DISEÑO Y PROGRAMACIÓN DE UN SISTEMA DE COMUNICACIÓN BASADO EN CAN-BUS

Agradecimientos

A mis padres y hermano por todo el apoyo que me han brindado durante tantos buenos y malos momentos.

A Ramón González Carvajal por su inestimable ayuda durante varios meses; y ser además de tutor, amigo.

A Miguel Ángel Bobillo por toda la ayuda prestada durante la realización de esta ardua tarea; y convertirse también en un buen amigo.

A toda mi familia por la confianza y la ayuda que siempre me han prestado a pesar de la distancia.

A todos mis amigos que durante tantos años han hecho que mi paso por la Universidad haya sido la mejor experiencia de mi vida.

De todo corazón muchas gracias a todos.

Rafa

1 INTRODUCCIÓN

El presente proyecto ha sido desarrollado en la Escuela Superior de Ingenieros de la Universidad de Sevilla, bajo la tutela del profesor D. Ramón González Carvajal. Consiste en el diseño y programación de una interfaz CAN-Bus para el robot forestal ROFOR de la empresa Servicios Forestales S.L.

A continuación se expondrá la situación de la que se parte para la realización del proyecto así como los objetivos a cumplir con la finalización del mismo. Se hablará tanto del hardware como del software implicado en el proyecto y de las posibles líneas futuras del mismo.

1.1 SITUACIÓN DE PARTIDA

La máquina forestal se compone fundamentalmente de tres elementos:

- Retroexcavadora: compuesta por cabina y brazo articulado
- Autómata: situado en la cabina
- Procesador Forestal: situado en el extremo del brazo articulado

La comunicación entre el autómata y el procesador forestal se realiza con una manguera multihilo de 26 hilos. Estos hilos son:

- 1 de alimentación permanente a 24 V
- 2 de alimentación a 24 V que se activan cuando se pulsa algún mando
- 1 de tierra
- 2 señales analógicas para la medición del diámetro
- 2 señales digitales procedentes de encoder para medida de longitud de árbol
- 18 señales digitales de entrada/salida

Este sistema de comunicación descrito presentaba los siguientes problemas fundamentales:

- Difícil manipulación de la manguera multihilo

- Alta probabilidad de avería del conjunto
- Compleja sujeción de la manguera multihilo
- Aparatosos procesos de recambio

Se hacía necesaria por tanto, la búsqueda de un nuevo sistema de comunicación que paliase toda esta problemática. Para ello se usaría un bus de comunicación muy extendido en el campo de la automoción y en actual expansión, el **"CAN bus".** Los nuevos 6 hilos cumplirián las siguientes funciones:

- 1 de alimentación permanente a 24 V
- 2 de alimentación a 24 V que se activan cuando se pulsa algún mando
- 2 hilos que forman el bus de campo
- 1 hilo de reserva

Así pues, se realizaron en la Escuela Superior de Ingenieros de Sevilla y bajo la tutela del profesor D. Ramón González Carvajal los siguientes proyectos:

- *"Diseño de una interfaz CAN Bus-PLC para un robot industrial",* realizado por Miguel Ángel Bobillo Dorado
- *"Diseño de un concentrador de datos de un robot industrial Telemandazo por CAN-Bus"*, realizado por Mª Isabel Gutierrez Armada

Gracias al trabajo realizado en los mencionados proyectos se conseguía sustituir el sistema de comunicación anterior; por otro constituido por dos placas (una situada en la cabina y conectada al autómata, y otra situada en el procesador forestal y conectada a los sensores y actuadotes del mismo) conectadas entre sí por únicamente 6 hilos.

De esta forma se conseguía no sólo evitar los problemas comentados anteriormente respecto a la manguera multihilo; sino que también nos permitiría disponer de un sistema comercial de constrastado funcionamiento, a más bajo coste.

No obstante el resultado de ambos proyectos abría un campo amplio de posibles mejoras que nos llevaran a un sistema más rápido, fiable y de fácil monitorización.

Con esa motivación surge el presente proyecto, de manera que con su realización se ha pretendido cubrir algunas de las más importantes mejoras sugeridas por los anteriores.

1.2 OBJETIVOS

Teniendo en cuenta y como punto de partida los proyectos que sobre el sistema se realizaron con anterioridad al presente; se establecen los siguientes objetivos a cubrir en la realización del que nos ocupa:

- Comunicación eficaz mediante CAN-Bus
- Evolución desde el PIC16F877 al PIC18F458
- Unificación de protocolos serie
- Eliminación de cuellos de botella debidos a la lentitud de algunos dispositivos
- Programación de las señales de encoder
- Uso del compilador Picc Compiler (alto nivel)

De todos ellos los dos primeros son los que cobran mayor relevancia. En cuanto al CAN-Bus la comunicación conseguida a la finalización de los anteriores proyectos no era todo lo fiable y robusta que cabría esperar. Además no se conseguían las tasas binarias máximas (ni tan siquiera cercanas) que el protocolo ofrece.

En relación a este objetivo está justificado el empleo de un nuevo PIC ya que aquellos que pertenecen a la familia 18FXXX incorporan un módulo CAN que evita así la comunicación mediante SPI entre el PIC16F877 y el integrado MCP2510. Esto aportará al sistema una mayor fiabilidad y velocidad.

En cuanto a los objetivos tercero y cuarto decir que se ha optado por el

protocolo SPI en detrimento del I2C debido a que el mercado ofrece mayores velocidades en los dispositivos basados en comunicación mediante SPI que en los basados en comunicación mediante I2C. De esta forma cumplimos ambos objetivos; por un lado unificación de protocolos, y por otro eliminación de cuellos de botella debido a comportamientos de baja velocidad por parte de alguno de los dispositivos integrados en el sistema.

Para terminar decir también que actualmente las señales de encoder continúan conectadas directamente desde el procesador forestal al autómata. Será pues objetivo de este proyecto, incorporar dichas señales al procesado que se realiza en el cabezal de la máquina y enviar así la información relativa a las mismas a la cabina; mediante el protocolo CAN-Bus.

En cuanto al compilador, si bien no es fundamental su empleo si que resulta de gran ayuda en la medida en que facilita enormemente las tareas de depuración de errores, así como las de generación y modificación del código de programa. Actualmente se ha realizado el desarrollo en lenguaje ensamblador con la consiguiente óptima eficiencia del código pero la también evidente complejidad en cuanto a legibilidad y comprensión del mismo.

1.3 FUNDAMENTOS

A continuación se incluyen algunos fundamentos teóricos básicos para la comprensión del proyecto como son los protocolos CAN Bus y SPI ; además de unas breves nociones sobre microcontroladores a fin de entender mejor su elección para este sistema.

1.3.1 MICROCONTROLADORES

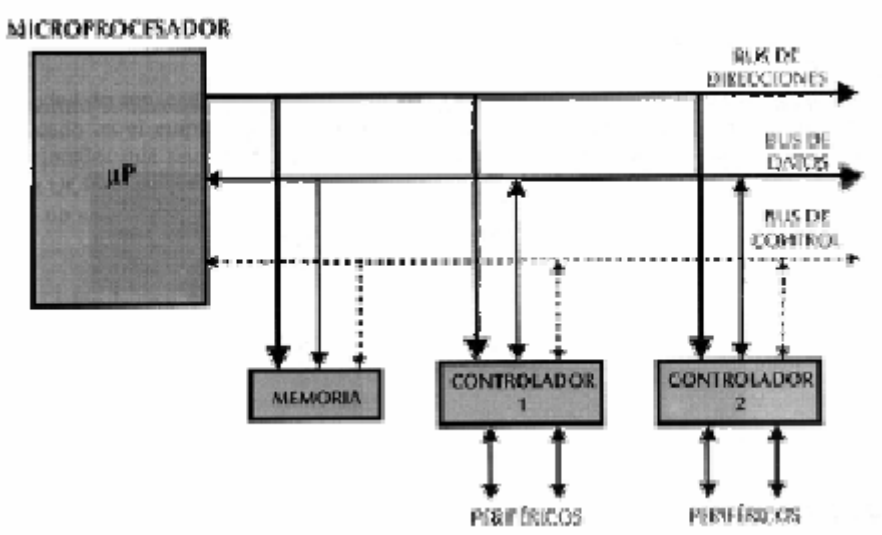
1.3.1.1 ¿Qué es un microcontrolador?

Un microcontrolador es un computador completo (microprocesador + E/S + memoria + otros periféricos), aunque de limitadas prestaciones, que está contenido en el chip de un circuito integrado programable y se destina a gobernar una sola tarea con el programa que reside en su memoria. Sus líneas

de entrada/salida soportan el conexionado de los sensores y actuadores del dispositivo a controlar.

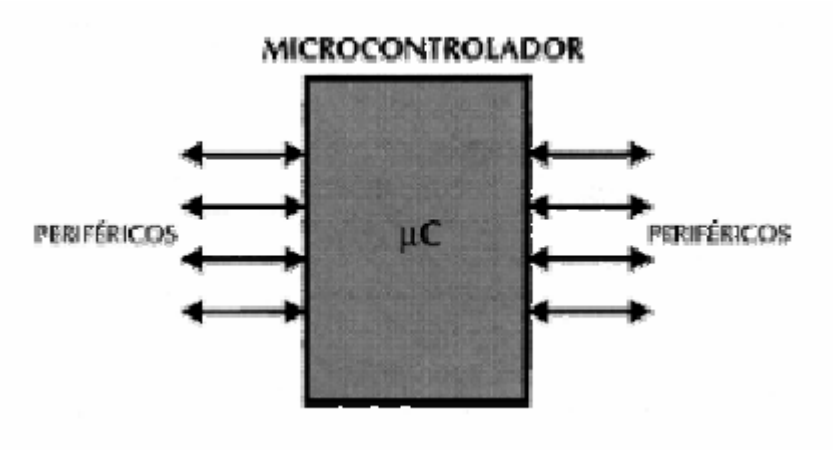
1.3.1.2 Diferencia entre microprocesador y microcontrolador

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (UCP), también llamada procesador, de un computador. La UCP está formada por la Unidad de Control, que interpreta las instrucciones, y el camino de datos, que las ejecuta. Los pines de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la Memoria y los Módulos de E/S y configurar un computador implementado por varios circuitos integrados. Se dice que un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine.



Estructura de un sistema abierto basado en un microprocesador. La disponibilidad de los buses en el exterior permite que se configure a la medida de la aplicación. Si sólo se dispusiese de un modelo de microcontrolador, éste debería tener muy potenciados todos sus recursos para poderse adaptar a las exigencias de las diferentes aplicaciones. En la práctica cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos. Es posible seleccionar la capacidad de las memorias, el número de líneas de E/S, la cantidad y potencia de los

elementos auxiliares, la velocidad de funcionamiento, etc. Por todo ello, un aspecto muy destacado del diseño es la selección del microcontrolador a utilizar.



El microcontrolador es un sistema cerrado. Todas las partes del computador están contenidas en su interior y sólo salen al exterior las líneas que gobiernan los periféricos.

1.3.1.3 Diversidad de usos de un microcontrolador

Las extensas áreas de aplicación de los microcontroladores, que se pueden considerar ilimitadas, como pueden ser juguetes, horno microondas, frigoríficos, televisores, computadoras, impresoras, módems, el sistema de arranque de nuestro coche, etc.

1.3.1.4 El mercado de los microcontroladores

Existe una gran diversidad de microcontroladores. Quizá la clasificación más importante sea entre 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. La razón de esta tendencia es que los microcontroladores de 4 y 8 bits son apropiados para la gran mayoría de las aplicaciones, lo que hace absurdo emplear micros más potentes y consecuentemente más caros. Uno de los sectores que más tira del mercado del microcontrolador es el mercado automovilístico. De hecho, algunas de las

familias de microcontroladores actuales se desarrollaron pensando en este sector, siendo modificadas posteriormente para adaptarse a sistemas más genéricos. El mercado del automóvil es además uno de los más exigentes: los componentes electrónicos deben operar bajo condiciones extremas de vibraciones, choques, ruido, etc. Y seguir siendo fiables. En cuanto a las técnicas de fabricación, cabe decir que prácticamente la totalidad de los microcontroladores actuales se fabrican con tecnología CMOS 4 (Complementary Metal Oxide Semiconductor). Esta tecnología supera a las técnicas anteriores por su bajo consumo y alta inmunidad al ruido. La distribución de las ventas según su aplicación es la siguiente:

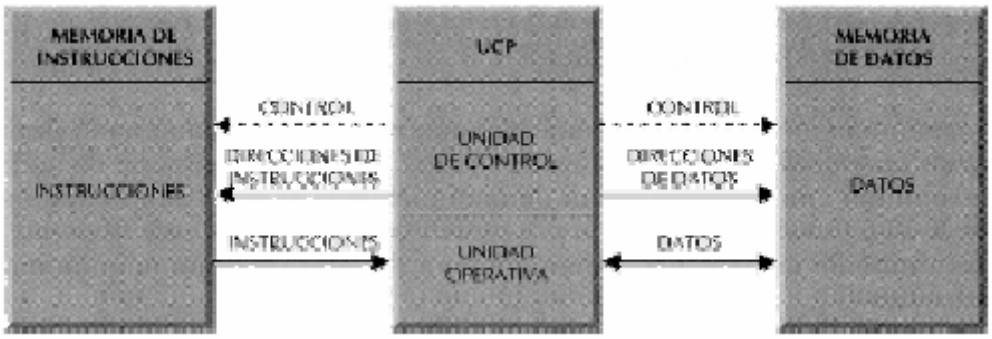
- Un 30% se absorbe en las aplicaciones relacionadas con los ordenadores y sus periféricos.
- Otro 25% se utiliza en las aplicaciones de consumo (electrodomésticos, juegos, TV, vídeo, etc.)
- El 20% de las ventas mundiales se destinó al área de las comunicaciones.
- Un 15% fue empleado en aplicaciones industriales.
- Del resto de los microcontroladores vendidos en el mundo, aproximadamente un 10% fueron adquiridos por las industrias de automoción.

También los modernos microcontroladores de 32 bits van afianzando sus posiciones 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.

1.3.1.5 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 datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).

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



1.3.1.6 El procesador o UCP

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 la instrucción en curso, su decodificación y la ejecución de la operación que implica dicha instrucción, así como 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 (Computadores de Juego de Instrucciones Complejo). Disponen de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución.

Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.

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.

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

SISC (Computadores de Juego de Instrucciones Específico): En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es "específico", es decir, las instrucciones se adaptan a las necesidades de la aplicación prevista.

1.3.1.7 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 conjunto de instrucciones que ejecuta la aplicación. Otra parte de memoria es del tipo RAM, volátil, y se destina a guardar las variables y los datos. Según el tipo de memoria ROM que dispongan los microcontroladores, la aplicación y utilización de los mismos es diferente. Las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado son:

- 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 grandes cantidades de los mismos.
- OTP. Es una memoria no volátil de sólo lectura "programable una sola vez" por el usuario. OTP (One Time Programmable). La versión OTP es recomendable cuando la tirada del producto es baja , o bien, en la construcción de prototipos y series muy pequeñas.
- EPROM. Los microcontroladores que disponen de memoria EPROM (Erasable Programmable Read Only Memory) pueden borrarse y grabarse muchas veces. Si se desea borrar el contenido, disponen de una ventana de cristal en su superficie por la que se somete a la EPROM a rayos ultravioleta durante varios minutos. Las cápsulas son de material cerámico y son más caros que los

microcontroladores con memoria OTP que están hechos generalmente con plástico.

- EEPROM. Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programmable Read Only Memory). No disponen de ventana de cristal en la superficie. Los microcontroladores dotados de memoria EEPROM una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se quiera sin ser retirados de dicho circuito. Para ello se usan "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 puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Este tipo de memoria es relativamente lenta.
- FLASH. Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar, es programable en el circuito, es más rápida que la EEPROM y tolera más ciclos de escritura/borrado.

1.3.1.8 Puertos de entrada/salida

La principal utilidad de las líneas de E/S es comunicar 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.

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 (Universal Serial Bus), bus moderno serie para los PC
- Bus I2C, interfaz serie de dos hilos desarrollado por Philips
- CAN (Controller Area Network), para permitir la adaptación con redes de conexionado multiplexado desarrollado conjuntamente por Bosch e Intel para el cableado de dispositivos en automóviles

1.3.1.9 Reloj principal

Todos los microcontroladores disponen de un circuito oscilador que sincroniza de todas las operaciones del sistema.

Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo.

1.3.1.10 Recursos auxiliares

Cada fabricante oferta numerosas versiones de una arquitectura básica de microcontrolador. En algunas amplía las capacidades de las memorias, en otras incorpora nuevos recursos, en otras reduce las prestaciones al mínimo para aplicaciones muy simples, etc. La labor del diseñador es encontrar el modelo mínimo que satisfaga todos los requerimientos de su aplicación. De esta forma, minimizará el coste, el hardware y el software.

Los principales recursos específicos que incorporan los microcontroladores son:

- Temporizadores o "Timers". Se emplean para controlar periodos de tiempo (temporizadores) y para llevar la cuenta de acontecimientos que suceden en el exterior (contadores).
- Perro guardián o "Watchdog". Temporizador que cuando se bloquea el sistema, provoca un reset automáticamente.
- Protección ante fallo de alimentación o "Brownout". Se trata de un circuito que resetea al microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo ("brownout").
- Estado de reposo o de bajo consumo. Para ahorrar energía cuando el microcontrolador no está funcionando, éstos disponen de una instrucción especial (SLEEP en los PIC), que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. Al activarse una interrupción ocasionada por el acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo.
- Conversor A/D (CAD). Los microcontroladores que incorporan un Conversor A/D (Analógico/Digital) pueden procesar señales analógicas.

- Conversor D/A (CDA). Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analógica.
- 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. La salida del comparador proporciona un nivel lógico 1 ó 0 según una señal sea mayor o menor que la otra.
- Modulador de anchura de impulsos o PWM. Son circuitos que proporcionan en su salida impulsos de anchura variable.

1.3.1.11 ¿Qué microcontrolador emplear?

A la hora de escoger el microcontrolador a emplear hay que tener en cuenta multitud de factores, como la documentación y 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.):

- **Costes.** Para el fabricante que usa el microcontrolador en su producto una diferencia de precio en el microcontrolador de algunos céntimos es importante (el consumidor deberá pagar además el coste del empaquetado, el de los otros componentes, el diseño del hardware y el desarrollo del software). Si el fabricante desea reducir costes debe tener en cuenta las herramientas de apoyo con que va a contar: emuladores, simuladores, ensambladores, compiladores, etc. Es habitual que muchos de ellos siempre se decanten por microcontroladores pertenecientes a una única familia.
- **Aplicación.** Antes de seleccionar un microcontrolador es imprescindible analizar los requisitos de la aplicación:
  - *Procesamiento de datos:* puede ser necesario que el microcontrolador realice cálculos críticos en un tiempo limitado. En ese caso debemos asegurarnos de seleccionar un dispositivo suficientemente rápido para ello. Por otro lado, habrá que tener en cuenta la precisión de los datos a manejar: si no es suficiente con un microcontrolador de 8 bits, puede ser necesario acudir a microcontroladores de 16 ó 32 bits, o incluso a hardware de coma flotante.
  - *Entrada Salida:* para determinar las necesidades de Entrada/Salida del sistema es conveniente conocer el diagrama de bloques del mismo, de tal forma que sea sencillo identificar la cantidad y tipo de señales a controlar.

Una vez realizado este análisis puede ser necesario añadir periféricos externos o cambiar a otro microcontrolador más adecuado a ese sistema.

- *Consumo:* algunos productos que incorporan microcontroladores están alimentados con baterías. Lo más conveniente en un caso como éste puede ser que el microcontrolador esté en estado de bajo consumo pero que despierte ante la activación de una señal (una interrupción) y ejecute el programa adecuado para procesarla.

- *Memoria:* El tipo de memoria a emplear vendrá determinado por el volumen de ventas previsto del producto: de menor a mayor volumen será conveniente emplear EPROM, OTP y ROM. En cuanto a la cantidad de memoria necesaria deberemos hacer una estimación de cuánta memoria volátil y no volátil es necesaria y si es conveniente disponer de memoria no volátil modificable.

- *Ancho de palabra:* el criterio de diseño debe ser seleccionar el microcontrolador de menor ancho de palabra que satisface los requerimientos de la aplicación. Usar un microcontrolador de 4 bits supondrá una reducción en los costes importante, mientras que uno de 8 bits puede ser el más adecuado si el ancho de los datos es de un byte. Los microcontroladores de 16 y 32 bits, debido a su elevado coste, deben reservarse para aplicaciones que requieran altas prestaciones (Entrada/Salida potente o espacio de direccionamiento muy elevado).

- *Diseño de la placa:* la selección de un microcontrolador concreto condicionará el diseño de la placa de circuitos. Deberá tenerse en cuenta el encapsulado del mismo, de los cuales podemos encontrar el encapsulado DIP o DIL, Este es el encapsulado más empleado en montaje por taladro pasante en placa. Este puede ser cerámico (marrón) o de plástico (negro). Un dato importante en todos los componentes es la distancia entre patillas que poseen, en los circuitos integrados es de vital importancia este dato, así en este tipo el estándar se establece en 0,1 pulgadas (2,54mm). Se suelen fabricar a partir de 4, 6, 8, 14, 16, 22, 24, 28, 32, 40, 48, 64 patillas, estos son los que más se utilizan.

1.3.2 PROTOCOLO CAN BUS

El CAN se creó inicialmente como un método de comunicación serie bastante robusto.

La mayoría de las aplicaciones de redes persiguen un acercamiento por capas a la implementación de sistemas. Este acercamiento sistemático permite interoperabilidad entre productos de diferentes fabricantes. Por esto ISO (Organización del Standard Internacional) creó un standard, como base para conseguir dicho acercamiento. Se llamo el Modelo de Referencia por Capas de Redes ISO (Interconexión de Sistemas Abiertos).

El protocolo del CAN implementa en sí mismo casi todo lo incluido en las dos capas más bajas de este modelo de referencia. La porción de comunicación intermedia del modelo fue dejada a propósito fuera de la especificación del CAN para permitir a los diseñadores de sistemas adaptar y optimizar el protocolo de comunicación múltiple para maximizar la flexibilidad. Con esto llega la posibilidad de la interoperabilidad.

