atras
              {
                    PORTD = 0x2;// esta encendido solo el motor izquierdo
                    temp= ((falta*1000)*36)/16;
                    PR4 = (int)temp;//pasamos temp a numero entero
                    mT4ClearIntFlag();
                    return;
              }
           }
     //  }
         else // ya se ha completado  apagamos
           {
            PORTD = 0x0;
            if(recto==1)
           {
            temp= ((fb*1000)*36)/16;
            PR4 = (int)temp;//pasamos temp a numero entero
            mT4ClearIntFlag();
            return;
           }
           else
           {
            temp= ((fb2*1000)*36)/16;
            PR4 = (int)temp;//pasamos temp a numero entero
            mT4ClearIntFlag();
            return;
            }
         }
    // 5. clear the flag and exit


} // T4 Interrupt Handler

//************************************************************
//INICIO
//************************************************************
void INIM (void)
{
TRISD = 0; // Configura el puerto D como salida
return;
}

//************************************************************
//SUBRUTINAS CONTROL MYBOT
//************************************************************
```

```
//************************************************************
//SUBRUTINAS CONTROL MYBOT
//************************************************************

void PWM2 (float factoralto)
{
// 1. init timers
   fb= 20-factoralto;
   fa2=3-factoralto;
   fb2 =20-fa2;
   falta=(factoralto -fa2);
   if (falta>0)//Si el resultado es positivo sentido adelante
      {
        temp= (int)((fb*1000)*36)/16;
        PR4 = temp;//pasamos temp a numero entero
        T4CON = 0x8040; //1000 0000 0100 0000;//preescaler 16
        recto =1;
       }
   else//sentido hacia atras
      {
        falta=falta*(-1);
        temp= (int)((fb2*1000)*36)/16;
        PR4 = temp;//pasamos temp a numero entero
        T4CON = 0x8040; //1000 0000 0100 0000;//preescaler 16
        recto=0;
      }
 // 2. init interrupts

   mT4SetIntPriority( 1); // El que tiene mayor prioridad  será servido primero
   mT4SetIntSubPriority(1);
   INTEnableSystemMultiVectoredInt();
   mT4IntEnable( 1);
   return;
}
//************************************************************
void PWM (float factoralto)
{

// 1. init timers
   fb= 20-factoralto;
   temp= (int)((fb*1000)*36)/16;
   PR2 = temp;//pasamos temp a numero entero
   T2CON = 0x8040; //1000 0000 0100 0000;//preescaler 16

 // 2. init interrupts

   mT2SetIntPriority(1); // El que tiene mayor prioridad  será servido primero
   mT2SetIntSubPriority(1);
   INTEnableSystemMultiVectoredInt();
   mT2IntEnable( 1);
   return;
}
//************************************************************
```

```
void Temporizador (int tiempor)
{


   count=(tiempor*10/2);
   PR3 = 28125 ;// contara 0.2 segundos

   T3CON = 0x8070;//prescaler de 256
   mT3SetIntPriority(1);
   mT3SetIntSubPriority(3);
   mT3IntEnable( 1);
// hay que crear un flag que indique que ha terminado de contar
   return;
}

//************************************************************
void AnulInte(void)
{
mINTDisableSystemMultiVectoredInt();

mT2IntEnable(0);
mT4IntEnable(0);
mT2ClearIntFlag();
mT4ClearIntFlag();
PORTD = 0x0;
PR3=0;
PR2=0;
PR4=0;
fa=0;
fb=0;
fa2=0;
fb2=0;
falta=0;
count=0;
ocupado=0;
return;