Para facilitar algunos de estos objetivos, la Organización del Standard Internacional y la Sociedad de Ingenieros de Automoción (SAE), han definido algunos protocolos basados en CAN; que incluyen la definición de la Interface de Dependencia Media, tal y como son especificadas las dos capas más bajas:

- ISO11898 es un standard para aplicaciones de alta velocidad.
- ISO11519 es un standard para aplicaciones de baja velocidad.
- J1939 (de SAE) es para aplicaciones de bus

Todos estos protocolos especifican un bus eléctrico de tensión diferencial de 5v como la interface física.

El resto de las capas de los protocolos ISO/OSI se dejan para que las implemente el que desarrolle el software del sistema. Los protocolos de las capas más altas (HLPs) son usados generalmente para implementar las cinco capas superiores del Modelo de Referencia OSI:

- Para estandarizar el procedimiento de comienzo incluyendo el bit rate usado.
- Distribuir direcciones a través de los nodos participantes o tipos de

mensajes.

- Determinar la estructura de los mensajes
- Proveer de rutinas de error

1.3.2.1 Acceso de múltiple sensores con detección de colisión (CSMA/CD)

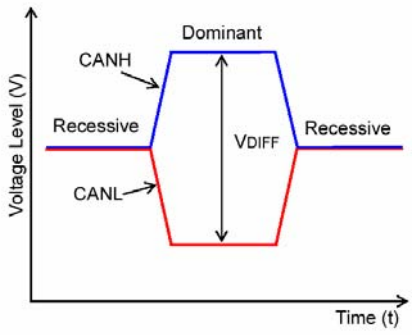
El protocolo de comunicaciones CAN es un protocolo CSMA/CD, lo que quiere decir que, cada nodo de la red puede ver el bus por un periodo de no actividad antes de intentar enviar un mensaje al bus. Además una vez pasado este periodo de inactividad, cada nodo del bus tiene la misma oportunidad de transmitir un mensaje. Si dos nodos en la red comienzan a transmitir al mismo tiempo, los nodos detectaran la colisión y tomaran la acción apropiada.

En el protocolo CAN se utiliza un método de arbitraje que no destruye el bit. Esto significa que el mensaje permanece intacto después de que el proceso de arbitraje se haya completado, incluso si se han detectado colisiones. Todo este arbitraje tiene lugar sin deterioro ni retardo en el mensaje de más lata prioridad.

Para soportar este arbitraje no destructivo se requieren dos cosas:

- El estado lógico debe estar definido como dominante o recesivo
- Y el nodo que transmite debe captar el estado del bus para ver si el estado lógico que esta intentando enviar aparece en el bus.

El CAN define el bit lógico 0 como el dominante y el 1 como recesivo. El estado de bit dominante siempre ganara a uno recesivo, esto es cuanto más bajo sea el valor del campo de arbitraje en el identificador de Mensaje, mas alta es la prioridad del mensaje.



1.3.2.2 Comunicación basada en mensajes

El protocolo CAN esta basado en mensajes no en direcciones. Esto significa que los mensajes no son transmitidos de un nodo a otro basándose en direcciones. En el mensaje esta incluida la prioridad y el contenido de los datos a transmitir. Todos los nodos del sistema reciben cada mensaje transmitido en el bus (y sabrán sí el mensaje fue recibido apropiadamente). Se debe ir a cada nodo del sistema para decidir si el mensaje recibido debería ser descargado inmediatamente o guardado para ser procesado. Un simple mensaje puede ser destinado para un nodo en particular que lo reciba, o muchos nodos incluidos en la red del sistema.

Otra característica muy útil es la capacidad de un nodo de requerir información de otro nodo. Esto se llama Respuesta de Transmisión Remota (RTR). En este caso requiere datos que le sean enviados.

Un beneficio adicional de este protocolo basado en mensajes es que se pueden añadir nodos al sistema sin la necesidad de reprogramar todos los otros nodos para que reconozcan esta acción. Este nuevo nodo comenzara a recibir mensajes desde la red, y basándose en el ID del mensaje, decide si procesar o descargar la información recibida.

1.3.2.3 Descripción de la estructura del CAN basado en mensajes

Se definen 4 tipos diferentes de mensajes o estructuras:

- Data Frame. Se usa cuando un nodo transmite información a cualquier o a todos los nodos del sistema.
- Remote Frame. Es básicamente como el anterior con el bit RTR activado para indicar que requiere respuesta remota.
- Error Frame. Son generados por nodos que detectan alguno de los muchos errores de protocolo definidos en CAN.
- Overload Errors. Generados por nodos que requieren mas tiempo para procesar los mensajes ya recibidos.

Data Frames esta constituido por campos que proveen de información

adicional sobre el mensaje. Dentro del Data Frame nos encontramos con:

- Campo de Arbitraje. Es usado para dar prioridad a los mensajes en el bus. Se define el 0 como el dominante. Esta constituido por 32 bits, en la versión actual, que se dividen en 29 bits identificadores, 1 bit para definir el mensaje como una estructura de datos extendida, un bit SRR que no se usa, y un bit RTR.

  El RTR se usa cuando se requiere información de otro nodo. Si el bit es recesivo entonces el mensaje es del tipo Remote Frame, y si es dominante, el mensaje es del tipo Data Frame.

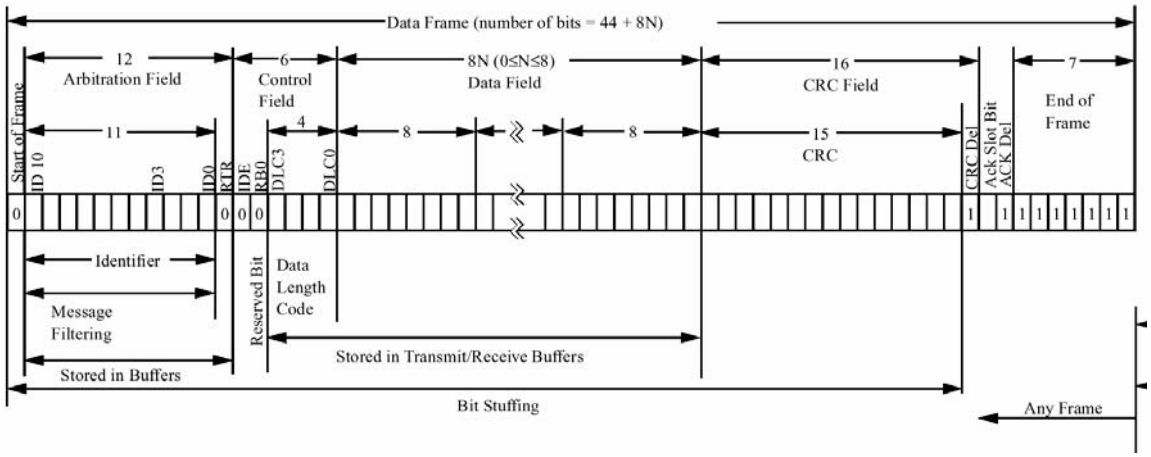
- Campo de Control. Consiste en 6 bits. El MBS es el IDE bit, que especifica que el bus tiene campos de datos extendidos. Es un bit reservado al igual que el RB0. Los cuatro bits LSB son los bits de Codificación de la Longitud de los Datos. Esto determina cuantos bytes de datos se incluyen en el mensaje. El modo Remote Frame no tiene campo de datos.

- Campo de Datos consiste en el número de bits de datos descritos en los bits de longitud de código.

- CRC Field consiste en un campo de 15 bits y un delimitador. Se usa en los nodos que reciben para determinar si han ocurrido errores en la transmisión.

- El campo de conocimiento se utiliza para indicar si el mensaje ha sido recibido correctamente. Cualquier nodo que haya recibido el mensaje correctamente, sin importar si el nodo procesa o descarga la información, pone un bit dominante sobre el bus en el tiempo del bit ACK.

A continuación se muestran los bits que componen es Data Frame de un mensaje en protocolo CAN.



1.3.2.4 Comunicación Robusta y Rápida

La máxima velocidad de comunicación alcanzada hasta ahora es de 1Mbit/seg. A estas velocidades incluso los parámetros mas críticos pueden ser transmitidos en serie sin sucesos importantes. Aparte el CAN tiene una lista de errores que puede detectar la completa integridad de los mensajes.

Los nodos del CAN tienen la habilidad de determinar condiciones de fallos y transición a diferentes modos basado en la importancia de los problemas que se han encontrado. Además tienen la habilidad de detectar pequeñas perturbaciones desde fallos permanentes y modificar su funcionalidad de acuerdo a ello. Pueden pasar de un funcionamiento normal, transmitiendo y recibiendo mensajes normalmente, al apagado completo (bus-off) basado en la importancia de los errores detectados. Esta característica se llama Fault Confinement. Ningún nodo o nodos que han fallado podrán monopolizar todo el ancho de banda de la red porque los fallos serán alojados en los nodos fallidos y estos se apagaran antes de traer la red abajo.

Esto es muy poderoso porque el Fault Confinement garantiza el ancho de banda para informaciones de sistemas críticos:

- Errores Detectados

  - CRC Error. Se calcula un Cyclic Redundancy Check (CRC) de 15 bits por el nodo de transmisión y este valor se transmite al bus. Todos los nodos de la red reciben este mensaje, calculan un CRC y verifican que el valor de CRC coincide. Si no coincide, hay un error de CRC y se genera un Error de Estructura.

  - Acknowledge Error. Indica si al menos un nodo recibió correctamente el mensaje. Si el bit es recesivo entonces ningún nodo lo recibió. Después de que se genera el error de estructura se repite el mensaje original después de un tiempo de intermedio adecuado.

  - Form Error. Si el nodo detecta un error en End of Frame, o Interframe, Ack, CRC, entonces salta el error de forma. El

mensaje se reenvía después de un tiempo prudencial.

- Bit Error. Sucede cuando un transmisor envía un bit dominante y detecta uno recesivo, o viceversa. En este caso no se genera el Bit Error si es algo normal del proceso de Arbitraje. Pero si se genera el mensaje es reenviado.

- Stuff Error. Esto significa que el nivel de bit es colocado en el bus en el periodo de tiempo del bit. CAN es además asíncrono, y el bit stuffing es usado para permitir al nodo que recibe sincronizar por recuperación del reloj desde la tormenta de datos. Los nodos que reciben sincronizan desde recesivas a dominantes transiciones. Si hay mas de 5 bits de la misma polaridad en una fila, CAN automáticamente produce una polaridad diferente en el bit en la tormenta de datos. El nodo que recibe la usara la sincronizarse, pero ignorara el bit de staff para fines de datos. Si, entre el comienzo de Start of Frame y CRC Delimiter, hay 6 bits consecutivos con la misma polaridad, entonces las reglas del bit stuffing se han violado, y ocurre un Staff Error.

- Estados de Error

  - Error Activo. Un nodo de error activo puede tomar parte en el bus de comunicación, incluso enviando un flag de error activo, el cual consiste en 6 bits dominantes consecutivos. La activación del Error Flag viola las reglas del bit de staff y provoca que los otros nodos envíen como respuesta un Error Flag, llamado Error Echo Flag. Esto provoca hasta 12 bits dominantes consecutivos en el bus. 6 del Error Active Flag, y desde 0 hasta 6 mas del Error Echo Flag, dependiendo de cuando cada nodo detecta un error en el bus. Un nodo tiene Error Activo cuando tanto el Transmit Error Counter (TEC) como el Receive Error Counter (REC) están por debajo de 128. El Error Activo es el modo normal de operación, permitiendo al nodo transmitir y recibir sin restricciones.

  - Error Pasivo: Un nodo tiene Error Pasivo cuando tanto el Transmit Error Counter (TEC) como el Receive Error Counter (REC) exceden de 127. Con Error Pasivo no se permite transmitir en el bus un Active Error Flag, pero en su lugar se transmite un Pasive Error Flag, que consiste en 6 bits recesivos. Si solo esta transmitiendo el nodo de Error Pasivo entonces el error pasivo violara las reglas de stuff y el nodo que esta recibiendo responderá con su propio Error Flag. Si no esta solo transmitiendo el nodo de Error Pasivo, o es un receptor, entonces el Passive Error Flag no tendrá efecto en el bus debido a la propia naturaleza recesiva del Error flag. Cuando un nodo de error pasivo transmite y

recibe un bit dominante, debe ver que el bus esta desocupado durante 8 veces mas antes de reconocer el bus como disponible. Después de este tiempo probara a retransmitir.

- Bus Off: Un nodo entra en el estado Bus-Off cuando el Transmiter Error Counter es mayor de 255. En este modo el nodo no puede ni mandar ni recibir, ni conocer mensajes, ni transmitir Error Frames de ninguna clase. Así es como se alcanza un Fault Confinement. Hay un bus de secuencia de recuperación definido por el CAN que permite recuperar un nodo del estado de Bus-Off al estado de Error Activo, y pudiendo ser transmisor de nuevo si la condición de fallo desaparece.

1.3.2.5 CONCLUSION

El protocolo CAN se ha optimizado para sistemas que necesitan transmitir y recibir relativamente pequeñas cantidades de información entre los otros nodos de la red. CSMA/CD permite a cada nodo tener la misma oportunidad de acceso al bus, y pequeñas colisiones.

Desde que el protocolo se basa en mensajes y no en direcciones, todos los nodos del bus pueden recibir y conocer cada mensaje independientemente de si lo necesitan o no. Esto permite al bus poder operar de nodo a nodo o en formato de mensajes para múltiples nodos sin tener que enviar diferentes tipos de mensajes.

La transmisión de mensajes robusta y rápida con confinamiento de fallos es además un plus para el CSN porque los nodos con fallos son automáticamente alejados del bus no permitiéndoles hacer caer la red. Esto garantiza efectivamente que el ancho de banda este siempre disponible cuando se transmiten mensajes críticos.

1.3.3 PROTOCOLO SPI

Una interfaz SPI puede constar de uno o más maestros y varios esclavos, los cuales se comunican a través de tres líneas; una para datos de ida, otra para datos de retorno, y un reloj impuesto por el master. Además cada

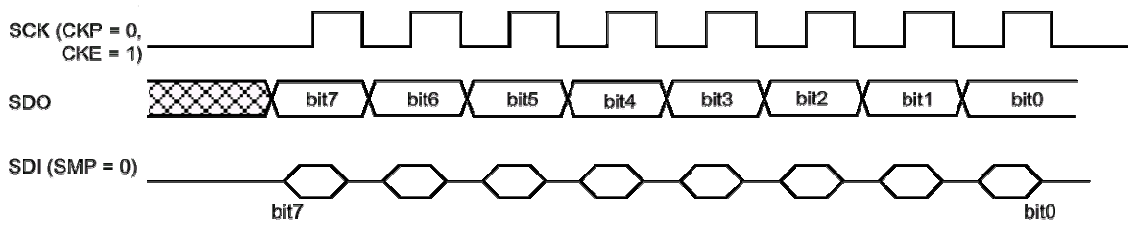
esclavo tiene una línea SS la cual selecciona con cual esclavo se va a realizar la transferencia, e indica el comienzo de la misma en algunos modos de funcionamiento.

Existen 4 configuraciones para el reloj impuesto por el master (SCK). Estas son indicadas mediante dos parámetros de tipo binario CPOL (Clock POLarity) y CPHA (Clock PHAse).

El sistema funciona como un shift-register, en el cual una vez cargados los datos, estos van siendo transmitidos a través de la línea MISO (Master In Slave Out; conectada al MSB), mientras que se van ingresando datos recibidos a través de la línea MOSI (Master Out Slave In; conectada al LSB).El reloj SCK transmitido por el master es quien indica al esclavo cuándo dejar su salida estable para que pueda ser leída por el master y a su vez grabar el dato que le llega de este, y cuándo mover los datos en el shift- register.

Para el modo de transferencia, CPHA = 1 CPOL = 0, comienza cuando el reloj SCK (activo por nivel alto) pasa a uno, en ese momento el esclavo debe poner el MSB en la línea MISO, a continuación, en el flanco de bajada de SCK se guarda el MSB del dato a recibir. Luego en el flanco de subida de SCK se rota el shift-register.

Luego de varios ciclos de reloj SCK los datos recibidos son guardados en un registro y SPIF sube. La transferencia termina cuando el reloj SCK vuelve a su estado inactivo.



2 HARDWARE

El presente proyecto consta fundamentalmente de dos placas denominadas microcontroladoras cuyas descripciónes detalladas y funcionalidades serán comentadas a continuación. Además de estas dos placas se dispone de otras cuatro que se emplean para funciones auxiliares como programar las aplicaciones en cada placa microcontroladora o generar ciertas entradas y medir sus correspondientes salidas. Pasamos por tanto a ver cada una de dichas placas con detalle.

2.1 PLACAS MICROCONTROLADORAS

Bajo esta definición se encuentran las dos placas que verdaderamente constituyen el sistema objeto de este proyecto. Si bien la circuitería de ambas es muy similar; la funcionalidad de cada una de ellas las diferencia claramente. A continuación se estudiará cada una por separado empezando por la placa de la cabeza del robot y terminando por la de la cabina del mismo. Se detallarán tanto las distintas señales de entrada y salida vinculadas a las placas, como la circuitería que presenta cada una de ellas para el tratamiento de estas señales.

2.1.1 PLACA CABEZA

Su función es la de interfaz CAN bus – sensores/actuadores; para el robot forestal (ROFOR) de la empresa Servicios Forestales.

Como ya se ha dicho, el robot se compone de una cabina de mando y un brazo mecánico; en el extremo del cual tiene un cabezal robotizado (procesador forestal). Esta placa está situada en el cabezal de dicho robot.

El objeto de la placa es la comunicación con la cabina a través del CAN Bus, de la información recibida de los sensores/actuadores del cabezal robotizado. Para ello tiene que recibir las entradas procedentes de la placa de la cabina y enviarlas al cabezal; así como recibir la información del cabezal y enviarla a la cabina. Todo esto se realizará con el bus de campo CAN.

Las entradas/salidas que tiene que tratar la placa del cabezal son las siguientes:

- Comunicación con los sensores/actuadores.
    - 15 salidas digitales en el rango de 0 a 24 V. Estas deben estar optoaisladas
    - 4 entradas digitales 0-24 V
    - 2 entradas digitales 0-24 V a 500 hz
    - 2 entradas digitales 0-24V a 2 Khz. procedentes de dos encoders desfasadas 90 grados
    - 3 entradas analógicas 0-10 V
    - 2 salidas PWM
    - Se dispone de una alimentación de 24 V no estabilizada
- Comunicación mediante CAN-Bus con la cabina de mando del robot. A la cabina le son enviadas, previo tratamiento, todas las entradas procedentes de los sensores y a su vez la cabina suministra las salidas para los actuadotes.

El corazón del sistema es un microcontrolador PIC18F458 el cual llevará las labores de control y manejo de los datos. Es un dispositivo de la gama alta del fabricante Microchip. La elección de este tipo de microcontrolador está basada en criterios como una alta calidad, un bajo coste y un excelente rendimiento. Así como una facilidad de conseguir todo tipo de herramientas de desarrollo, gratuitas, incluso programadores, emuladores, etc. También important resuta la circunstancia de que sepuede disponer de muestras gratuitas de dicho integrado con el consiguiente ahorro económico a la hora de abordar el proyecto.

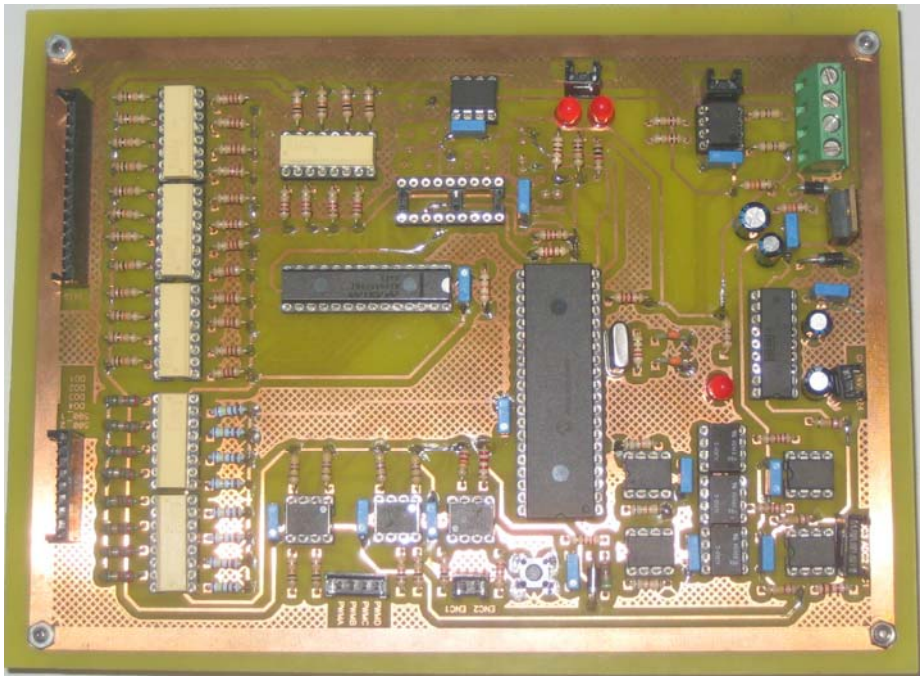
Los datos recibidos deben ser previamente adaptados a los niveles de tensión del microcontrolador y en el caso de las entradas/salidas analógicas debe procederse a una conversión digital-analógica, analógico-digital según estemos tratando salidas o entradas respectivamente. También el sistema