}
```

“MYBOT:DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN DE UN MICRO ROBOT DE BAJO COSTE CON TECNOLOGIA PIC”

PLANOS



Autor	Directora:	Escala:
Diego P. González González	Asunción Vicente Ripoll	1:2

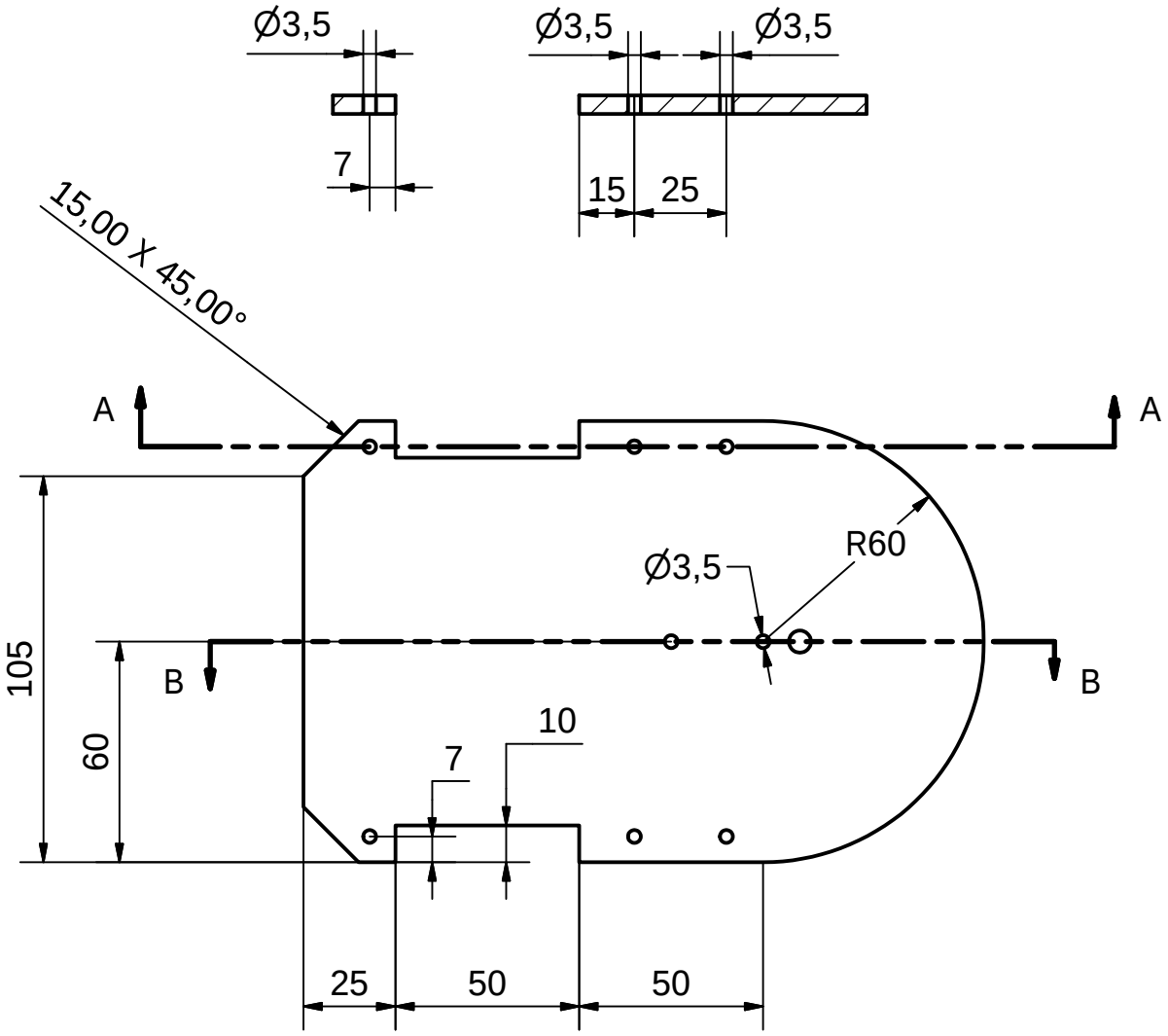
mybot Proyecto final de carrera — UNIVERSITAS Miguel Hernández

Estructura MyBot piso inferior

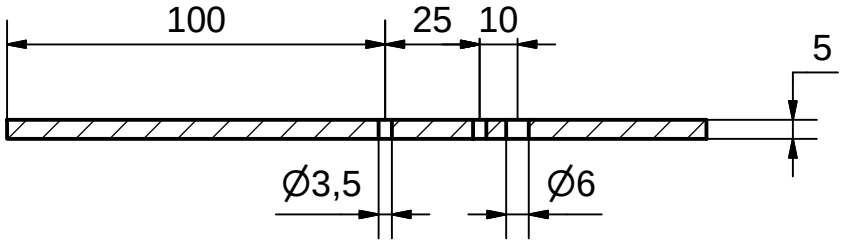
estruc_inf_mybot

Nº de hoja 1 / 1

A-A

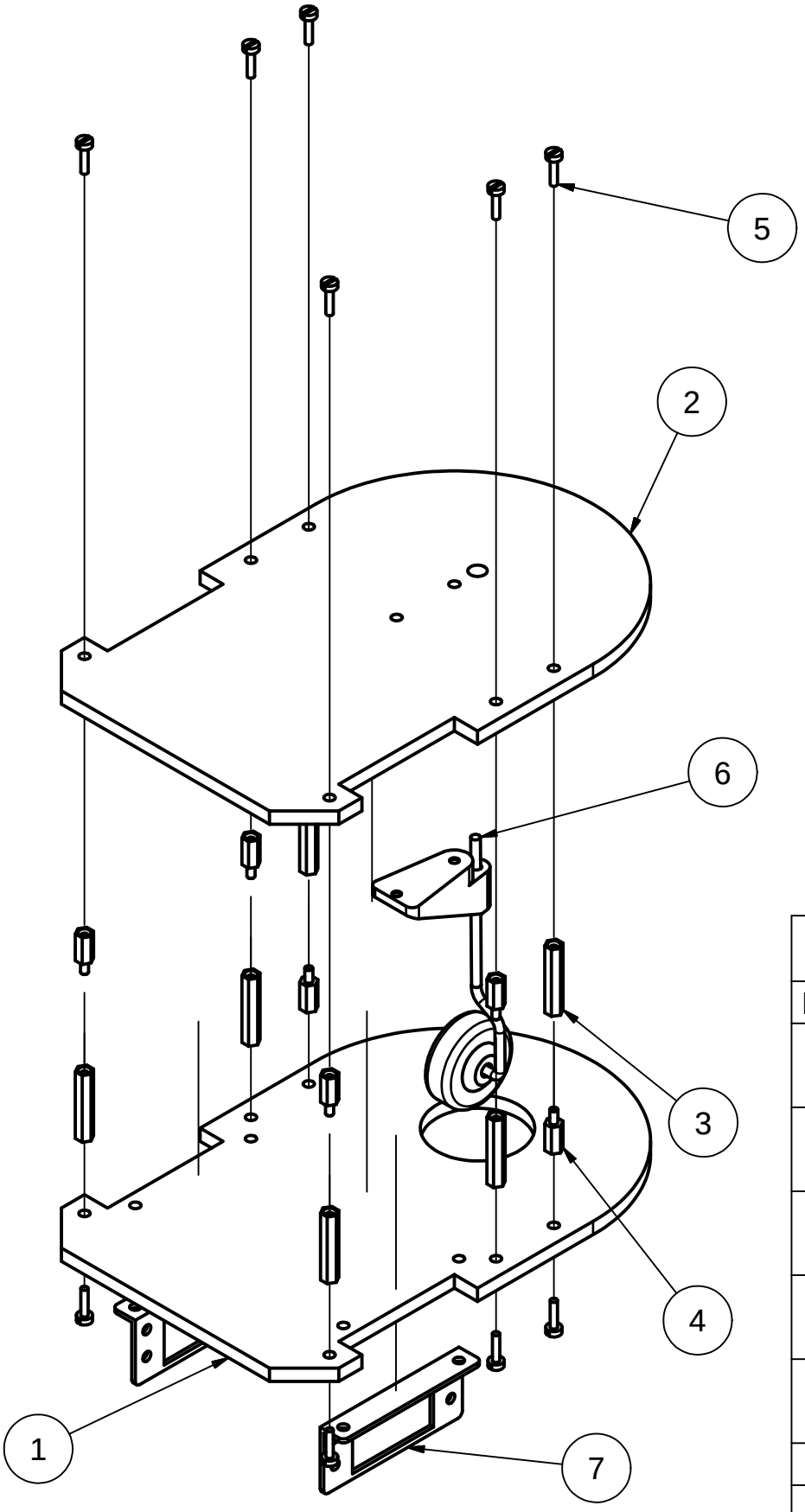


B-B



Autor :	Directora :	Escala:
Diego P. González González	Asunción Vicente Ripoll	1:2

mybot Proyecto final de carrera	Estructura MyBot piso superior	
	estruc_sup_mybot	Nº de hoja: 1 / 1



PARTS LIST		
ITEM	QTY	PART NUMBER
1	1	estruc_inf_mybot
2	1	estruc_sup_mybot
3	6	separador metalico h-h
4	6	separador metalico h-m
5	12	tornillo din84 2.5x10
6	1	rueda loca
7	2	posicionador servo

Autor:	Directora:	Escala:
Diego P. González González	Asunción Vicente Ripoll	1:2

myBot
Proyecto final de carrera

UNIVERSITAS Miguel Hernández

Conjunto explotado

conj_explotado

Nº de hoja: 1 / 1

BIBLIOGRAFÍA

[1] Microbot Moway, http://www.moway-robot.com/.

[2] "Microcontrolador PIC16F84, Desarrollo de proyectos", E. Palacios, F. Remiro, L.J. López. Ra-Ma, 2004

[3] Proyecto final de carrera "DESARROLLO DE APLICACIONES Y PUESTA EN MARCHA DEL SISTEMA EASYPIC PARA MICROCONTROLADORES PIC" de María de las Nieves Robles Botella

[4] Proyecto final de carrera, "MANEJO DE UNA PANTALLA TÁCTIL CON EL PIC32 PARA EL CONTROL DE DISPOSITIVOS EXTERNOS" de Jesus Fernández González

[5] Microchip, http://www.microchip.com

[6]Información sobre el microcontrolador PIC16F877A,http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010242

[7]*Información sobre el núcleo del PIC32 M4K CPU basado en la tecnología MIPS32.* http://www.mips.com/products/processors/32-64-bit-cores/mips32-m4k/index.cfm#features

[8]"Brief Introduction to MIPS32®M4K® Core Shadow Registers for microcontroller Applications", MIPS Technologies, Inc.

[9] Información sobre la familia de de microcontroladores de 32 bits, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2591

[10] Mikroelektronika,http://www.mikroe.com/.

[11] MSE-F87X de *Ingeniería de Microsistemas Programados S.L.,* http://www.msebilbao.com/tienda/product_info.php?cPath=53_55&products_id=93&osCsid=206c2845429374aa5adfc132d9f7b144

[12] Tarjeta de control PIC-P40 de OLIMEX, http://www.olimex.com/dev/pic-p40.html

[13] Información y descarga del MPLAB IDE, Microchip Technology Inc., http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

BIBLIOGRAFÍA

[14]Información sobre el starterKitPIC32, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en532453

[15] Información y descarga del compilador C32, Microchip Technology Inc http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en532454&redirects=c32