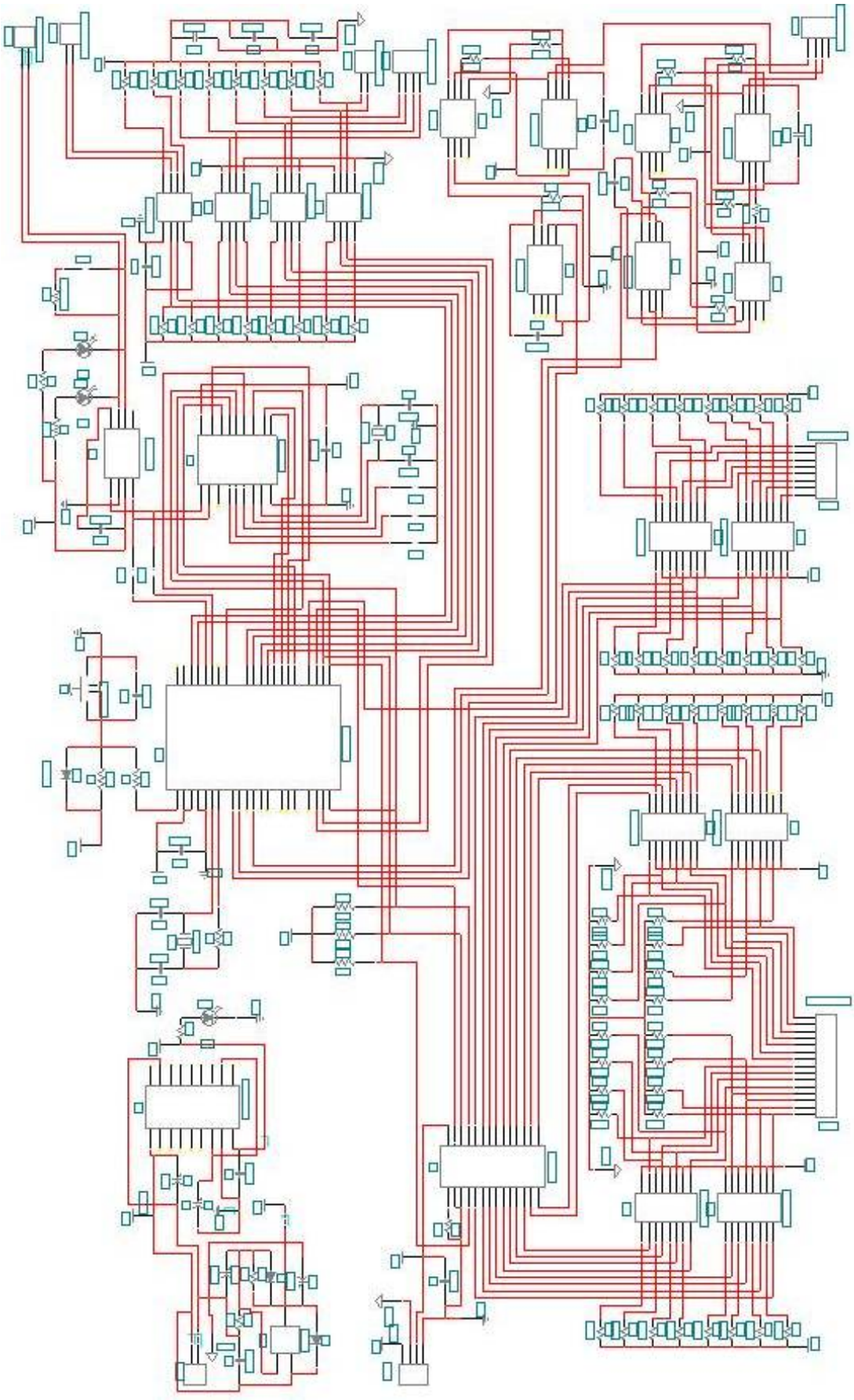
necesita un aislamiento óptico total del exterior para su protección contra fallos. Por ello el microcontrolador requiere de la presencia de periféricos externos para la consecución de los objetivos. Estos periféricos son:

- 6 Optoacopladores digitales PS 2501-4
- 3 Optoacopladores lineales TIL 300
- 4 Amplificadores operacionales duales Rail to Rail
- 1 Expansor de E/S digitales MAX-6957
- 1 Convertidor 24-5 V optoacoplado DCR022405
- 1 Regulador de tensión 24-20V LM317
- 1 Interfaz CAN-Bus MCP2551

Para las comunicaciones entre el PIC y los periféricos se utiliza el bus SPI. Esto da al sistema una gran versatilidad al poderse añadir multitud de periféricos sin consumir las entradas/salidas del PIC.

Pasamos ya a la descripción detallada de cada uno de los circuitos que componen la placa así como de los integrados involucrados en dichos circuitos. A continuación podemos ver una figura de la placa así como el esquemático completo de la misma:





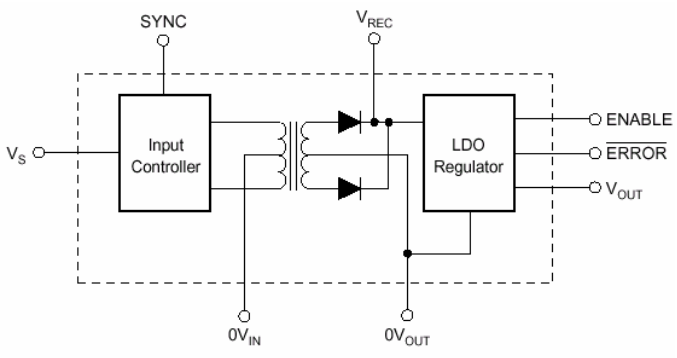
2.1.1.1 Circuito alimentación

El circuito de alimentación realiza en esta placa una doble función. Por un lado se encarga de la adaptación de la tensión de entrada a los niveles de tensión necesarios para el funcionamiento del sistema. La única fuente de alimentación que nos proporciona el robot es una batería de 24V, por lo que nuestro circuito de alimentación transforma los 24V de entrada en 5V estabilizados de salida.

Por otro lado cabe destacar que una de las principales características de esta placa es que está optoaislada de las entradas y salidas del sistema. Por tanto la fuente de alimentación también se aísla del exterior incorporando un regulador de tensión aislado galvanicamente.

Así pues el circuito de alimentación aparte de suministrar la tensión de 5V proporciona un aislamiento galvánico de la entrada con la salida, bajo ruido, y gran exactitud. Se disponen por tanto de dos tierras independientes, una para cada tensión.

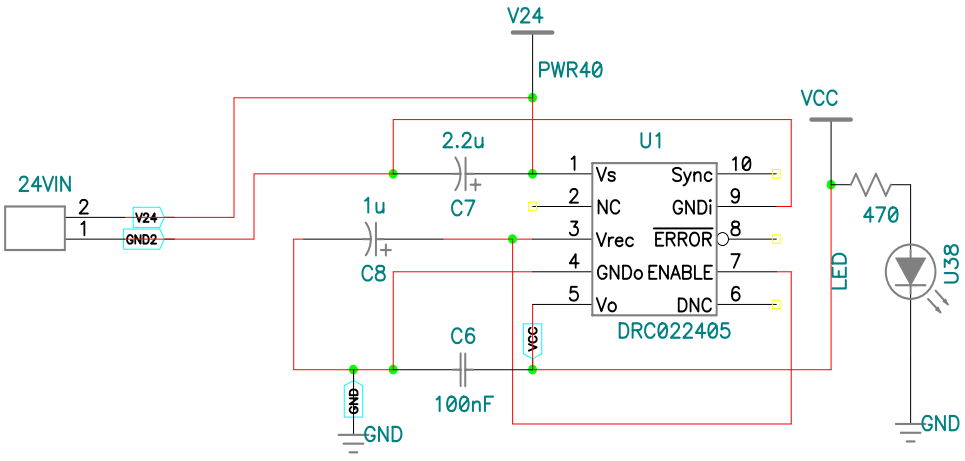
El dispositivo empleado para ello es el convertidor optoacoplado DCR022405 del fabricante Texas Instruments. Es un integrado que reúne ambas características. Ofrece aislamiento galvánico y salida regulada de tensión a partir de una fuente no regulada. La tensión de entrada puede variar entre 21.6V y 26.4V, dando una potencia nominal de salida de 2W, con una tensión regulada de 5V, y una corriente mínima de 400mA, que nos sirve para alimentar a toda la circuitería interna de la placa. El esquema interno del integrado es el siguiente:



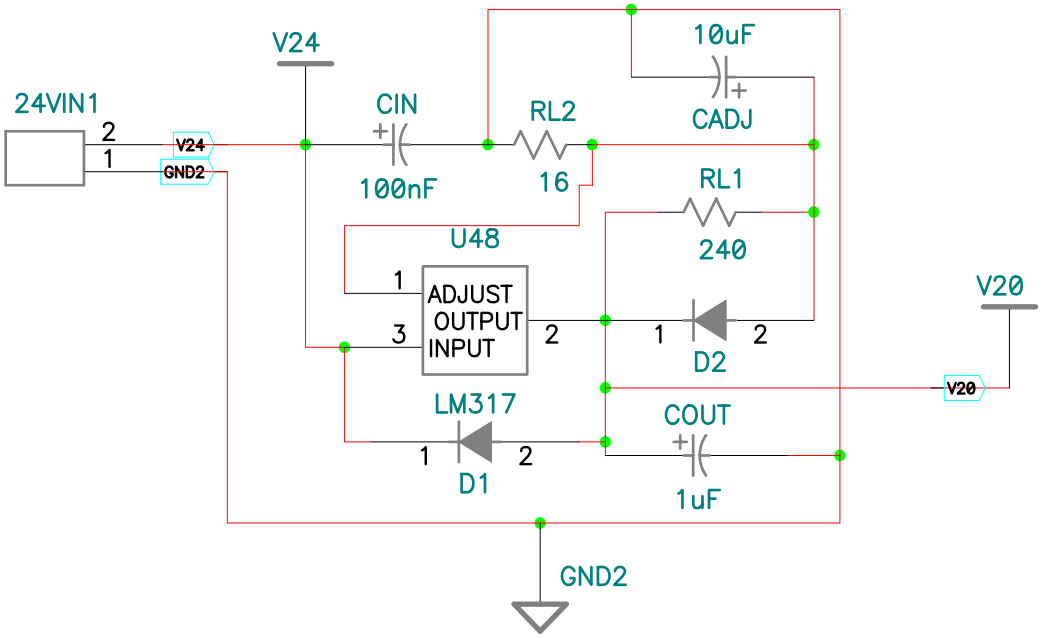
A continuación se verá una descripción de los pines del integrado:

- El pin SYNC se utiliza para sincronizar varios convertidores en el caso de que la carga requiera más de 2W de potencia. En nuestro diseño debido a que el consumo es menor a 2W se deja al aire. También puede ser usado para activar o desactivar el DCR022405. Poniéndolo a nivel bajo estará desactivado.
- El pin ENABLE sirve para tener un control sobre el encendido o apagado de la tensión de salida. Si se pone bajo el DCR022405 estará deshabilitado. Al no requerir esta característica se ha conectado a la tensión de 5V, con lo que esta habilitado permanentemente.
- El pin ERROR proporciona un flag de buen funcionamiento del dispositivo. Este se activará cuando la regulación no funcione adecuadamente.
- El pin Vrec es una salida que proporciona una onda rectificada con un rizado de 800 Khz.
- Los pines Vs y 0Vin son respectivamente la entrada de tensión y la tierra de ésta.
- Los pines Vout y 0Vout son respectivamente la salida de tensión y la tierra de ésta.

En el esquemático siguiente se muestra el conexionado y los diferentes elementos pasivos necesarios para su funcionamiento. Se utiliza un condensador de bajo ESR, de valor 2.2 uF, entre la entrada de tensión y la tierra de ésta para estabilizar la tensión de entrada. Otro de valor 1uF y bajo ESR, entre Vrec y la tierra de 5V para reducir el rizado. El condensador de 100nF y bajo ESR, conectado entre la tensión de salida y su tierra se utiliza para reducir el ruido en la salida.



Para no sobrecargar demasiado el DCR022405 se utiliza un regulador de tensión ajustable LM317 que alimenta a los amplificadores de las entradas/salidas analógicas. Este se alimenta con una tensión de 24V y proporciona en la salida una tensión estabilizada de 20V siendo capaz de proporcionar una intensidad máxima de 1.5A. El esquema sería el siguiente:



Al igual que el DCR éste requiere la presencia de los siguientes elementos pasivos:

- Cin, 100nF, se usa para estabilizar la tensión de entrada
- Cout, 1uF: Mejora la respuesta transitoria. No afecta a la estabilidad.
- Cadj, 10uF: Sirve para eliminar el rizado. Previene su amplificación a más alto voltaje de salida
- R2 16Ω y R1 240Ω dan la tensión de salida a partir de la siguiente formula:

$$Vo = 1.25 \times \left(1 + \frac{R2}{R1}\right) + Iadj \times R2$$

  El termino Iadj x R2 no se tiene en cuenta para los cálculos ya que Iadj es del orden de 50uA.

- D1 y D2 protegen al regulador en caso de que este tenga un cortocircuito a tierra. Evita que los condensadores Cadj y Cout se descarguen a través de los caminos de baja impedancia del integrado.

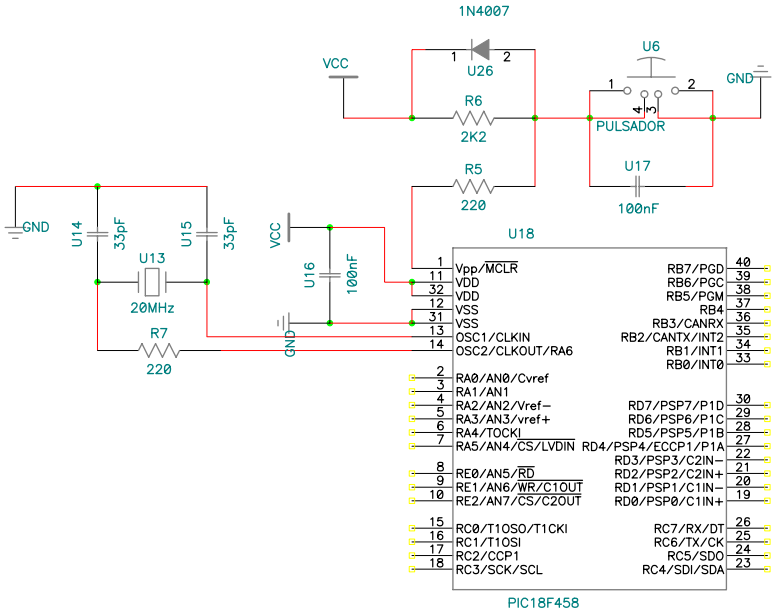
2.1.1.2 Circuito de reset y circuito oscilador

El circuito de reset manual se ha diseñado para reinicializar el microcontrolador cuando sea necesario, sin necesidad de desconectar la alimentación. Está formado por un interruptor pulsador, conectado al negativo de la alimentación, y en paralelo con un condensador de 100nF, que se usa para estabilizar y eliminar los ruidos que introduce el pulsador. Este circuito también posee una resistencia de Pull-Up, y un diodo 1N4007 de protección.

Todos los microcontroladores disponen de un circuito oscilador que genera una onda cuadrada de alta frecuencia, que configuran los impulsos de reloj usados en la sincronización de todas las operaciones del sistema. Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo.

Estos componentes exteriores son un cristal de cuarzo de 20Mhz, dos condensadores cerámicos de 33pF y una resistencia limitadora de 220Ω, para la señal de oscilación de salida. Con esta configuración, el periodo de la señal de reloj es de T_{RELOJ} =50ns.

Para terminar se muestra a continuación un esquemático con los circuitos del oscilador y reset manual y su conexionado al PIC.



2.1.1.3 Pic 18F458

El PIC18F458 constituye el elemento fundamental del sistema. Es por ello que le serán dedicados a continuación una serie de apartados a fin de conocer el funcionamiento y configuración del mismo. Se tendrá primero una visión general de todas las posibilidades que ofrece y a continuación se detallarán aquellos módulos y aquellos aspectos que se han empleado en la realización del presente proyect, desechando los que no se usarán.

2.1.1.3.1 Descripción PIC. Funcionalidad de los pines

Este dispositivo pertenece a la subfamilia de microcontroladores PIC de MICROCHIP de la gama alta, que se identifica por tener como memoria de programa una de tipo FLASH, que se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. Funciona como una ROM (memoria de solo lectura) y una RAM (memoria de acceso aleatorio) 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 le memoria EEPROM, tolerando más ciclos de escritura / borrado que esta. La alternativa FLASH está recomendada cuando se precisa gran cantidad de memoria no volátil.

Este microcontrolador está encapsulado en 40 pines, por lo que posee hasta un máximo de 33 líneas de entrada / salida.

Los principales recursos que posee el microcontrolador PIC18F458 son:

- Procesador de arquitectura RISC avanzada
- Ampliación de memoria de programa externa de hasta 2 Mbyte con unidades de almacenamiento, tales como memorias EEPROM o FLASH
- Ampliación de memoria de datos de hasta 4 Kbytes
- A 40 Mhz 10 millones de instrucciones por segundo (MIPs). A la velocidad actual de 20 Mhz se ejecutan por tanto 5 MIPs
- Juego de 70 instrucciones. Cada instrucción es una palabra de 16

bits de longitud. Todas ellas se ejecutan en un ciclo de instrucción, menos las de salto que tardan dos

- Frecuencia máxima de funcionamiento de 40Mhz
- Ciclo de instrucción de 100 ns a 40 Mhz y 200 ns a 20 Mhz
- Multiplicador 8x8 en hardware. Gracias a el las operaciones de multiplicación entre dos bytes se realizan en un solo ciclo de instrucción
- Niveles de prioridad para las interrupciones
- 32 Kbytes, 16384 palabras de 16 bits para la Memoria de Código, tipo FLASH
- 1536 bytes de Memoria de Datos RAM
- 256 bytes de Memoria de Datos EEPROM
- 14 fuentes de interrupción internas y externas
- Pila con 31 niveles
- Modos de direccionamiento directo, indirecto y relativo
- Perro Guardián (WDT)
- Código de protección programable
- Modo SLEEP de bajo consumo
- Programación serie a través de dos pines
- Voltaje de alimentación comprendido entre 2 y 5.5 V
- Bajo consumo (menos de 2 mA a 5 V y 5 Mhz)
- Propiedades de los periféricos:

    - Alta corriente recepción/fuente 25mA/25mA
    - 3 pines de interrupción externa
    - Timer 0 : Temporizador-Contador de 8 bits/16 bits
    - Timer 1: Temporizador-Contador de 16 bits
    - Timer 2: Temporizador-Contador de 8 bits con registro de periodo (para el PWM) de 8 bits
    - Dos módulos de Captura-Comparación-PWM
    - Resolución PWM de 8 ó 10 bits. Max. Frecuencia para 8 bits 156 Khz y 39 Khz para 10 bit

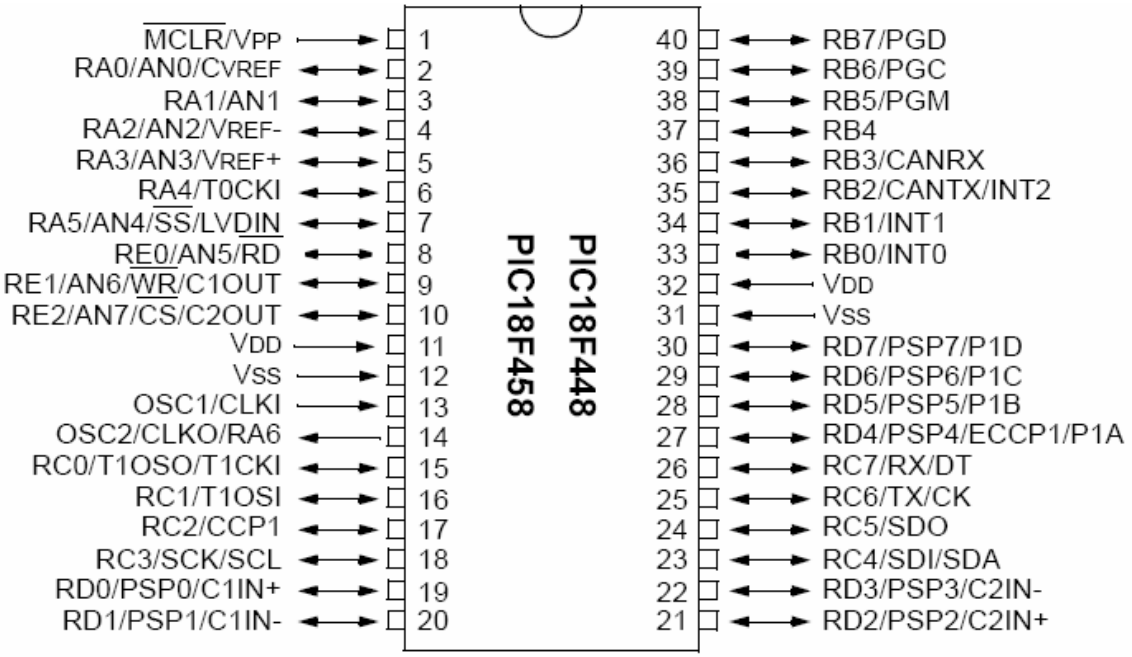
- Modulo PWM avanzado para control de motores. Permite añadir tiempos muertos, seleccionar la polaridad del PWM y elegir entre 1,2 o 4 PWM, con la excepción que solo habrán dos PWM totalmente independientes, los otros 3 pines dependerán de uno principal
- Convertidor A/D de 10 bits de precisión
- Modulo de comunicación Serie Síncrona (MSSP), con modo SPI e I2C. Soporta los 4 modos SPI a una velocidad de 5 Mhz con oscilador de 20 Mhz e I2C a 1 Mb/s
- Interface de comunicación Serie SCI, también llamado USART
- Puerto Paralelo Esclavo (PSP)
-

- Características analógicas avanzadas:
  - 8 convertidores analógicos digitales de 10 bit de precisión con conversión disponible durante el modo sleep
  - Comparador analógico con programación de entrada y salida
  - Modulo comparador con voltaje de referencia
  - Detector de bajo voltaje programable que dispone de interrupción en caso de detección de bajo voltaje (BROWN OUT RESET)

- Propiedades de CAN bus:
  - Cumple con los test CAN ISO
  - Velocidad máxima de 1 Mb/s
  - Cumple las especificaciones del CAN 2.0 B

A continuación, y para terminar con esta breve descripción sobre las facilidades básicas que ofrece el PIC; podemos ver un gráfico con el pinout del dispositivo así como una descripción de cada uno de los 40 pines de que consta el PIC. Se han distribuido para ello los pines de acuerdo a los puertos a los que cada uno de ellos pertenece.Asimismo se ha indicado también los que

son de propósito general:



PINES DE PROPÓSITO GENERAL

- **OSC1/CLKIN (Pin 13)** ⇒ Entrada del cristal de cuarzo o del oscilador externo
- **OSC2/CLKOUT/RA6 (Pin 14)** ⇒ Salida del cristal de cuarzo. En modo RC el pin **OSC2** saca la cuarta parte de la frecuencia que se introduce por el pin **OSC1**, que determina el ciclo de instrucción. También puede ser utilizado como un pin de proposito general de entrada/salida, particularmente el PIN 7 del puerto A
- **VSS (Pines 12 y 31)** ⇒ Conexión a tierra
- **VDD (pines 11 y 32)** ⇒ Entrada de la alimentación positiva
- **MCLR#/VPP (pin 1)** ⇒ Entrada de RESET o entrada del voltaje de programación (voltaje alto en el modo test)

PINES DE LA PUERTA A

- **RA0/AN0/Cvref (pin 2)** ⇒ Este pin puede funcionar como una línea digital de E/S o como una entrada analógica (canal 0) del convertidor A/D. También puede funcionar como pin de salida del voltaje de referencia del comparador
- **RA1/AN1 (pin 3)** ⇒ Este pin puede funcionar como una línea digital de E/S o como una entrada analógica (canal 1) del

convertidor A/D

- **RA2/AN2/V_{REF-} (pin 4)** ⇒ Este pin a parte de las dos funciones anteriores, también puede funcionar como entrada del voltaje de referencia negativo, necesario para la realización de algunas aplicaciones con el convertidor A/D

- **RA3/AN3/V_{REF+} (pin 5)** ⇒ Este pin puede funcionar como una E/S digital, como entrada analógica (canal 3) o como un voltaje de referencia positivo de entrada para el convertidor A/D

- **RA4/T0CKI (pin 6)** ⇒ Este pin puede funcionar como una E/S digital o como una entrada de reloj externo para el Timer 0

- **RA5/AN4/SS#/LVDIN (pin 7)** ⇒ Este pin puede funcionar como una E/S digital, como una entrada analógica (canal 4) o como selección de la puerta serie síncrona para su funcionamiento como esclavo. Puede funcionar como entrada de detección de bajo voltaje

PINES DE LA PUERTA B

- **RB0/INT (pin 33)** ⇒ Este pin puede funcionar como una E/S digital o como una entrada de petición de interrupción externa, para el control de diversos periféricos externos

- **RB1/INT1 (pin 34)** ⇒ Este pin puede funcionar como una E/S digital o como una entrada de petición de interrupción externa, para el control de diversos periféricos externos

- **RB2/CANTX/INT2 (pin 35)** ⇒ Este pin puede funcionar como una E/S digital, como una entrada de petición de interrupción externa, para el control de diversos periféricos externos y como pin de transmisión de datos a periférico CAN bus

- **RB3/CANRX (pin 36)** ⇒ Este pin puede funcionar como una E/S digital o como pin de recepción de datos de periférico CAN bus

- **RB4 (pin 37)** ⇒ Este pin solo puede funcionar como E/S digital

- **RB5/PGM (pin 38)** ⇒ Este pin puede funcionar como una E/S digital o como la entrada del voltaje bajo para programación

- **RB6/PGC (pin 39)** ⇒ Este pin puede funcionar como una E/S digital o como la entrada de la señal de reloj en la programación serie del microcontrolador

- **RB7/PGD (pin 40)** ⇒ Este pin puede funcionar como una E/S digital o como la entrada de datos en la programación serie del microcontrolador

PINES DE LA PUERTA C

- **RC0/T1OSO/T1CKI (pin 15)** ⇒ Este pin puede funcionar como una E/S digital, como una salida de reloj desde el Timer 1 o como una entrada de reloj externa para el Timer 1
- **RC1/T1OSI (pin 16)** ⇒ Este pin puede funcionar como una E/S digital, como una entrada al oscilador del Timer 1
- **RC2/CCP1 (pin 17)** ⇒ Este pin puede funcionar como una E/S digital o como:
  - La entrada del módulo de captura nº1
  - La salida del módulo de comparación nº1
  - La salida del módulo de PWM nº1
- **RC3/SCK/SCL (pin 18)** ⇒ Este puede funcionar como una E/S digital o como:
  - La señal de reloj para la comunicación serie síncrona en modo SPI
  - La señal de reloj de para la comunicación serie síncrona en modo I2C
- **RC4/SDI/SDA (pin 23)** ⇒ Este puede funcionar como una E/S digital, o como:
  - La Entrada de datos en modo SPI
  - La E/S de datos en modo I2C
- **RC5/SDO (pin 24)** ⇒ Este pin puede funcionar como una E/S digital o como la salida de datos en modo SPI
- **RC6/TX/CK (pin 25)** ⇒ Este pin puede funcionar como una E/S digital o como:
  - La pin del transmisor del USART asíncrono
  - El reloj del USART síncrono

- **RC7/RX/DT (pin 26)** ⇒ Este pin puede funcionar como una E/S digital o como:

  - La pin del receptor del USART asíncrono
  - La línea de datos del USART síncrono

PINES DE LA PUERTA D

- **RD0/PSPO/C1IN+** ⇒ Puede funcionar como E/S digital, como pin 1 del puerto paralelo o como entrada positiva analógica del comparador 1
- **RD1/PSP1/C1IN-** ⇒ Puede funcionar como E/S digital, como pin 2 del puerto paralelo o como entrada negativa analógica del comparador 1
- **RD2/PSP2/C2IN+** ⇒ Puede funcionar como E/S digital, como pin 3 del puerto paralelo o como entrada positiva analógica del comparador 2
- **RD3/PSP3/C2IN-** ⇒ Puede funcionar como E/S digital, como pin 4 del puerto paralelo o como entrada negativa analógica del comparador 2
- **RD4/PSP4/ECCP1/P1A** ⇒ Este pin puede funcionar como una E/S digital, como pin 5 del puerto paralelo o como:

  - La entrada del módulo de captura ECCP
  - La salida del módulo de comparación ECCP
  - La salida 1 del módulo de PWM ECCP

- **RD5/PSP5/P1B**⇒ Puede funcionar como E/S digital, como pin 6 del puerto paralelo o como salida 2 del modulo PWM ECCP
- **RD6/PSP6/P1C**⇒ Puede funcionar como E/S digital, como pin 7 del puerto paralelo o como salida 3 del modulo PWM ECCP
- **RD7/PSP7/P1D**⇒ Puede funcionar como E/S digital, como pin 8 del puerto paralelo o como salida 4 del modulo PWM ECCP

PINES DE LA PUERTA E

- **RE0/RD#/AN5 (pin 8)** ⇒ Este pin puede funcionar como una E/S digital o:
  - Como la señal de lectura para la puerta paralela esclava.
  - Como el canal 5 del convertidor A/D.

- **RE1/WR#/AN6/C1OUT (pin 9)** ⇒ Este pin puede funcionar como una E/S digital, como salida analógica del comparador 1 o:
  - Como la señal de escritura para la puerta paralela esclava.
  - Como el canal 6 del convertidor A/D.

- **RE2/CS#/AN7/C2OUT (pin 10)** ⇒ Este pin puede funcionar como una E/S digital, como salida analógica del comparador 2 o:
  - Como la señal de activación /desactivación de la puerta paralela esclava.
  - Como el canal 7 del convertidor A/D.

2.1.1.3.2 Módulo CAN

El CAN bus es una interfaz serie para la comunicación con otros periféricos tales como sensores o microcontroladores. Este protocolo fue diseñado para permitir comunicaciones dentro de ambientes ruidosos.

El PIC18F458 implementa en su modulo CAN los protocolos CAN 2.0 A/B dentro de las especificaciones de BOSCH. Soporta CAN 1.2, 2.0 A, 2.0 B pasivo y 2.0 B. El módulo contiene todo el protocolo del CAN bus con lo que no tenemos que preocuparnos de detalles tales como el formato del mensaje, bits redundantes, mensajes erróneos, etc. Dispondremos de los registros necesarios para la detección de errores y la comunicación de los diferentes tipos de mensajes del CAN bus.

Las características de este módulo son las siguientes:

- Cumple con las conformidades del test CAN ISO
- Permite mensajes estándar y extendidos
- Longitud de datos en el mensaje desde 0 hasta 8 bytes
- Velocidad máxima programable hasta 1 Mb/s
- Soporta tramas remotas, es decir, es capaz de recibir mensajes en los que se le pide al nodo que envíe una respuesta específica.
- Dos buffers de recepción con prioridades.
- 3 buffers de transmisión con prioridad programable.
- 2 filtros de admisión para el buffer de recepción 1 y 4 para el buffer de recepción 2
- 2 máscaras, una por cada buffer de recepción. Las máscaras sirven para saber que bit del identificador debe ser testeado por los filtros.
- Un mensaje en el CAN bus puede sacar al PIC del modo SLEEP.
- Detección de errores de transmisión y recepción mediante interrupción específica para tal fin.
- Fuente de reloj programable
- Modo bajo consumo

El módulo CAN usa los pines RB2/CANTX/INT2, configurado como salida, y RB3/CANRX, configurado como entrada.

Nos encontramos con 6 modos de funcionamiento:

- Modo Configuración: En este modo se programa la forma en la que queremos que funcione el CAN bus.
- Modo Deshabilitado: Cuando un nodo esta falla repetidas veces entra en este modo.
- Modo de operación normal
- Modo de escucha únicamente
- Modo bucle: permite la transmisión de mensajes de los buffers de transmisión a sus propios buffers de recepción. Se utiliza como test para verificar el buen funcionamiento del módulo.
- Modo de reconocimiento de errores: En este modo se reciben todos los mensajes validos y no validos en los buffers de recepción.

El PIC18F458 necesita de la interfaz CAN MCP2551 para la conexión física con los dos hilos que componen el bus. Esta interfaz transformará los niveles lógicos TTL de salida de los pines del PIC a los valores de tensión de la comunicación CAN bus.

2.1.1.3.3 Módulo de Comunicación Serie Síncrona

La comunicación serie es una forma muy apreciada de comunicación digital entre sistemas y circuitos integrados, debido al reducido numero de líneas que se necesitan. Este microcontrolador lo lleva implementado, proporcionando un excelente interfaz de comunicación con diversos periféricos.

El módulo MSSP (Master Synchronous Serial Port) admite dos de las alternativas más usadas en la comunicación serie síncrona:

- **SPI** (Serial Peripheral Interface): Este tipo de comunicación la utilizan principalmente las memorias (RAM y EEPROM) y utiliza tres líneas para ello. Admite una velocidad de 10 Mhz a 40 Mhz y los cuatro modos de funcionamiento de este protocolo.
- **I2C** (Inter-Integrated Circuit): Este tipo de comunicación implantada por la marca PHILIPS, tiene como principal característica la utilización de solo dos hilos y, recientemente, ha conseguido una importante implantación en la comunicación de circuitos integrados como memorias, controladores, relojes, convertidores, etc. Velocidad máxima de transferencia de 1 Mb/s.

Para nuestro sistema usaremos la comunicación SPI, ya que nos permite conectar periféricos que soportan más velocidad de comunicación.

Este modo de funcionamiento permite la transferencia de datos de 8 bits en serie, que pueden ser transmitidos y recibidos de forma síncrona y simultánea. Esta comunicación utiliza tres líneas del microcontrolador:

- **SDO** (Serial Data Out): Esta pin es la salida de datos en serie.
- **SDI** (Serial Data In): Esta pin es la entrada de datos en serie.
- **SCK** (Serial Clock): Esta pin es el reloj de sincronización.

El microcontrolador trabaja en modo maestro hay que programar la pin **RC3/SDO** como salida, la pin **RC4/SDI** como entrada, y la pin **RC5/SCK** también como salida.

2.1.1.3.4 Módulo ADC

Este microcontrolador posee un convertidor A/D de 10 bits de resolución y 8 canales de entrada.

La resolución que tiene cada bit procedente de la conversión tiene un valor que es función de la tensión de referencia V_{REF}, de acuerdo con la siguiente formula:

$$\text{Resolución}=(V_{REF+}-V_{REF-})/1024=V_{REF}/1024$$
$$5))/1024=V_{REF}/1024$$

Por lo que si tomamos como tensión de referencia positiva la V_{DD} y la de referencia negativa la tierra, la resolución seria de 4,8 mV/bit. Por lo tanto, a la entrada analógica de 0 voltios le corresponde una digital de 00 0000 0000 y para la de 5 voltios una de 11 1111 1111. La tensión de referencia determina los límites máximo y mínimo de la tensión analógica que se puede convertir. El voltaje diferencial mínimo es de 2 voltios.

Este convertidor A/D utiliza la técnica de aproximaciones sucesivas, y es el único dispositivo interno del microcontrolador que puede funcionar en modo de reposo (**SLEEP**), para ello el reloj del convertidor deberá conectarse al oscilador RC interno.

En la aplicación de este proyecto usaremos solo 4 canales de entrada que son:

- Pin RA0/AN0/C_{VREF} como canal analógico 1
- Pin RA1/AN1 como canal analógico 2
- Pin RA2/AN2/Vref- como canal analógico 3

- Pin RA3/AN3/Vref+ como referencia de tensión positiva para la conversión.
- Pin RA5/AN5 como canal analógico 4

2.1.1.3.5 Módulo de comparación y PWM

El microcontrolador PIC18F458 dispone de 2 de estos módulos, denominados CCP1 y CCP2 que son idénticos excepto a la modalidad de Disparo Especial. Estos módulos pueden realizar tres funciones:

- Captura: Una pareja de registros de uno de estos dos módulos captura el valor que tiene el TMR1 cuando ocurre un evento especial en la pin RC2 (para el módulo CCP1) o en la pin RC1 (para el módulo CCP2).
- Comparación: Se compara el valor de 16 bits del TMR1 con otro valor cargado anteriormente en una pareja de registros de un módulo CCPx y cuando coinciden se produce un evento en la/s pin/s RC2 y/o RC1.
- Modulación de anchura de pulsos (PWM): En esta función se realiza dentro del intervalo del periodo de un impulso controlando la anchura en la que la señal está a nivel alto. Esta técnica es muy útil para el control de motores de corriente continua.

Para nuestro proyecto solo utilizaremos el modo de funcionamiento PWM el cual detallo a continuación:

Con este modo de funcionamiento, se consiguen impulsos lógicos cuya anchura del nivel alto es de duración variable, que son de enorme aplicación en el control de motores de corriente continua, de triacs, etc. Aquí se aplicará sobre bobinas de electroválvulas proporcionales.

Las pines **RC2 y RD0** están configuradas como salidas y basculan entre los niveles lógicos 0 y 1, a intervalos de tiempo variables. Lo que se intenta es obtener un impulso cuyo nivel alto tenga una anchura variable (Duty Cycle) dentro del intervalo del periodo de trabajo.

Para lograr este basculado se usa un comparador que pone a 1 (Set) un

flip-flop cuando el valor del registro **PR2** coincide con la parte alta del TMR2, momento en el que el TMR2 toma el valor 00h. Luego el flip-flop se resetea cuando otro comparador detecta la coincidencia del valor existente en **CCPR1H** con el de la parte alta del TMR2. De esta manera, variando los valores que se cargan en **PR2** y en **CCPR1L** (que luego se traspasa al **CCPR1H**) se varía el intervalo de tiempo en el que la pin está a 1 y a 0.

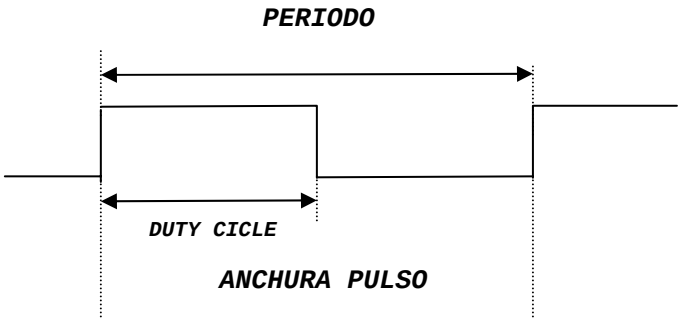
Cuando se trabaja con una precisión de 10 bits, los 2 bits **CCP1CON<5:4>** se concatenan con los 8 del registro **CCPR1L** y, de la misma forma, los 8 bits de más peso del TMR2 se concatenan con los dos bits de menos peso del reloj interno, haciendo que TMR2 cuente cada T_{OSC} en vez de cada $4 \cdot T_{OSC}$.

El tiempo que dura el período de la onda depende del valor cargado en el registro **PR2**, según la siguiente formula:

$$Periodo=[(PR2)+1] \cdot 4 \cdot T_{OSC} \cdot Predivisor\ TMR2$$

El tiempo que la pin de salida está a nivel alto, que es la anchura del impulso, depende del contenido cargado en **CCPR1L** y de los bits 5 y 4 del registro **CCP1CON**, cuando se trabaja con una precisión de 10 bits.

$$Anchura\ Impulso=(CCPR1L:CCP1CON<5:4>) \cdot T_{OSC} \cdot Predivisor\ TMR2$$



2.1.1.4 Circuito Entradas/Salidas digitales

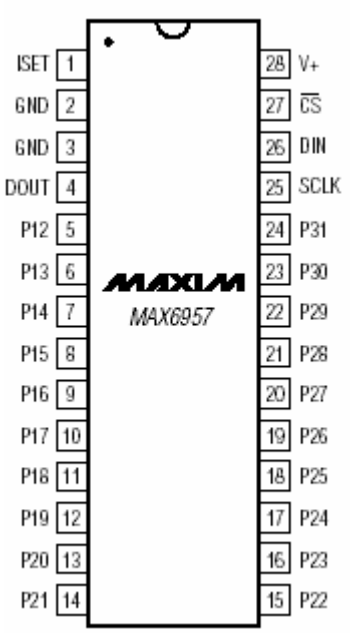
La placa situada en la cabeza del robot ha de gestionar las señales digitales procedentes de la placa de la cabina y asimismo enviar a ésta las que recoje de los sensores. Debido a la gran cantidad de señales involucradas (15 salidas y 6 entradas) se hace necesario el empleo de un expansor de E/S digitales para permitir así la gestión de dichas señales. Este expansor habrá de comunicarse con el PIC mediante alguno de los protocolos serie compatibles como pueden ser I2C o SPI. De esta forma se liberan gran cantidad de pines del PIC que pueden ser usados para otras funciones.

En concreto se ha elegido un expansor de E/S digitales del fabricante Dallas Semiconductor. Se trata del integrado MAX 6957 el cual consta de 20 puertos y se comunica con el PIC mediante protocolo SPI. Veamos algunas características de este expansor:

- Interfaz serie compatible SPI/QSPI/MICROWIRE de alta velocidad (26MHz)
- Rango de operación de 2.5V a 5.5V
- Rango de temperatura de -40°C a +125°C
- 20 puertos configurables como entradas, entradas con pull-up interno,salidas o drivers para LED's
- Corriente de disparo de 11µA (máximo)
- Control de corriente programable individualmente en 16 pasos para cada LED
- Detección lógica de transición en 7 puertos de E/S

Este expansor está disponible en varios encapsulados desde 28 pines hasta 40. En concreto el encapsulado elegido para este proyecto tanto en esta placa como en la situada en la cabina es el DIP de 28 pines. En las características generales del expansor se ha mencionado que los puertos de éste pueden configurarse en cuatro modos. Dado que el MAX-6957 va ser utilizado para ampliar el número de E/S digitales del sistema se obviarán el resto de funcionalidades del dispositivo y sólo se entrará en detalles sobre el empleo del expansor como tal. En la siguiente figura podemos ver la

distribución de los pines en el encapsulado así como una breve explicación de los mismos en forma de tabla:



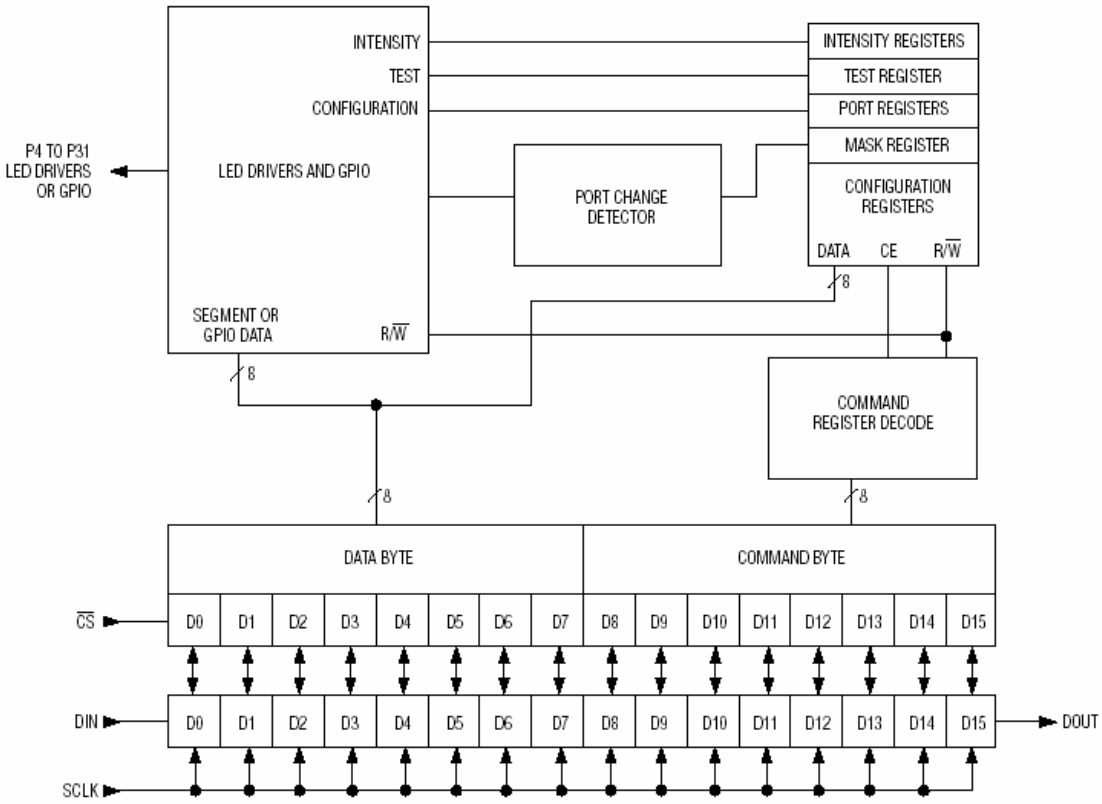
PIN		NAME	FUNCTION
SSOP DIP	SSOP		
1	1	ISET	Segment Current Setting. Connect ISET to GND through a resistor (R_{ISET}) to set the maximum segment current.
2, 3	2, 3	GND	Ground
4	4	DOUT	4-Wire Serial Data Output Port
5–24	—	P12–P31	LED Segment Drivers and GPIO. P12 to P31 can be configured as CA LED drivers, GPIO outputs, CMOS logic inputs, or CMOS logic inputs with weak pullup resistor.
—	5–32	P4–P31	LED Segment Drivers and GPIO. P4 to P31 can be configured as CA LED drivers, GPIO outputs, CMOS logic inputs, or CMOS logic inputs with weak pullup resistor.
25	33	SCLK	4-Wire Serial Clock Input Port
26	34	DIN	4-Wire Serial Data Input Port
27	35	$\overline{CS}$	4-Wire Chip-Select Input, Active Low
28	36	V+	Positive Supply Voltage. Bypass V+ to GND with a minimum 0.047µF capacitor.