[16] Información PICFLASH, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en532453

[17] Información y descarga del programador PIC-PG2, http://www.olimex.com/dev/pic-pg2.html

[18] Información y descarga del programa de grabacion IC-PROG, http://www.ic-prog.com/

[19] http://www.ai.mit.edu/people/brooks/brooks.html

[20] http://www.microbotica.es/

[21] "Where am I?Sensors and Methods for Mobile Robot Positioning " ,Información general sobre micro robotica Edited and compiled by J. Borenstein.

[22] Información sobre LDR , http://es.wikipedia.org/wiki/Fotorresistencia

[23] Programa CAE gratuito de diseño de cirtcuitos EAGLE, http://www.cadsoft.de/

[24]Información sobre el sensor CNY70, http://www.vishay.com/optical-sensors/list/product-83751/

[25] Información sobre el sensor Sharp GP2Y0A21YK, http://www.sharpsma.com/Page.aspx/americas/en/part/GP2Y0A21YK/

[26] Manual del Acelerómetro ADXL330 iMEMS, Analog Devices http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf

[27] Información y descarga accel board de MiKroelektronika, http://www.mikroe.com/en/tools/accel/

[28]Inforamción sobre el servomotor HITEC HSR1422CR, http://www.robomaa.com/index.php?main_page=product_info&products_id=18&language=en

[29] “Programming 32-bit Microcontrollers in C: Exploring The PIC32 (Embedded Technology)”,L. Di Jasio. Newnes Books (2008).