Nosotros nos comunicaremos con el expansor a través del protocolo SPI. Los pines involucrados en dicha comunicación serán los que se detallan a continuación:

- SCLK: es la línea de datos de reloj. Un dispositivo Master, en este caso el microcontrolador, genera una señal de reloj en esta línea. Los datos se enviarán o recibirán, según el modo SPI seleccionado de los 4 posibles, en cada pulso de subida o bajada de reloj.

- DIN/SDI: esta línea es la de entrada de datos
- DOUT/SDO: línea de salida de datos
- CS: Es la línea mediante la cual el Master selecciona al Slave con el que se quiere comunicar

Como ya se ha dicho el dispositivo es compatible con el protocolo serie SPI por lo que será el empleado por nosotros para la configuración, lectura y escritura. En el datasheet del integrado se explican los pasos a seguir en los procesos tanto de lectura como de escritura del mismo. Se detallan asimismo las palabras a escribir para el control del expansor, mediante tablas. Antes de pasar a ver cómo son estos procesos de lectura/escritura veremos un diagrama funcional del dispositivo. Es el siguiente:



Pasamos ya a comentar la interfaz serie del dispositivo. El MAX-6957 se comunica a través de una interfaz SPI de 4 líneas. La interfaz tiene tres entradas y una salida. Las entradas son la señal de reloj (SCLK), la entrada de datos (DIN) y el selector de dispositivo (CS) activo a nivel bajo. La salida es la correspondiente a los datos (DOUT). La señal CS como se ha indicado ha de

estar a nivel bajo para colocar datos tanto a la entrada como a la salida del dispositivo. Además DIN ha de estar estable cuando se de el flanco de subida de la señal de reloj para ser muestreado correctamente.

Las señales DIN y SCLK deben ser usadas para transmitir datos a otros periféricos por lo que el MAX-6957 ignora cualquier actividad en SCLK y DIN salvo en los instantes entre una bajada y una subida de la señal CS.

El control del dispositivo requiere el envio de unapalabra de 16 bits. El primer byte, desde D15 hasta D8; es la orden de dirección y el segundo byte, D7 hasta D0; es el byte de datos.

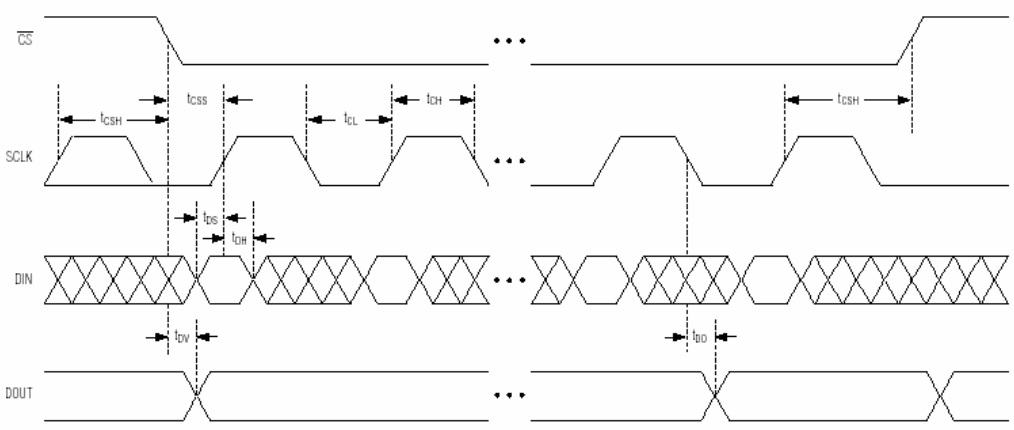
Pueden conectarse varios dispositivos en cadena conectando la salida DOUT de un dispositivo a la entrada DIN del siguiente y monitorizando SCLK y CS en paralelo. Los datos que aparecen en DIN se propagan a través de un registro de desplazamiento interno y aparecen en DOUT 15.5 ciclos de reloj después; en el flanco de bajada de SCLK. Cuando se envian comandos a múltiples MAX-6957's, todos los dispositivos son accedidos al mismo tiempo. Un acceso requiere (16 x n) ciclos de reloj donde n es el número de expansores conectados. Para actualizar sólo un dispositivo de la cadena puede enviarse el comando de No-operación a los demás. Pasamos a ver ahora los procesos de lectura y escritura del dispositivo.

El MAX-6957 contiene un registro de desplazamiento de 16 bits en el cual son desplazados los datos presentes en la entrada DIN con cada flanco de subida de SCLK y cuando CS está a nivel bajo. Cuando CS está a nivel alto las transiciones en SCLK no tienen efecto. Cuando CS está a nivel bajo y conmuta a nivel alto; los 16 bits presentes en el registro de desplazamiento son cargados en un match de 16 bits. Estos 16 bits presentes en el match son decodificados y ejecutados. El MAX 69-57 se escribe usando la siguiente secuencia:

- Tomar SCLK a nivel bajo

- Llevar CS a nivel bajo. Esto habilita el registro de desplazamiento de 16 bits
- Introducir 16 bits en la entrada DIN (primero D15 y por último D0) teniendo en cuenta los tiempos de establecimiento y espera. El primer bit (D15) estará a nivel bajo indicando que se trata de un comando de escritura.
- Llevar CS a nivel alto
- Tomar SCLK a nivel bajo

La siguiente figura muestra una operación de escritura de 16 bits:



Se pueden introducir más de 16 bits entre un flanco de bajada y uno de subida de la señal CS pero sólo se tendrán en cuenta los últimos 16 siendo desechados el resto.

Cualquier registro interno de datos del MAX-6957 puede ser leído enviando la correspondiente palabra de 16 bits y colocando a nivel alto el bit D15 para indicar comando de lectura. La secuencia es la siguiente:

- Tomar SCLK a nivel bajo
- Llevar CS a nivel bajo. Esto habilita el registro de desplazamiento de 16 bits
- Introducir 16 bits en la entrada DIN (primero D15 y por último D0) teniendo en cuenta los tiempos de establecimiento y espera. El primer bit (D15) estará a nivel alto indicando que se trata de un comando de lectura. Los bits D14 a D8 contienen la dirección del registro a ser leído. Finalmente los bits D7 a D0 contienen un dato falso que ser

- Llevar CS a nivel alto cuando SCLK está aún a nivel alto después de muestrear el último bit de datos. Las posiciones D7 a D0 del registro de desplazamiento están ahora cargadas con el registro de datos diseccionado por los bits D1 a D8.
- Tomar SCLK a nivel bajo
- Volver a realizar otra orden de lectura o escritura (puede ser una orden de No-operación) y examinar la cadena de bits presentes en DOUT. El segundo byte contiene los datos del registro solicitado en los pasos anteriores.

Al iniciarse la operación del dispositivo todos los registros de control son peseteados y cargados con su mínimo valor haciendo que el expansor entre en modo de baja corriente. Para hacer que el dispositivo entre en el modo de funcionamiento normal o devolverlo de éste al modo de baja corriente habría que escribirlo siguiendo la siguiente tabla:

FUNCTION	ADDRESS CODE (HEX)	REGISTER DATA							
		D7	D6	D5	D4	D3	D2	D1	D0
Shutdown	0x04	M	I	X	X	X	X	X	0
Normal Operation	0x04	M	I	X	X	X	X	X	1

Además del expansor también están presentes en el circuito tanto de entradas como de salidas digitales los optoacopladores encargados de adaptar los niveles de señal. Dichos optoacopladores así como la circuitería externa asociada a los mismos se comentarán en cada caso particular. Pasamos entonces al estudio de los circuitos de entradas y salidas digitales.

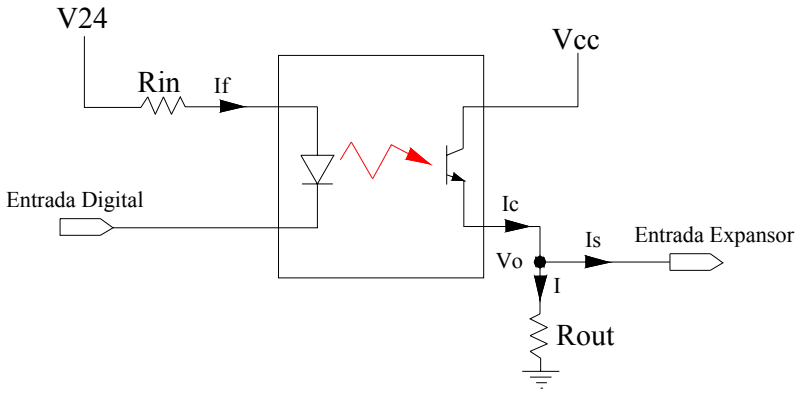
2.1.1.4.1 Entradas digitales

Existen seis entradas digitales, dos de las cuales van a una velocidad de 500Hz. De estas seis entradas digitales sólo estas dos de 500Hz son las que actualmente tienen definida una funcionalidad; siendo el resto de ellas señales de previsión para futuras mejoras del robot forestal. Una de ellas lleva la información relativa al detector de sierra y la otra la relativa al detector de presencia de árbol. En concreto la primera de ellas es de vital importancia pues en el modo manual de funcionamiento del robot su estado constituye uno de los parámetros de seguridad del sistema; de manera que un fallo en esta señal ocasiona que el robot deje de funcionar. Todas las señales varían sus valores

entre los niveles 24V-0V, correspondiendo a un 1 o un 0 respectivamente. Debido a que los niveles de entrada del expansor MAX6957 se mueven entre valores standard 5V-0V, tenemos que acondicionar dichas señales de entrada para que sean compatibles con los niveles lógicos de entrada del integrado.

Igualmente para conseguir el aislamiento total de la circuitería interna, se colocan optoacopladores, PS2501-4, que además nos servirán para escalar las señales de entrada al valor correspondiente.

El circuito de escalado y optoaislamiento de cada una de las entradas digitales es el siguiente:



El PS2501-4 es un optoacoplador que consiste en un diodo emisor de luz infrarroja y un fototransistor de unión de silicio NPN. Cuando If circula por el diodo emite una luz infrarroja que activa la tensión de base del fototransistor y deja circular una Ic proporcional a If.

La relación existente entre Ic e If, es lo que se llama Ratio de transferencia de corriente, CTR, y se suele expresar en porcentaje. El CTR depende de la temperatura ambiental, y del tiempo de funcionamiento del dispositivo. Varia entre un 80%-600%, siendo un valor aconsejable para los primeros cálculos del 100%.

$$Ic = CTR\% \cdot If$$

La caída de tensión en el diodo cuando esta conduciendo If = 3mA,

suele ser de 1.15V, de esta forma se tiene un valor aproximado de la resistencia de entrada Rin:

$$\text{Rin} = \frac{24\text{V} - 1.1\text{V}}{2\text{mA}} = 11450\ \Omega$$

Escogiendo una Rin = 11K, nos da una If e Ic:

$$\text{If} = \frac{24\text{V} - 1.1\text{V}}{11\text{K}} = 2.08\text{mA}$$

Ic = 100% · If = 2.08 mA

Para calcular la resistencia de salida R_{out} :

$$R_{out} = \frac{5\text{V} - 0.8\text{V}}{2.02\text{mA}} = 2019\,\Omega$$

Donde se ha escogido una R_{out}=2K4.

Todos estos cálculos se han basado en el hecho de que el expansor MAX6957 no consume prácticamente corriente en sus entradas digitales. Los niveles lógicos de entrada reconocidos por este integrado son:

$-0.5\text{V} \le V_L \le 1.5\text{V}$

$3.5\text{V} \le V_H \le 5.5\text{V}$

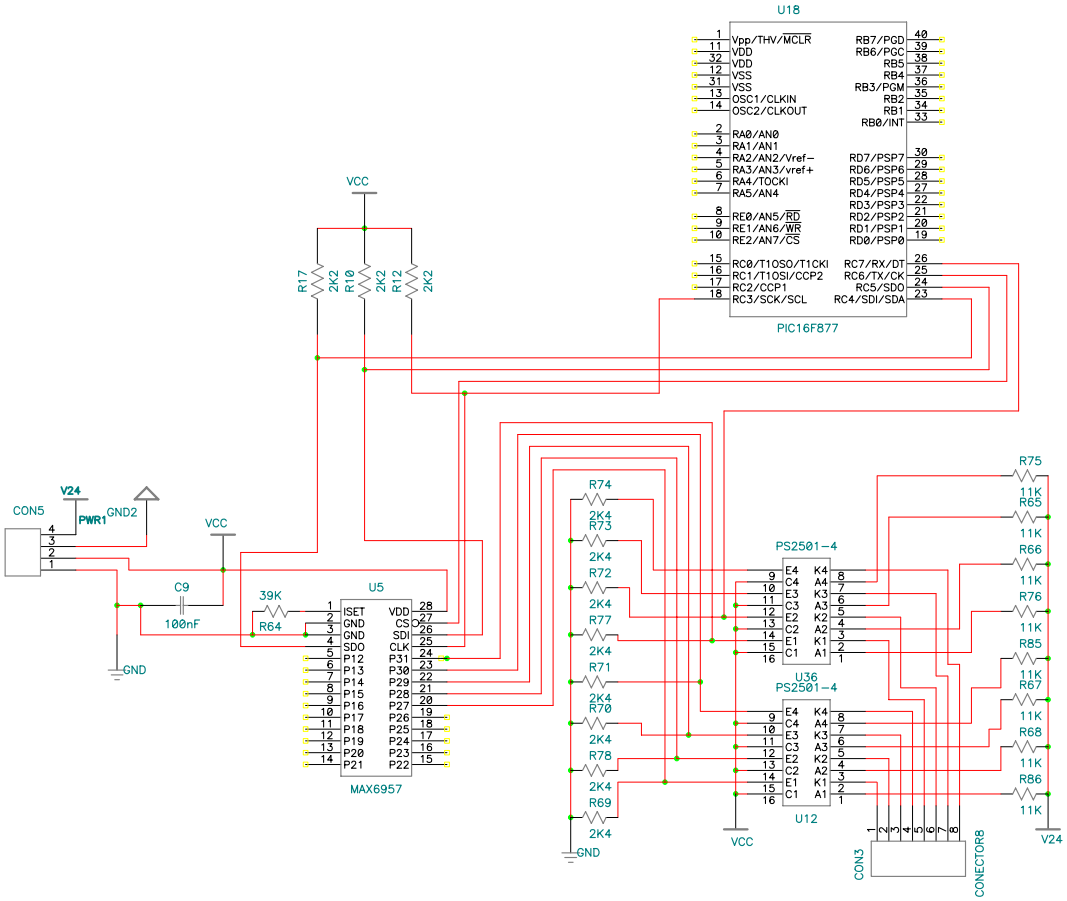
Y teniendo en cuenta que por el fototransistor circula una corriente de oscuridad de 100nA cuando el diodo no emite luz, hallamos los niveles de tensión que se presentan en las entradas del expansor:

$V_L = 5 - 2\text{K} \cdot 2.02\text{mA} = 0.960\text{V}$

$V_H = 5 - 2\text{K} \cdot 100\text{nA} = 4.99\text{V}$

De esta forma comprobamos que ambos valores se encuentran en los rangos lógicos de tensión del expansor.

Para terminar el esquemático del circuito correspondiente a las entradas digitales es el siguiente:



2.1.1.4.2 Salidas digitales

Las 15 señales digitales de salidas, actúan del mismo modo que las de entrada digitales. Todas varían sus valores entre los niveles 24V-0V, correspondiendo a un 1 o un 0 respectivamente. En cuanto a su funcionalidad todas controlan electroválvulas encargadas de abrir y cerrar rodillos, abrir y cerrar cuchillas, adelantar y retrasar motores, subir y bajar la sierra, etc. No obstante algunas están como previsión para futuras ampliaciones al igual que ocurría con las entradas.

Debido a que los niveles digitales de salida del expansor MAX6957 se

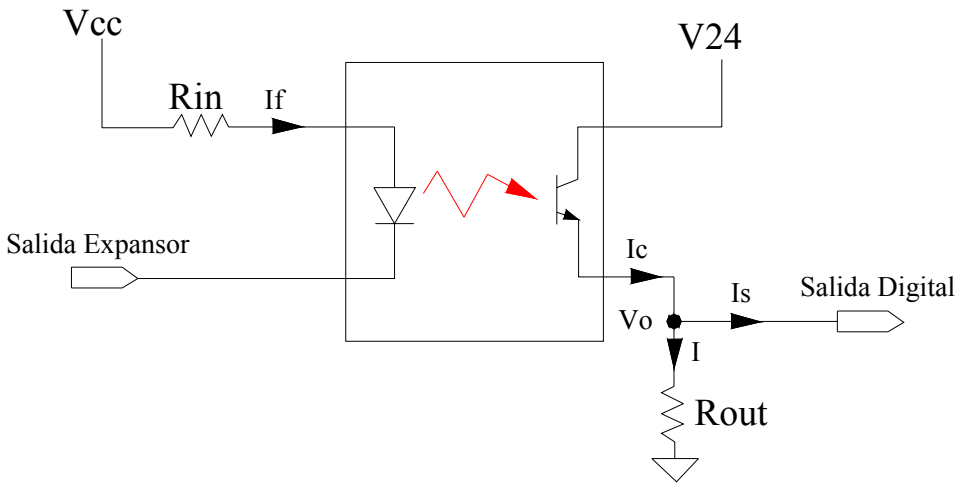
mueven entre valores standard 5V-0V, tenemos que acondicionar las tensiones de salida del expansor a las tensiones de salida de la placa.

Igualmente para conseguir el aislamiento total de la circuitería interna, se colocan optoacopladores, PS2501-4, que además nos servirán para escalar las señales de salida al valor correspondiente.

De este modo, el valor de cada señal de salida es convertida desde los niveles lógicos 5V-0V, a 24V-0V.

Estas señales digitales atacan a un buffer que proporciona a su salida los 400mA necesarios para el funcionamiento del robot. Este buffer consume 3mA como máximo, cuando la tensión de salida esta a nivel alto, es decir, alrededor de unos 20V. El buffer es el encargado de suministrar la tensión y la corriente necesaria a las salidas, 24V y 400mA.

El circuito de escalado y optoaislamiento de cada una de las salidas digitales es el siguiente:



El PS2501-4 funciona de la misma forma que se explicó en el apartado de entradas digitales, así como el CTR.

Si suponemos una I_F = 3mA, para cada una de las salidas, la caída de

tensión en el diodo suele ser de 1.15V, como se puede observar en las hojas de características del optoacoplador. De esta forma se tiene un valor aproximado de la resistencia de entrada Rin:

$$\text{Rin} = \frac{5\text{V} - 1.15\text{V}}{3\text{mA}} = 1283\Omega$$

Escogiendo una Rin = 1K2 Ω, nos da una If y Ic:

$$\text{If} = \frac{5\text{V} - 1.15\text{V}}{750} = 3\text{mA}$$

Ic = 100% · If = 3 mA

Para calcular la resistencia de salida Rout:

$$\text{Rout} = \frac{V_o}{i} = \frac{V_o}{I_c - I_s} = \frac{20\text{V}}{3\text{mA} - 1\text{mA}} = 10\text{K}\ \Omega$$

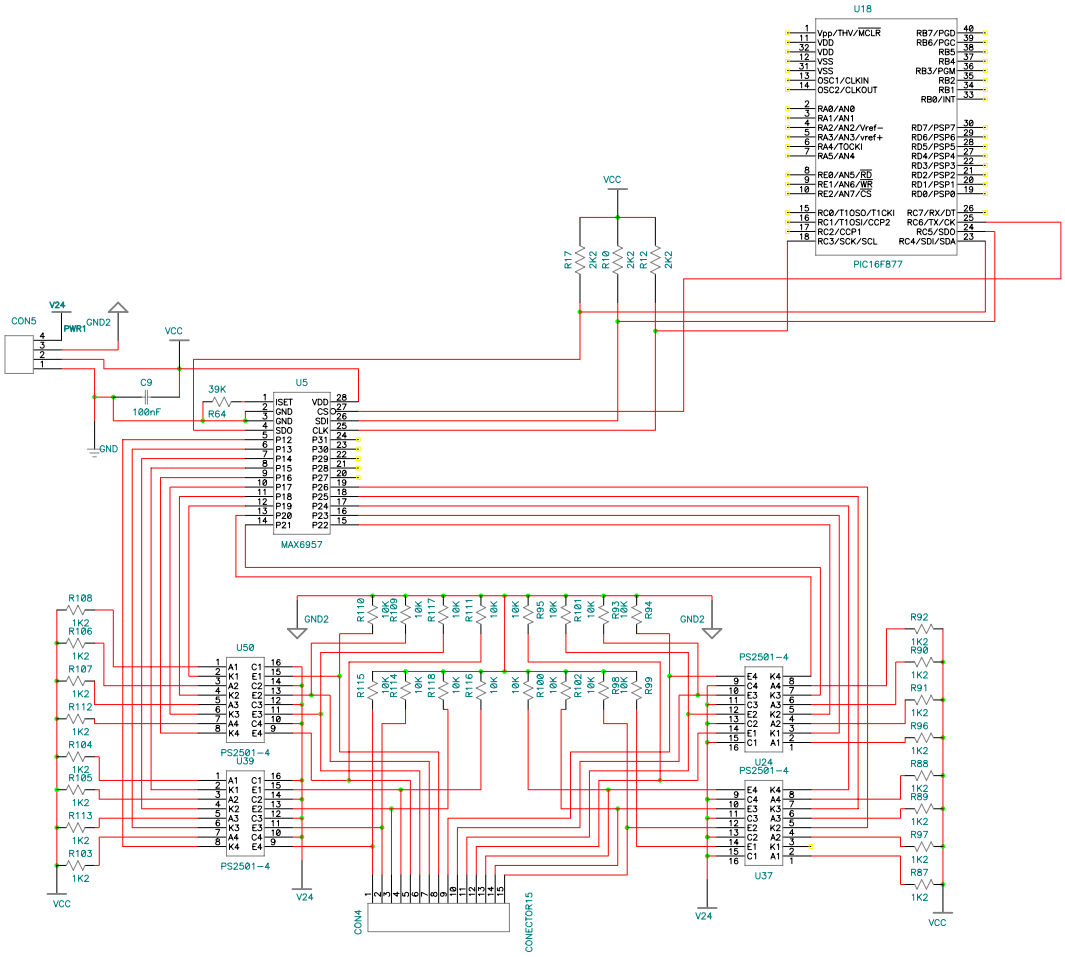
Donde se ha escogido una Rout = 10K Ω.

Los niveles lógicos a la entrada del buffer teniendo en cuenta que por el fototransistor circula una corriente de oscuridad de 100nA cuando el diodo no emite luz son:

V_L = 5K · 100nA = 500 µV
V_H = 5K (5.13 mA – 3mA) = 10.65 V

De esta forma comprobamos que ambos valores se encuentran en los rangos lógicos de tensión de entrada al buffer.

Nuevamente para finalizar con las salidas digitales mostraremos el esquemático del circuito correspondiente a dichas salidas:



2.1.1.5 Circuito entradas encoder

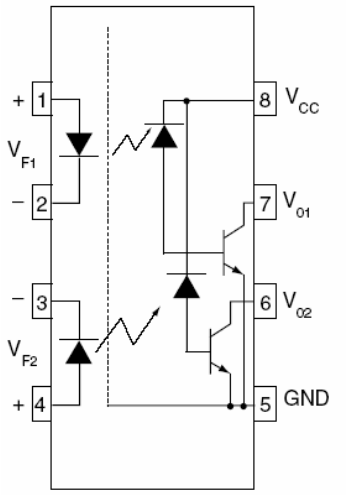
Existen dos entradas digitales procedentes de un encoder en cuadratura, que al igual que las señales digitales de entrada y de salida que ya se vieron anteriormente necesitan ser escaladas. Estas señales sólo tienen sentido para el modo de funcionamiento automático; siendo ignoradas en el modo manual. Se mueven entre los niveles digitales 24V-0V, correspondiendo a un 1 o un 0 respectivamente. Debido a que los niveles digitales de entrada de las puertas del PIC se mueven entre valores standard 5V-0V, tenemos que acondicionar dichas señales de entrada como ya se dijo antes.

Asimismo se ha de seguir manteniendo el aislamiento total de la circuitería interna, colocando optoacopladores, HCPL-2531, que además nos servirán para escalar las señales de entrada al valor correspondiente.

Las señales de encoder son de 2 Khz de frecuencia. Esta alta frecuencia

hace que el optoacoplador que se utilice tenga que ser lo suficientemente rápido para transmitir la señal. El HCPL-2531 de FAIRCHILD es un integrado que dispone de dos optoacopladores que pueden transmitir una señal de hasta 1 Mb/s con lo que cubrimos sobradamente la frecuencia de las señales procedentes del encoder.

Mediante la combinación adecuada de resistencias conseguimos que ante una entrada de pulsos de 5 V a 2 Khz tengamos una salida de pulsos de 24 V a 2 Khz. No están preparados para suministrar intensidad de salida a elementos de potencia, por lo cual es necesario el uso de transistores de efecto de campo MOS para alimentar a estos elementos. Veamos un esquema del integrado:



El diodo LED de entrada al circular intensidad por el radiará luz que incidirá en un fotodiodo haciendo que circule intensidad por la base del transistor de salida, de forma que este entre en saturación. En caso de no circular intensidad por él, el transistor permanecerá en corte.

El calculo de las resistencias es igual que en el apartado de entradas digitales. Los valores obtenidos son de 10K Ω para Rin y de 22K Ω para Rout.

El único aspecto a tener en cuenta es que las señales de encoder una vez escaladas y optoaisladas se conectan a las entradas digitales RB3 y RB4

del PIC. Dichas entradas son buffereadas por un Schmitt Trigger, cuyos niveles de tensión lógicos son:

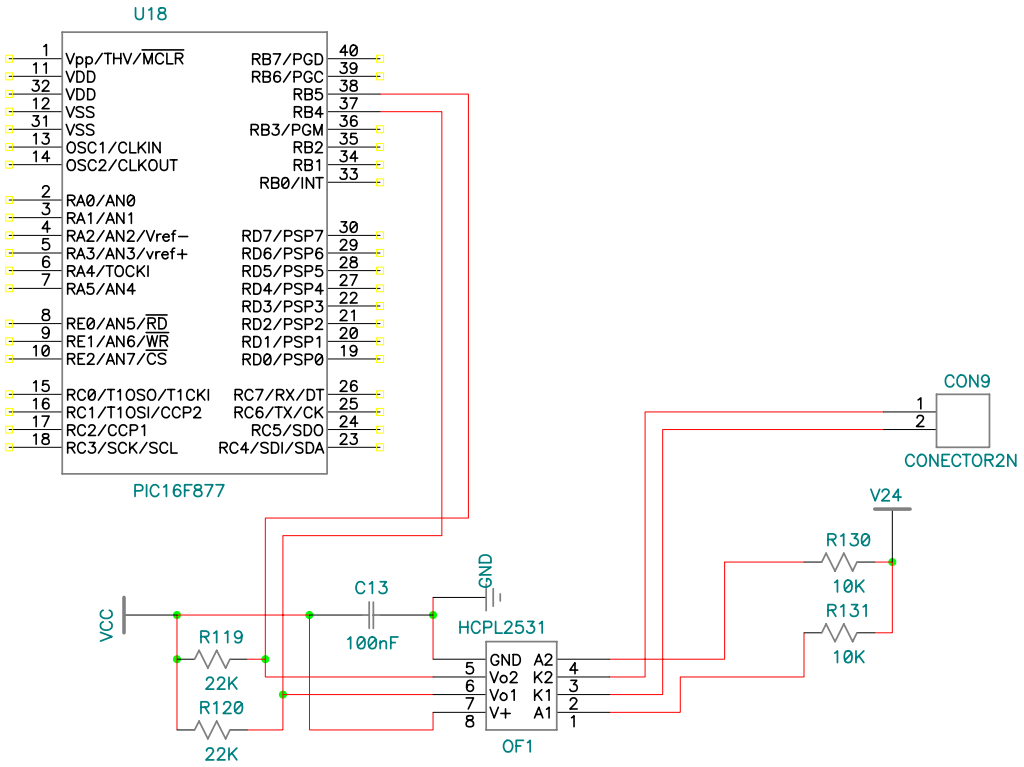
$$0V \leq V_L \leq 1V$$

$$4V \leq V_H \leq 5V$$

y comprobamos que son totalmente compatibles con las tensiones que nos da el circuito de escalado y que fueron calculados en el apartado anterior.

Para terminar indicar como característica importante de estas dos señales que su frecuencia es de 2KHz y el desfase entre ambas de 90°. Esto será importante a la hora de realizar el programa que gestione dichas entradas y que se verá con detenimiento en la parte software del proyecto.

Finalmente el esquemático correspondiente a estas entradas es el siguiente:

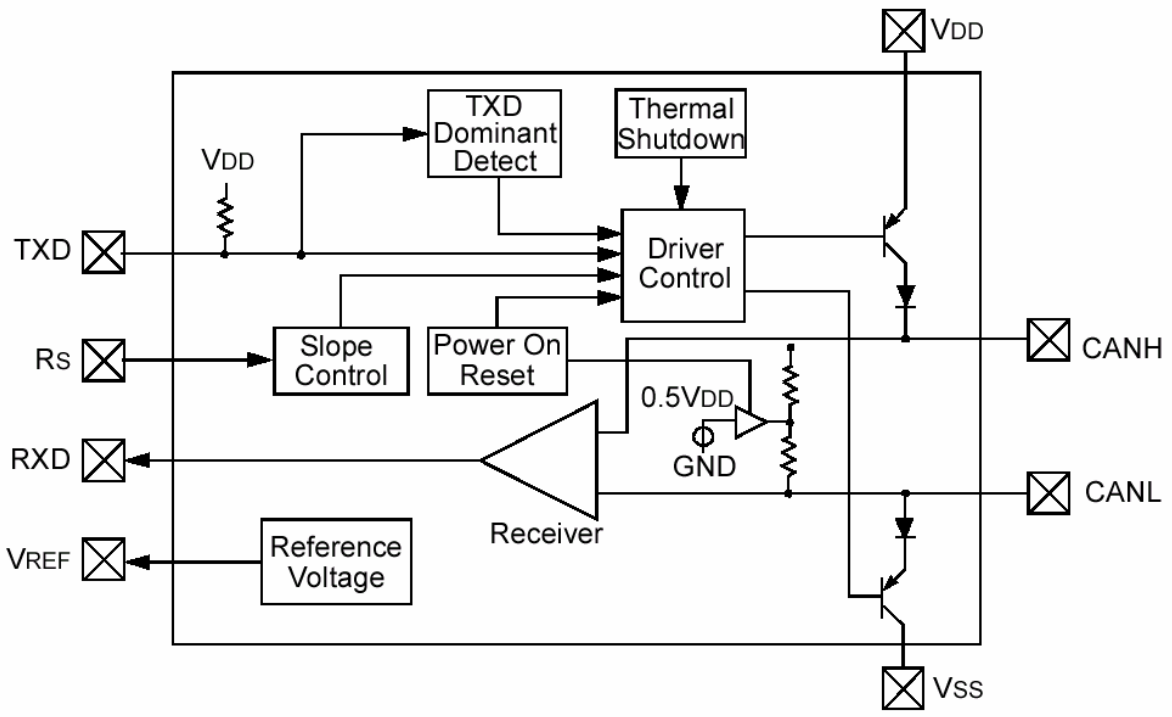


2.1.1.6 Circuito CAN

Las líneas de transmisión y recepción del CAN deben pasar por el MCP2551 que actúa de interface entre el controlador del protocolo CAN y la red física del bus. Este dispositivo es capaz de diferenciar entre las transmisiones y recepciones con el controlador CAN y es completamente compatible con el standard ISO 11898, incluyendo el requerimiento de 24V. Es capaz de operar hasta velocidades de 1Mb/s.

Normalmente, cada nodo en un sistema CAN debe tener un dispositivo que convierta las señales generadas por el controlador CAN a señales adecuadas a la transmisión sobre los cables del bus (salida diferencial). Además provee un buffer entre el controlador CAN y los enclavamientos de alta tensión que se producen en el BUS debido a fuentes externas tales como EMI, ESD, transitorios eléctricos, etc...

El diagrama de bloques del funcionamiento del MCP2551 es el que sigue:



Como se explicó en el protocolo CAN, éste se basa en dos estados, dominante y recesivo. Un estado dominante ocurre cuando la diferencia de tensión entre CANH y CANL es más grande que una tensión definida. Un estado recesivo ocurre cuando la diferencia de tensión es menor que una tensión definida. Los estados dominante y recesivo corresponden a los estados bajo y alto de la entrada TXD. Sin embargo, cuando un estado dominante es iniciado por otro nodo CAN, un estado recesivo se sobrescribe en el CAN bus.

El MCP2551 conduce un mínimo de carga de 45 Ω, permitiendo un máximo de 112 nodos conectados a la misma línea de CAN bus. Estas líneas de bus deben tener una resistencia de entrada diferencial mínima de 20KΩ y una resistencia nominal en la terminación de 120 Ω. Esta última se ha incluido en nuestro diseño para el correcto funcionamiento del bus.

La salida RXD refleja la diferencia de tensión entre CANH y CANL. Los estados bajos y alto de RXD corresponden a los estados dominantes y recesivos respectivamente.

Los pines CANH y CANL están protegidos contra cortocircuitos de baterías y transitorios eléctricos que puedan ocurrir sobre la línea CAN bus. Esta característica previene la destrucción de la salida de transmisión.

El dispositivo también esta protegido contra un exceso de corriente ya que posee una circuitería de apagado térmico que deshabilita las salidas cuando la temperatura de unión excede de los 165ºC. El resto de las partes del integrado permanecen operativas y la temperatura del chip decrece debido a una bajada de la disipación de potencia en la salida de transmisión del dispositivo. Esta protección es esencial contra cortocircuitos en la línea del bus que pueda producir daños.

El MCP2551 puede operar en tres modos diferentes:

- Alta velocidad
- Control de subida-bajada

- Standby

Cuando se encuentra funcionando en Alta velocidad o control de subida-bajada, los drivers de las señales de CANH y CANL se regulan internamente y proporcionan una simetría controlada para minimizar la emisión EMI.

La alta velocidad se selecciona conectando el pin Rs a Vss, es decir a tierra, como se puede apreciar en el esquema de nuestra placa. De esta forma los drivers de salidas al bus soportan las más altas velocidades permitidas.

Por otro lado, en el modo de control de las subidas y bajadas, las transiciones de las señales pueden ser controladas por una resistencia (conectando el pin Rs a tierra) con una inclinación proporcional a la corriente de salida por Rs, que reduce bastante las emisiones EMI.

El modo standbye se produce cuando se aplica un nivel alto al pin Rs. De esta forma el transmisor se apaga y el receptor opera a baja corriente. El pin RXD aún continua funcionando pero a velocidades menores. El microcontrolador de la placa puede ver el estado de RXD para ver el tipo de actividad del CAN y volver a colocar el dispositivo en modo normal de operación mediante el pin Rs.

A continuación se ven las características de los modos de operación y la tabla de verdad de la línea CAN bus:

TABLE 1-1: MODES OF OPERATION

Mode	Current at R_s Pin	Resulting Voltage at Rs Pin
Standby	$-I_{RS} < 10\ \mu A$	$V_{RS} > 0.75V_{DD}$
Slope Control	$10\ \mu A < -I_{RS} < 200\ \mu A$	$0.4V_{DD} < V_{RS} < 0.6V_{DD}$
High Speed	$-I_{RS} < 610\ \mu A$	$0 < V_{RS} < 0.3V_{DD}$

TABLE 1-2: TRANSCEIVER TRUTH TABLE

V_{DD}	V_{RS}	TXD	CANH	CANL	Bus State(1)	RXD(1)
$4.5\ V \le V_{DD} \le 5.5\ V$	$V_{RS} < 0.75V_{DD}$	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	$V_{RS} > 0.75V_{DD}$	X	Not Driven	Not Driven	Recessive	1
$V_{POR} < V_{DD} < 4.5\ V$ (See **Note 3**)	$V_{RS} < 0.75V_{DD}$	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	$V_{RS} > 0.75V_{DD}$	X	Not Driven	Not Driven	Recessive	1
$0 < V_{DD} < V_{POR}$	X	X	Not Driven/ No Load	Not Driven/ No Load	High Impedance	X

Si el MCP2551 detecta un estado bajo en la entrada TXD durante un periodo de tiempo grande, deshabilitara los drivers de salidas CANH y CANL para prevenir la corrupción de datos del CAN bus. Los drivers son deshabilitados si el estado bajo del TXD dura mas de 1.25ms. Esto implica que el tiempo de transmisión de un bit será como máximo de 62.5us o de otra forma, que la velocidad mínima de transmisión de sea de 16 kb/s (esto permite como máximo 20 transmisiones consecutivas de bits dominantes). Los drivers permanecen deshabilitados tanto tiempo como TXD permanece a nivel bajo. Una subida de tensión en TXD resetea el contador y habilita la salida de los drivers CANH y CANL.

Cuando se alimenta el dispositivo, CANH y CANL permanecen en estado de alta impedancia hasta que Vdd busca la tensión de alimentación. Por otro lado también permanecen en alta impedancia cuando TXD esta bajo y Vdd ha llegado al nivel de alimentación.

A continuación se detalla el funcionamiento de cada uno de los pines del integrado MCP2551:

- Pin TXD. Es la entrada de transmisión de datos. Es una entrada compatible TTL. Los datos de este pin son conducidos a los pines de salida diferencial CANH y CANL. Se conecta a la salida de transmisión del PIC. Tiene una resistencia interna de pull-up de 25 KΩ.
- Pin Vss. Es el pin de suministro de la tierra lógica del circuito
- Pin Vdd. Es el pin de la referencia de tensión positiva
- Pin RXD. Es la salida de los datos recibidos, también es compatible TTL.
- Pin Vref. Es una salida de tensión definida como Vdd/2. Normalmente saca unos 2.5V cuando se alimenta el integrado con 5V.
- Pin CANL. Es la salida que conduce el lado de tensión baja en el bus diferencial. Este pin esta unido internamente a la entrada del comparador de recepción.
- Pin CANH. Conduce el lado de alta tensión en el bus diferencial. Igualmente unido a la entrada del comparador de recepción.

- Pin Rs. Es la entrada para seleccionar el modo de operación del integrado.

Entre las características eléctricas de este dispositivo cabe destacar las siguientes definiciones, de acuerdo a la norma ISO 11898:

- Tensión del Bus. Es la tensión que tiene cada una de las líneas del bus, es decir, CANH y CANL con respecto a tierra. Se denota por VCANH y VCANL y sus valores dependen del estado en que se encuentre el bus:

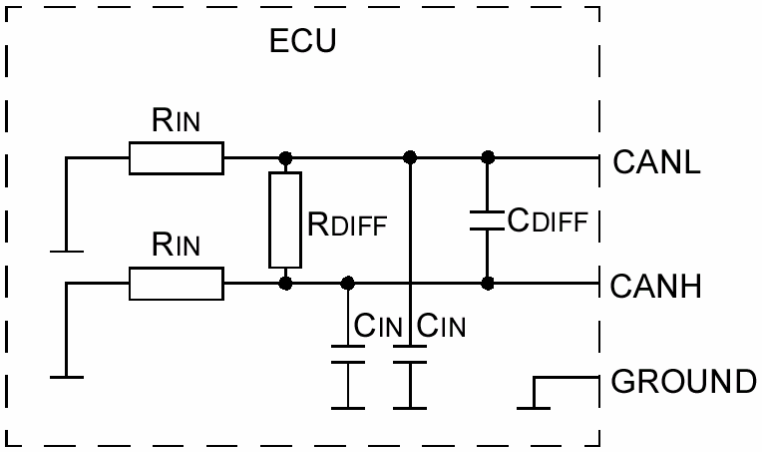
ESTADO	V_{CANH}	V_{CANL}
RECESIVO	2V – 3V	2V – 3V
DOMINANTE	2.75V – 4.5V	0.5V – 2.25V

- Rango de tensiones en modo común, o límite de niveles de tensiones V_{CANH} y V_{CANL} para los cuales pueden ocurrir operaciones apropiadas si se conecta el máximo numero de nodos al CAN.
- Capacidad diferencial, C_{DIFF} , es la capacidad vista entre CANH y CANL durante el estado recesivo cuando el nodo es desconectado del bus. Esta capacidad suele ser de unos 10pF
- Resistencia diferencial, R_{DIFF} , es la resistencia vista entre las dos líneas CAN durante el estado recesivo cuando el nodo es desconectado del bus. Su valor oscila entre 20 KΩ – 100 KΩ.
- Tensión diferencial, V_{DIFF} , es la diferencia de tensiones entre V_{CANH} y V_{CANL} . Sus valores dependen del estado en que se encuentre el bus:

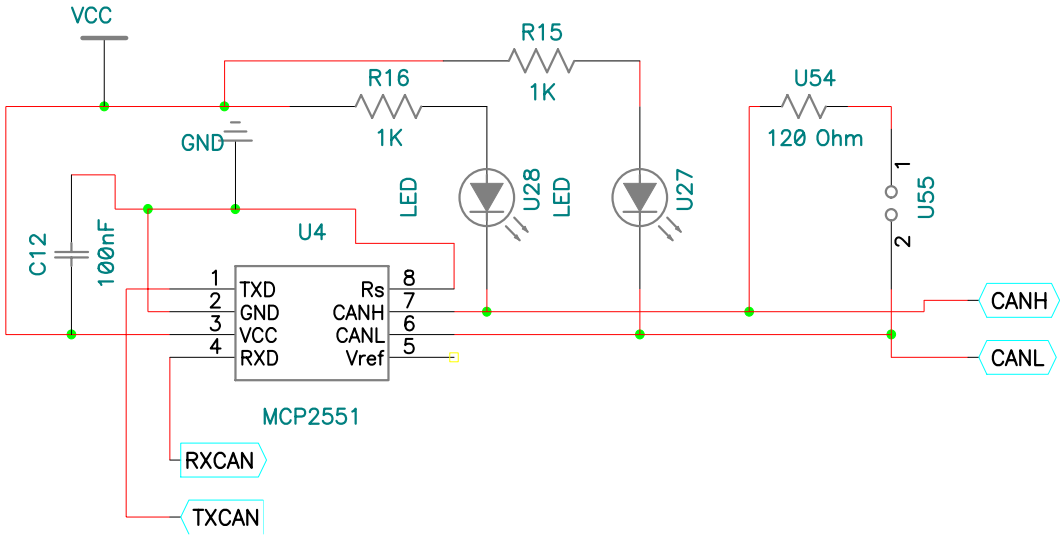
ESTADO	V_{DIFF}
RECESIVO	-500mV - 50mV
DOMINANTE	1.5V – 3V

- La capacidad interna, C_{IN} , es la capacidad vista entre cada una de las líneas del bus y tierra durante el estado recesivo cuando el nodo CAN se desconecta del bus. Su valor suele ser de 20pF.
- La resistencia interna, R_{IN} , es la resistencia vista entre cada una de las líneas del bus y tierra durante el estado recesivo cuando el nodo CAN se desconecta del bus. Su valor oscila entre 5 KΩ – 50 KΩ.

En este esquema se puede ver todos los parámetros descritos:



Finalmente y como en apartados anteriores se muestra el esquemático correspondiente a las salidas procedentes del módulo de CAN Bus del PIC:



2.1.1.7 Circuito salidas PWM

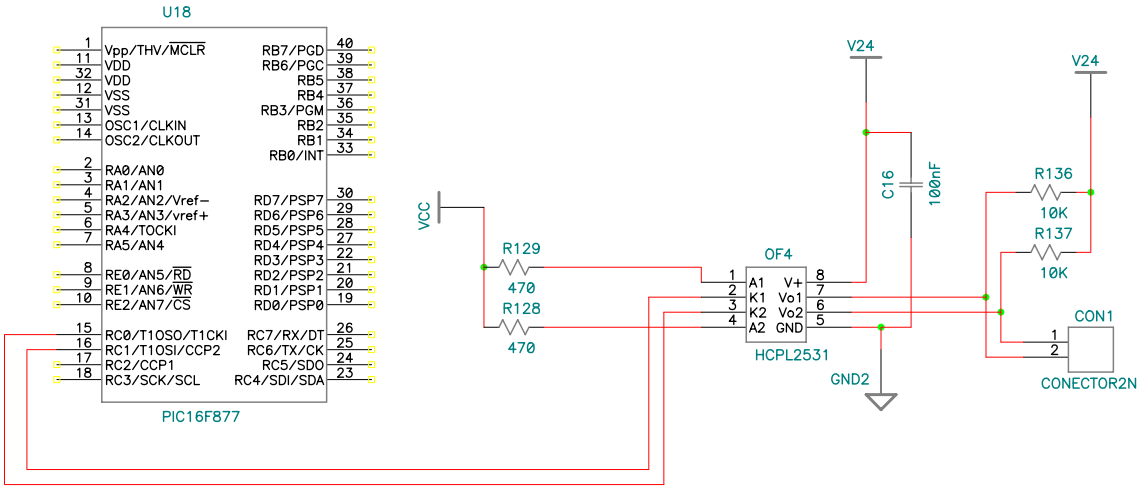
En cuanto a las salidas PWM decir que actúan sobre electroválvulas proporcionales que controlan el caudal en rodillos y cuchillas. Al igual que en casos anteriores hay que acondicionar dichas señales a los valores de tensión de nuestro sistema así como realizar el aislamiento de la placa con el exterior.

Emplearemos al igual que hacíamos con las entradas procedentes del encoder, el integrado HCPL2531 del cual hablamos con detalle en dicho

apartado.

Nuevamente habrá que rehacer los cálculos a fin de determinar los valores de resistencias necesarios para el fucionamiento deseado del circuito. En nuestro caso estos cálculos no se repetirán de nuevo y nos limitaremos a indicar los valores obtenidos. Así pues tenemos que emplear resistencias de 470Ω para Rin y de 10K Ω para Rout.

El esquemático completo de este circuito es el que puede verse a continuación:

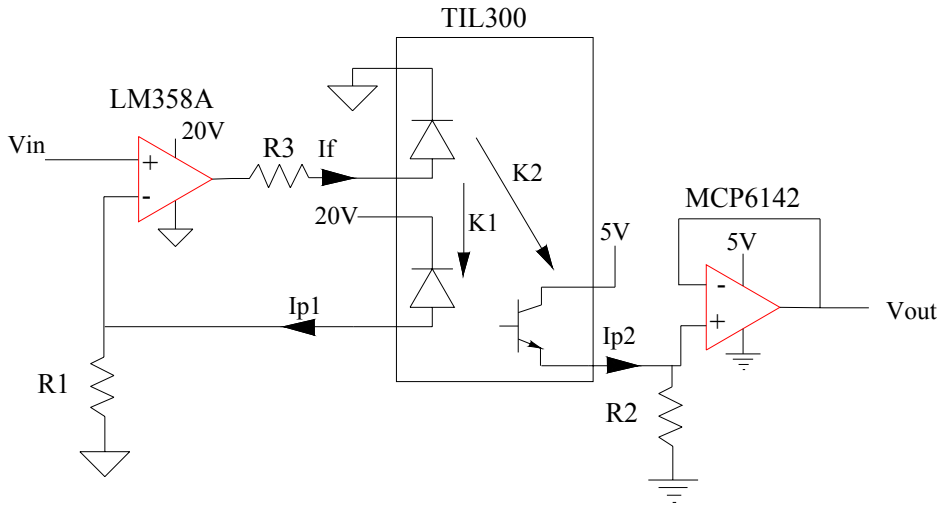


2.1.1.8 Circuito entradas ADC

Las entradas analógicas varían entre 10V y 0V teniendo una sensibilidad de mV y siendo enviadas al PLC con una conversión de al menos 8 bits. Sólo se emplean 2 de ellas como potenciómetros de diámetro (medida de diámetro). Podría trabajar la máquina con sólo uno de ellos pero resulta impreciso. Debido a que los niveles de tensión de entrada de los canales analógicos del ADC del PIC se mueven entre los valores standard 5V-0V, tenemos que acondicionar dichas señales para que sean compatibles con los niveles lógicos del integrado. Los amplificadores usados pueden ser el NE5532 y MCP6002, pero estos poseen un ancho de banda mayor y dan lugar a salidas más oscilatorias. Por esto se usan los MCP6142 y LM358.

Igualmente para conseguir el aislamiento total de la circuitería interna, se colocan optoacopladores lineales, IL300, que además nos servirán para escalar las señales de entrada al valor correspondiente.

El circuito de escalado y optoaislamiento de cada una de las entradas analógicas es el siguiente:



El IL300 es un optoacoplador lineal que consiste en un diodo LED emisor de luz infrarroja y dos fotodiodos capturadores de la radiación. El fotodiodo del bucle de control feedback, captura un porcentaje del flujo emitido por el LED y genera una señal de control, I_{P1}, que puede ser usada como servo para regular la corriente que circular por el diodo LED. Esta técnica compensa la no linearidad del LED, y las características de tiempo y temperatura. El rango de funcionamiento más lineal es cuando el diodo LED conduce una corriente entre 5mA y 20mA. A su vez, el fotodiodo de salida produce una señal de salida, Ip2, que esta relacionada linearmente con el flujo creado por el LED.

La corriente que circula por el fotodiodo de control, Ip1, satisface la siguiente relación:

$$I_{p1} = \frac{V_{in}}{R_1}$$

La magnitud de esta corriente es directamente proporcional a una cierta

cantidad, K1, de veces la corriente que circula por el diodo LED emisor, es decir:

$$I_{p1} = K1 \cdot I_F$$

El amplificador operacional suministrara corriente al LED de forma que fuerce a mantener la tensión de sus entradas positiva y negativa iguales.

El fotodiodo de salida se conecta a un amplificador seguidor de tensión no inversor. La resistencia de carga, adecua la corriente a la conversión de tensión. La tensión de salida del amplificador es el producto de la corriente Ip2, con la resistencia de carga R2. A su vez la corriente Ip2, es K2 veces la corriente If. De este modo:

$$V_0 = I_{p2} \cdot R_2 = K_2 \cdot I_F \cdot R_2$$

Sin embargo, la ganancia de transferencia total del circuito que relaciona la tensión de entrada V_{in} con la tensión de salida V_o, es completamente independiente de I_F, y la relación entre tensiones viene fijada por la relación entre las resistencias de entrada y salida del circuito, tal y como vemos en la siguiente expresión:

$$\frac{V_o}{V_{in}} = \frac{K_2 \cdot R_2}{K_1 \cdot R_1} = K_3 \frac{R_2}{R_1}$$

Para calcular la magnitud de las resistencias en el circuito, lo primero que se establece es cuanta corriente atravesara el LED, I_F, ante valores intermedios de tensión de entrada V_{in}. Como el rango de entrada puede variar entre 10V y 0V, escogemos 5V como valor medio de la misma. Igualmente establecemos una corriente de I_F = 5mA para dicha tensión de entrada.

Observando en las curvas características de IL300, obtenemos la caída de tensión en el diodo LED para una I_F determinada. En este caso V_D=1.22V aproximadamente. La tensión a la salida del amplificador será de 12V, ya que

estamos operando en rangos intermedios de tensión. Con estos datos podemos determinar el valor de R_3:

$$R_3 = \frac{12V - 1.22V}{5mA} = 2156\Omega$$

Escogemos R_3 = 2K2. Del mismo modo se calcula R_1 a partir del valor de I_F. Observando en las curvas características tenemos un I_{p1} = 40uA aproximadamente, así:

$$R_1 = \frac{5V}{40\,\mu A} = 125K$$

Escogemos una R_1=130K, porque es un valor mas comercial.

La resistencia R_2, se escoge en función de la relación que deseemos establecer entre las tensiones de entrada y salida. En nuestro caso queremos que la salida sea la mitad que el valor de entrada:

$$\frac{V_0}{V_{in}} = K_3 \frac{R_2}{R_1} = 0.5$$

$$R_2 = 0.5 \frac{R_1}{K_3}$$

El valor de K_3 viene establecido en la hoja de características de IL300 como una letra añadida al nombre del integrado. Los optoacopladores usados en esta placa tienen una K_3 de valor aproximadamente la unidad.

Para calcular el valor de R_2, sustituimos el valor de K_3:

$$R_2 = 0.5 \times 125 = 62.5K$$

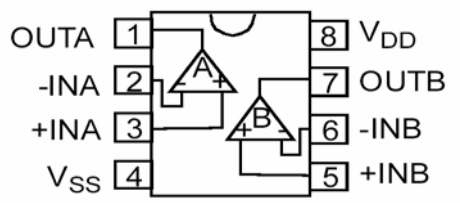
Eligiendo un valor mas comercial, R_2 = 62K.

De esta forma obtenemos los niveles de tensión adecuados para la entrada del conversor Analógico-Digital del propio PIC, y conservando el aislamiento de la circuitería interna.

Pasamos ahora a hablar de los amplificadores operacionales empleados en el montaje. Para los montajes de los optoacopladores lineales se han utilizado dos amplificadores, uno para el lado de 5 V y otro para el lado de 10 V. Ambos amplificadores son duales, dos amplificadores por integrado.

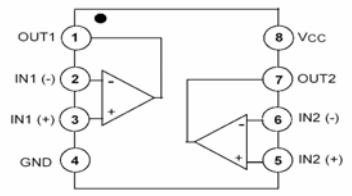
Para la parte de 5V se utiliza el integrado MCP6142 de Microchip. Este es un amplificador Rail to Rail, es decir, puede dar a su salida desde –Vcc a +Vcc (tensiones de alimentación del amplificador). Se ha utilizado para la parte de 5 V debido a que su rango de tensiones va desde 1.4 a 5.5 V y a su GBW de 100 khz, que lo hace menos oscilatorio.

Su esquema es el siguiente:



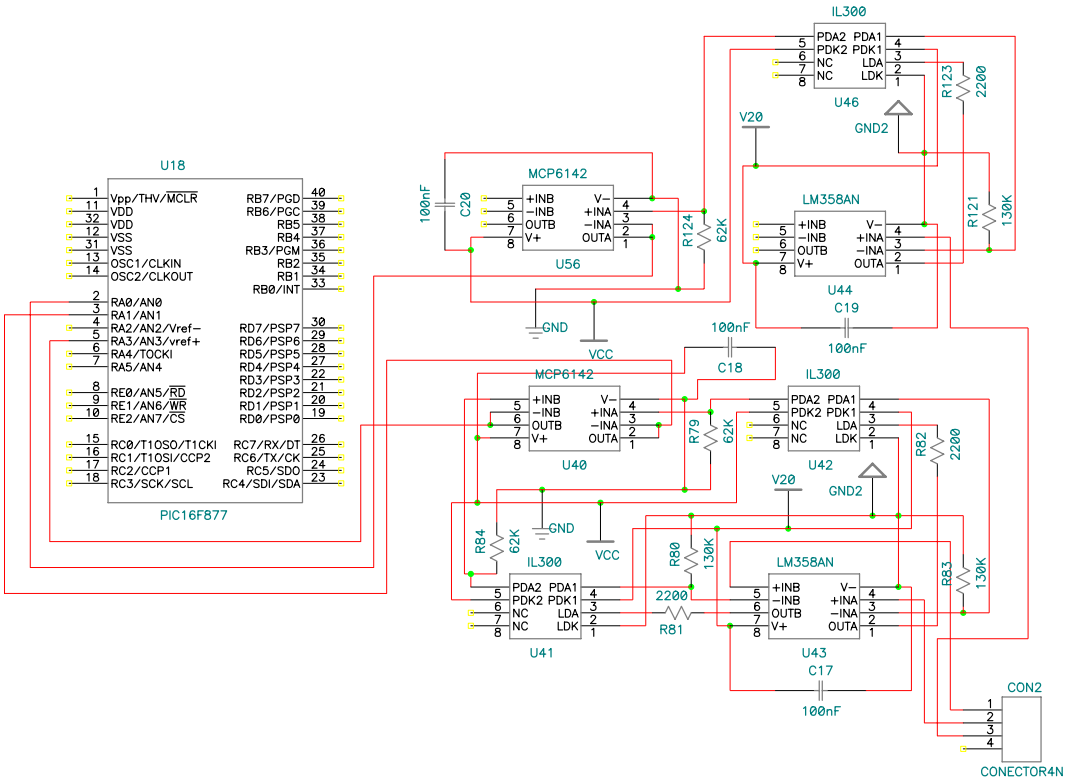
En la parte de 10 V se ha puesto el amplificador dual LM358A de Fairchild. Este tiene un rango de tensiones desde 3 hasta 32 V y está alimentado en el circuito a la tensión de 20 V. Previo a este se probo el amplificador NE5532 de Texas Instrument pero debido a su GBW de 10 Mhz resultaba muy oscilatorio para la aplicación a la que estaba destinado.

El esquema en este caso sería:



Para terminar se muestra a continuación el esquemático correspondiente

a todo el circuito de entradas al convertidor analógico-digital:



2.1.2 PLACA CABINA

Su función es la de interfaz CAN bus – PLC para un robot industrial en concreto un robot forestal (ROFOR) de la empresa Servicios Forestales.

El robot se compone de una cabina de mando y un brazo mecánico en el extremo del cual tiene un cabezal robotizado. Esta placa está situada en la cabina de mando de dicho robot.

El objeto de la placa es la comunicación de un PLC con un cabezal robotizado. Para ello tiene que recibir las entradas procedentes del PLC y enviarlas al cabezal, así como recibir la información del cabezal y enviarla al PLC. Todo esto se realizará con el bus de campo CAN.

Las entradas/salidas que tiene que tratar la placa de la cabina son las siguientes:

- Comunicación con PLC.

- 15 Entradas digitales rango de 0 a 24 V con un consumo de 400 mA (carga inductiva). Estas deben estar optoaisladas.
- 4 Salidas digitales 0-24 V
- 2 Salidas digitales 0-24 V a 500 hz.
- 2 Salidas digitales 0-24V a 2 Khz. procedentes de dos encoders desfasadas 90 grados.
- 3 Salidas analógicas 0-10 V con una precisión de 8 bits, con sensibilidad de milivoltios
- 2 Entradas analógicas 0-10 V con precisión de 8 bits, consumo máximo de 954 mA, que controlan una electroválvula.
- Se dispone de una alimentación de 24 V no estabilizada

- Comunicación mediante CAN-Bus con el cabezal robotizado. A este cabezal le son enviadas, previo tratamiento, todas las entradas procedentes del PLC y a su vez el cabezal suministra las salidas.

El corazón del sistema es un microcontrolador PIC18F458 el cual llevará las labores de control y manejo de los datos. Es un dispositivo de la gama media del fabricante Microchip. La elección de este tipo de microcontrolador está basada en criterios como una alta calidad, un bajo coste y un excelente rendimiento. Así como una facilidad de conseguir todo tipo de herramientas de desarrollo, gratuitas, incluso programadores, emuladores, etc.

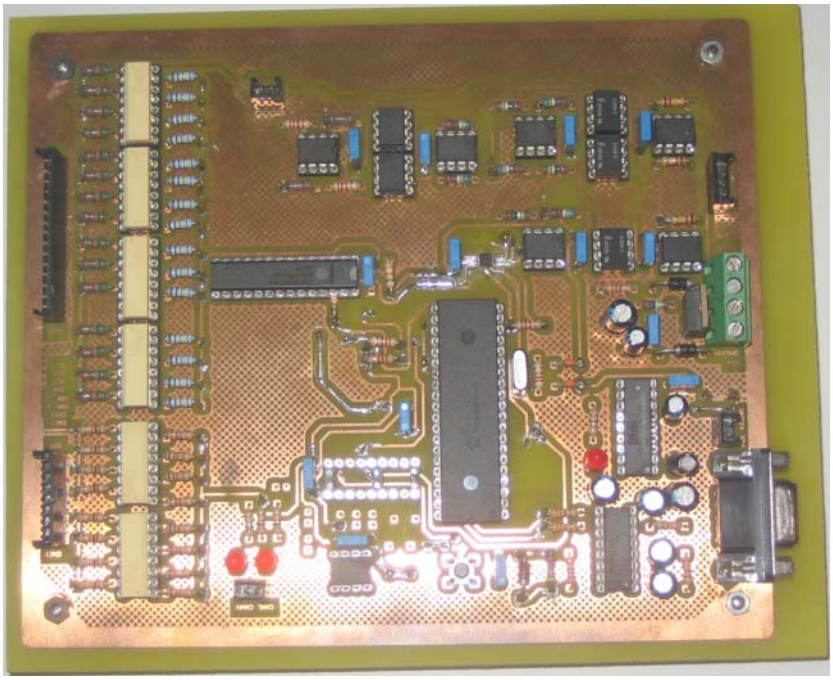
Los datos recibidos deben ser previamente adaptados a los niveles de tensión del microcontrolador y en el caso de las entradas/salidas analógicas debe procederse a una conversión digital-analógica, analógico-digital según estemos tratando salidas o entradas respectivamente. También el sistema necesita un aislamiento óptico total del exterior para su protección contra fallos, y una comunicación por puerto serie para poder tener un seguimiento del sistema. Por ello el microcontrolador requiere de la presencia de periféricos externos para la consecución de los objetivos. Estos periféricos son:

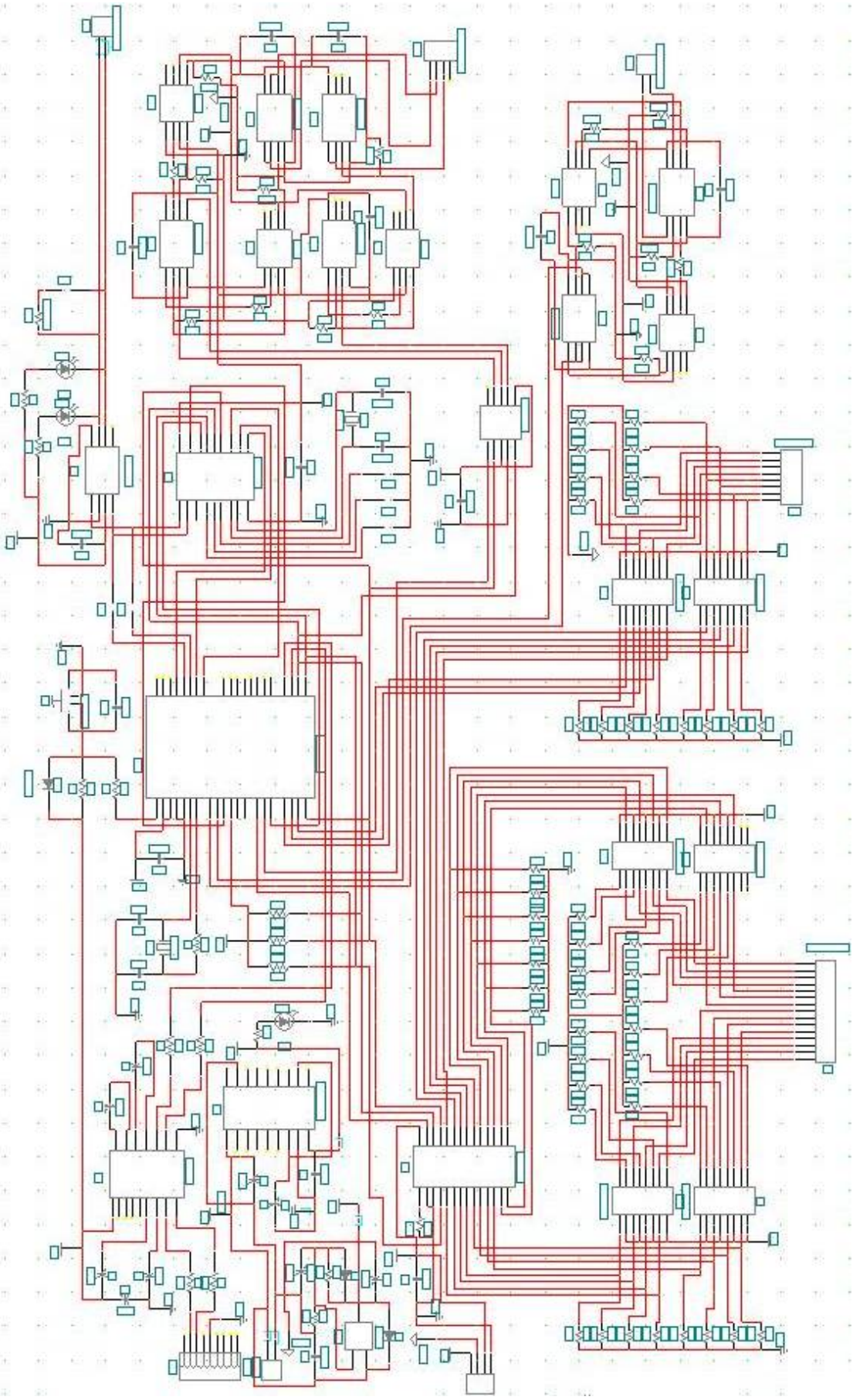
- 6 Optoacopladores digitales TLP 521-4

- 5 Optoacopladores lineales TIL 300
- 1 Convertidor digital analógico de 4 salidas MAX-5742
- 6 Amplificadores operacionales duales Rail to Rail
- 1 Expansor de E/S digitales MAX-6957
- 1 Convertidor 24-5 V optoacoplado DCR022405
- 1 Regulador de tensión 24-20V LM317
- 1 Interfaz CAN-Bus MCP2551
- 1 Interfaz RS232 para la comunicación por puerto serie MAX232

Para las comunicaciones entre el PIC y los periféricos se utiliza el bus SPI. Esto da al sistema una gran versatilidad al poderse añadir multitud de periféricos sin consumir las entradas/salidas del PIC.

Pasamos ya a la descrpción detallada de cada uno de los circuitos que componen la placa así como de los integrados involucrados en dichos circuitos. Dado que muchos de éstos son totalmente análogos a los de la placa del cabezal nos limitaremos a comentar aquellos circuitos que no está presentes en la placa antes mencionada. En concreto se trata de los circuitos de comunicación serie y de salidas del DAC. A continuación podemos ver una figura de la placa así como el esquemático completo de la misma:





2.1.2.1 Cicuito comunicación serie

Este circuito se encarga de la comunicación serie entre el microcontrolador y el PC. Esta comunicación es full-duplex o bidireccional asíncrona. Para la realización de este tipo se utiliza el protocolo RS-232, donde cada palabra de información o dato se envía independientemente de las demás.

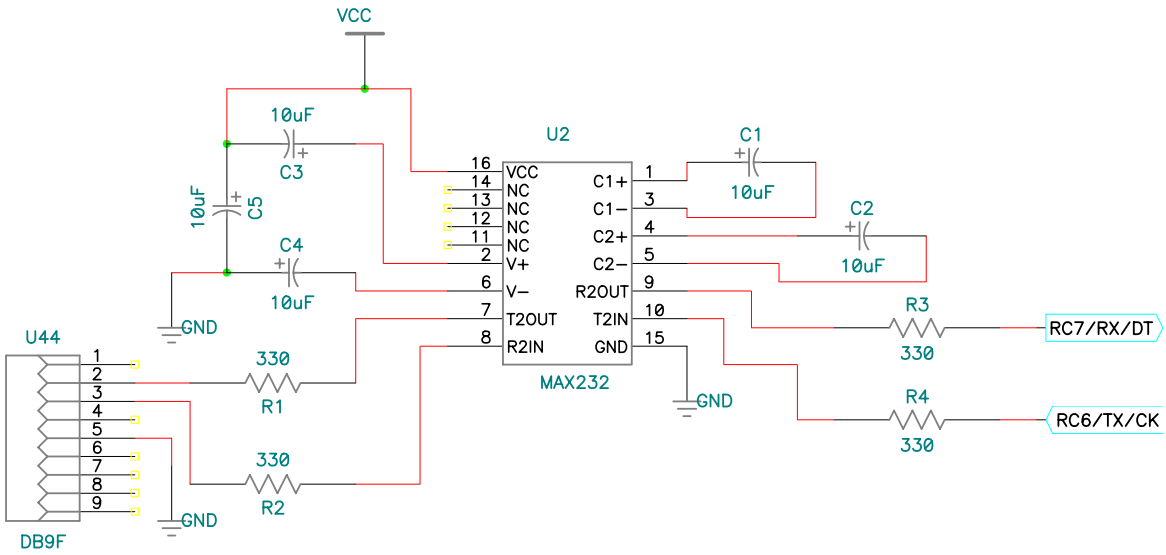
Está formado básicamente por el integrado MAX232, que se encarga de convertir los niveles lógicos TTL presentes en el pin 10 de transmisión (Tx), a niveles lógicos RS-232, que se obtienen por el pin 7 (TxD) y se aplican al conector serie DB9F. La señal de recepción RxD llega desde el canal serie a el pin 8 con niveles RS-232 que son convertidos a niveles TTL y se obtienen por el pin 9 (Rx).

Tanto la señal de recepción Rx como la de transmisión Tx se controlan desde los pines RC6 y RC7 cuando se está trabajando con un microcontrolador que posea un módulo USART por hardware, capaz de soportar la comunicación serie sincrona y asíncrona.

Para el correcto funcionamiento del circuito MAX232, se le conectan 4 condensadores electrolíticos (C1, C2, 3, C5) de 10uF respetando la polaridad indicada en el. También se incluye un condensador de 10uf (C4) entre Vcc y GND para mantener la estabilidad de la tensión de alimentación en el funcionamiento de la comunicación serie.

Por ultimo se incluyen 4 resistencias limitadoras de 330Ω, para los canales de transmisión y recepción, tanto en los niveles TTL como en los niveles RS232.

El esquemático completo del circuito de comunicación serie sería el que se muestra a continuación:



2.1.2.2 Circuito salidas DAC

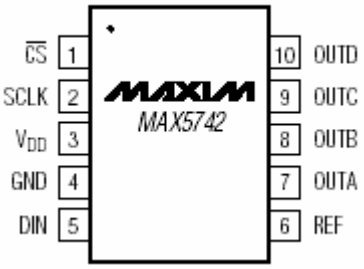
Básicamente son necesarias dos etapas en este circuito. Por un lado el DAC encargado de realizar la conversión digital-analógica de los datos que recibe a través del PIC; y por otro, la etapa de adaptación de los niveles de tensión de la señal de salida del DAC (0V-5V) a los niveles de salida de la placa (0V-10V). Se ha elegido como DAC el integrado MAX-5742. En cuanto a la etapa de adaptación de niveles se ha optado por un circuito análogo al empleado para las entradas analógicas. Pasamos a continuación a comentar con más detenimiento tanto el DAC como la etapa de adaptación.

El MAX-5742 es un convertidor analógico de bajo consumo, 12 bits, cuatro canales y salidas buffereadas. Se ofrece en un único encapsulado µMAX de 10 pines y tanto en rango de temperatura civil como militar. El ancho rango de voltaje de alimentación (+2.7V a +5.5V) y los 229µA de corriente de alimentación lo hacen ideal para aplicaciones de baja potencia y bajo voltaje. La entrada de referencia del MAX-5742 acepta un rango de voltaje desde 0 hasta Vdd. Finalmente indicar que el DAC ofrece comunicación serie compatible con los protocolos SPI, QSPI y MICROWIRE.

Algunas características del DAC pueden resumirse a continuación:

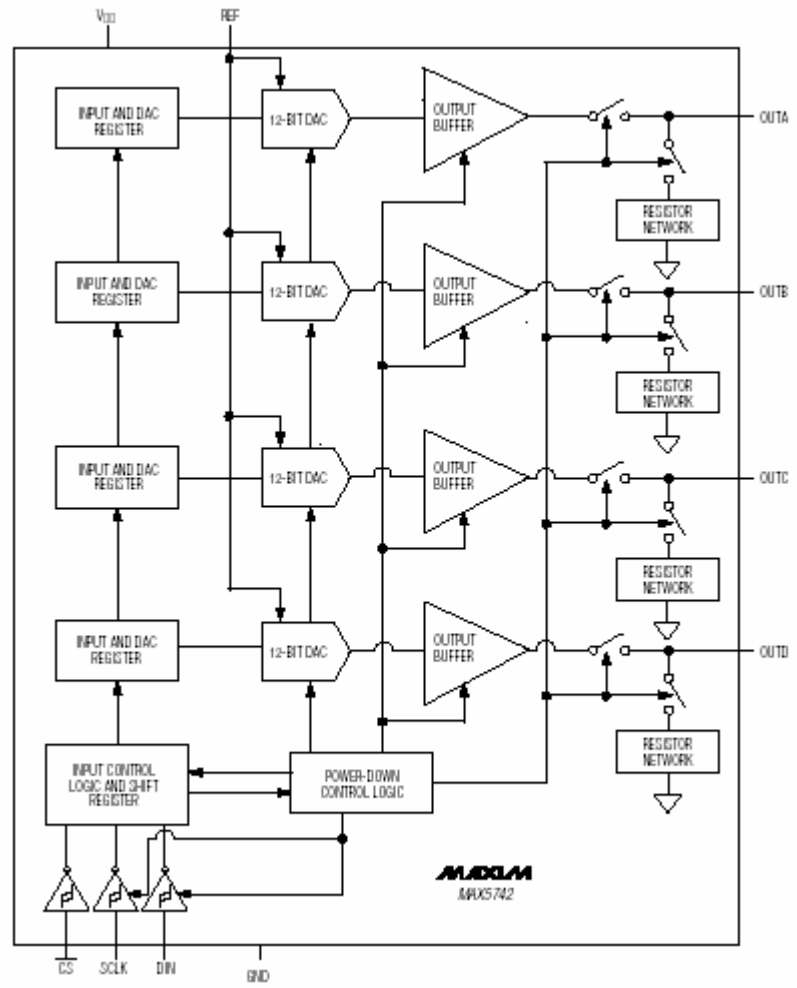
- Consumo de potencia muy bajo:
  - 229µA cuando Vdd es de +3.6V
  - 271µA cuando Vdd es de +5.5V
- Rango de tensión de alimentación de +2.7V a +5.5V
- Encapsulado µMAX de 10 pines
- Corriente de power-down de 0.3µA
- Tres impedancias de power-down seleccionables por software (100kΩ,1kΩ y alta impedancia)
- Interfaz serie compatible con los protocolos SPI,QSPI y MICROWIRE, a 20MHz
- Entradas lógicas Schmitt-Trigger para interfaz directa con optoacopladores
- Amplificadores de salida rail-to-rail

A continuación se muestra una figura con la distribución de los pines sobre el encapsulado y una breve descripción de los mismos en forma de cuadro:



PIN	NAME	FUNCTION
1	$\overline{CS}$	Chip-Select Input
2	SCLK	Serial-Clock Input
3	V_{DD}	Power-Supply Input
4	GND	Ground
5	DIN	Serial Data Input
6	REF	External Reference Voltage Input
7–10	OUTA –OUTD	DAC Voltage Outputs. Power-on reset sets DAC registers to zero, and internally connects OUT to GND with 100kΩ resistor.

Como ya se ha dicho antes el DAC tiene cuatro canales de conversión digital-analógica. Cada canal implementa una arquitectura de cadena de resistencias;la cual convierte una palabra de entrada digital de 12 bits en un voltaje de salida analógico equivalente y proporcional al voltaje de referencia aplicado. El MAX-5742 reparte la entrada de referencia entre los cuatro canales. El diagrama funcional del DAC es el siguiente:



La configuración y control del DAC es muy similar a la que ofrecía el expansor MAX-6957 que ya se vió en apartados anteriores. Básicamente el proceso consiste en escribir palabras de 16 bits mediante alguno de los protocolos serie con los que es compatible el chip. En nuestro caso se ha empleado el protocolo SPI para la comunicación entre el DAC y el PIC. El datasheet ofrece unas tablas tanto para la configuración como para la

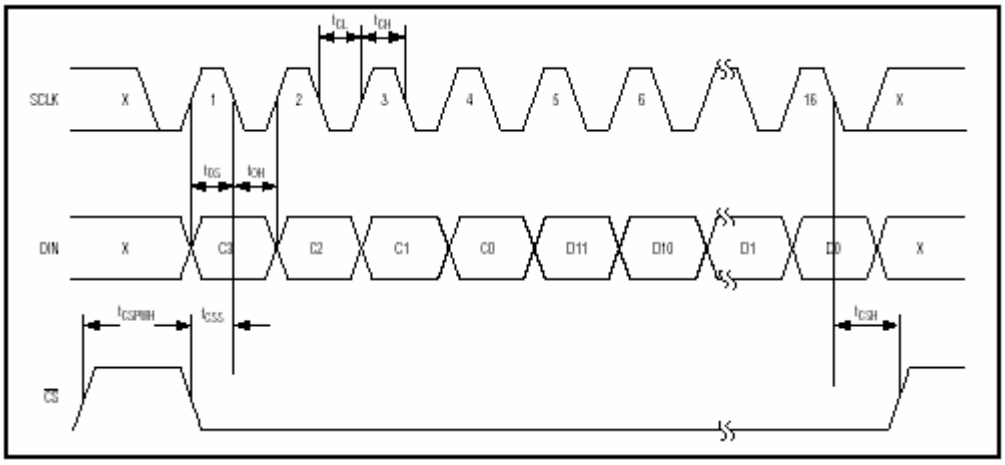
conversión y control del DAC. Dado que el DAC convierte datos digitales de 12 bits; los cuatro bits más significativos de la palabra de 16 bits de entrada se emplean como comando de control y los 12 menos significativos como datos. Esta palabra de 16 bits entra en un registro de desplazamiento interno del DAC con cada flanco de subida de la señal de reloj y entre un pulso de bajada y otro sucesivo de subida de la señal CS. Esta señal es activa a nivel bajo como puede deducirse de la descripción de la misma. Asi pues el proceso de escritura puede resumirse en los siguientes pasos:

- Tomar SCLK a nivel bajo
- Llevar CS a nivel bajo. Esto habilita el registro de desplazamiento de 16 bits
- Introducir 16 bits en la entrada DIN (primero D15 y por último D0) teniendo en cuenta los tiempos de establecimiento y espera.
- Llevar CS a nivel alto
- Tomar SCLK a nivel bajo

Un ejemplo de diagrama de tiempos del DAC así como la estructura o formato de las palabras de entrada serían:

CONTENTS OF SHIFT REGISTER

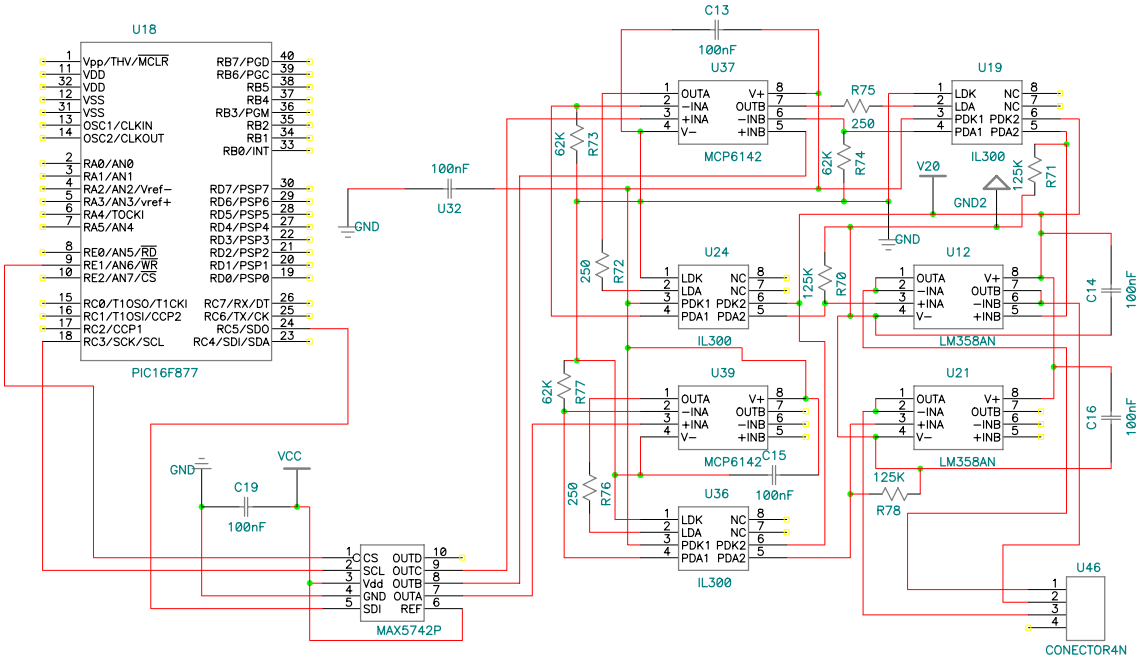
D15 (MSB)															D0 (LSB)
C3	C2	C1	C0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0



Por último recordar que tanto las palabras de configuración como de programación del DAC pueden consultarse en las tablas que ofrece el

fabricante en el datasheet del integrado.

Una vez comentado el DAC pasamos a ver la etapa de adaptación de niveles de tensión. A su salida el DAC ofrece una señal analógica de 0V-5V que habrá que convertir a los niveles de 0V-10V.

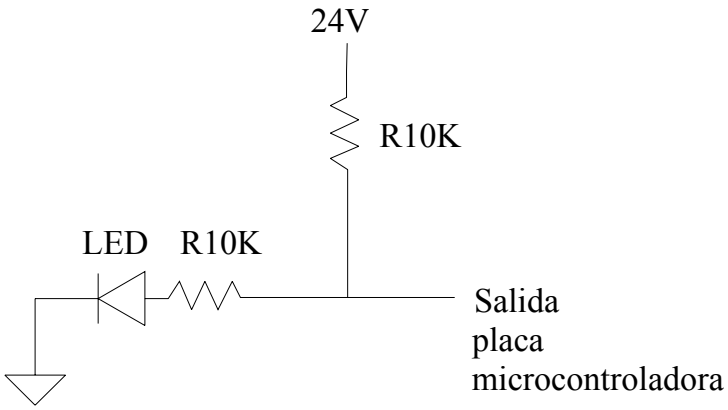


2.2 PLACAS E/S DIGITALES

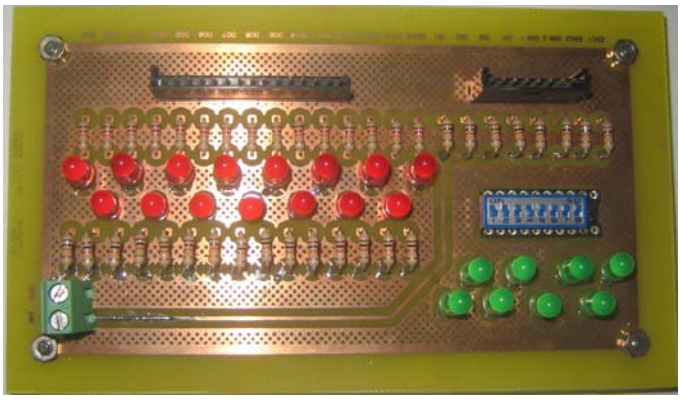
Estas placas se han desarrollado para probar las entradas y salidas digitáles del las placas microcontroladoras. Estas placas se alimentan de una tensión de 24V suministrada por la placa microcontroladora a la que se conectan; que actua como fuente de tensión.

Se componen basicamente de interruptores y diodos LED. En concreto dispone de 15 interruptores que se aplican a las entradas digitales de la placa microcontroladora de la cabina en un caso; y de 6 en la placa de la cabeza. Cuando están en ON introducen un '1' lógico por las entradas digitales. En caso de estar en OFF introducen un '0' lógico. Cada interruptor tiene asignado un LED que indica que interruptor está en ON en cada momento (LED's verdes). Los interruptores están conectados a una resistencia de 10 kΩ. Estas están conectadas en su otro extremo a una tensión de 24 V por lo que cuando se pulsa un interruptor se aplica una tensión de 24 voltios a la entrada de la placa del PIC. Al mismo tiempo el interruptor conecta con los LEDS, de forma que cuando esta en ON, cierra el circuito y se encienden estos.

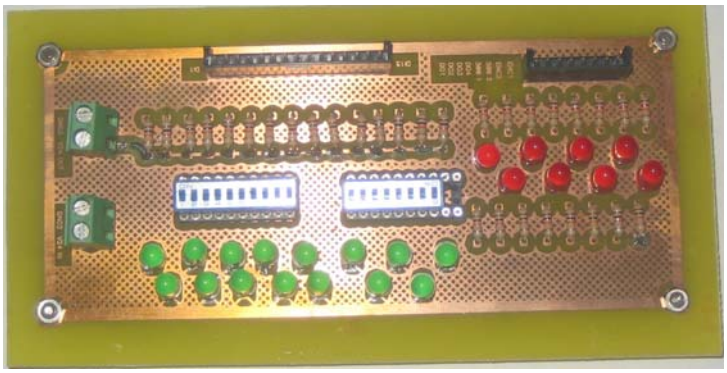
La parte de las salidas esta formada por los LEDS rojos. Estos se iluminan cuando reciben un '0' lógico. Estas están conectadas a las 4 salidas de 24 V de la placa microcontroladora, a las 2 de 500 Hz y a los Encoders en el caso de la placa de la cabina; y a las 15 salidas digitales en el caso de la placa de la cabeza. Cuando reciben por una de estas salidas una tensión de 24 V el LED permanecerá apagado. Su esquema es el siguiente:



Para terminar veremos una imagen de cada una de las placas de E/S. En primer lugar vemos la placa de E/S de la cabeza:



En cuanto a la placa de la cabina:



2.3 PLACAS PROGRAMADORAS

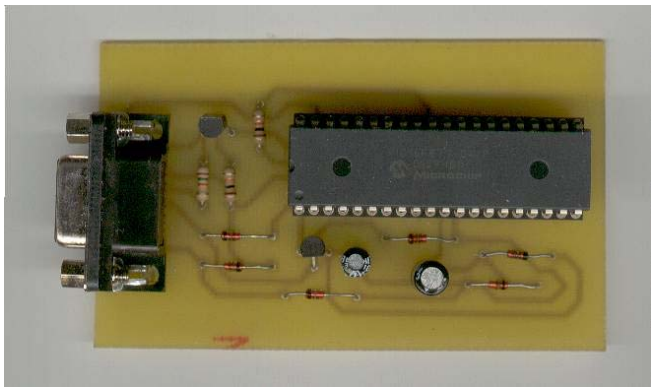
A continuación se comentarán las dos posibilidades barajadas a la hora de programar los PIC's de cada una de las placas microcontroladores. Se comentarán brevemente las características de cada una y se analizarán las ventajas de una respecta a la otra.

2.3.1 PROGRAMADOR JDM

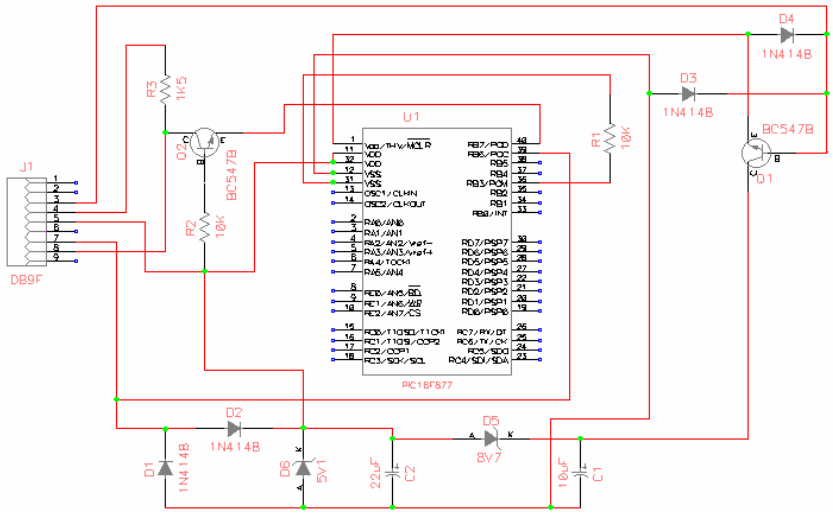
El programador JDM, tiene como principales características:

- Alimentación necesaria para la grabación de alto voltaje (comprendida entre 12 y 14 voltios), proporcionada por el PC.
- Grabación serie síncrona, utilizando el pin **RB6/PGC** para la introducción desde el PC de los impulsos de reloj y el pin **RB7/PGD** para los datos.

A continuación se muestra una imagen de dicho programador:



En cuanto al esquemático del circuito; lo vemos a continuación:



2.3.2 DEBUGGER ICD

Como alternativa al programador visto el fabricante del software ofrece una herramienta que permite disponer al mismo tiempo de programador de aplicaciones y de debugger de las mismas. Se trata del ICD del cual podemos ver a continuación una ilustración:

El debugger se conecta mediante puerto USB al PC y mediante un conector RJ-12 a la placa de grabación de aplicaciones para pasar el programa al PIC. Este método ofrece como ventajas fundamentales respecto al grabador JDM:

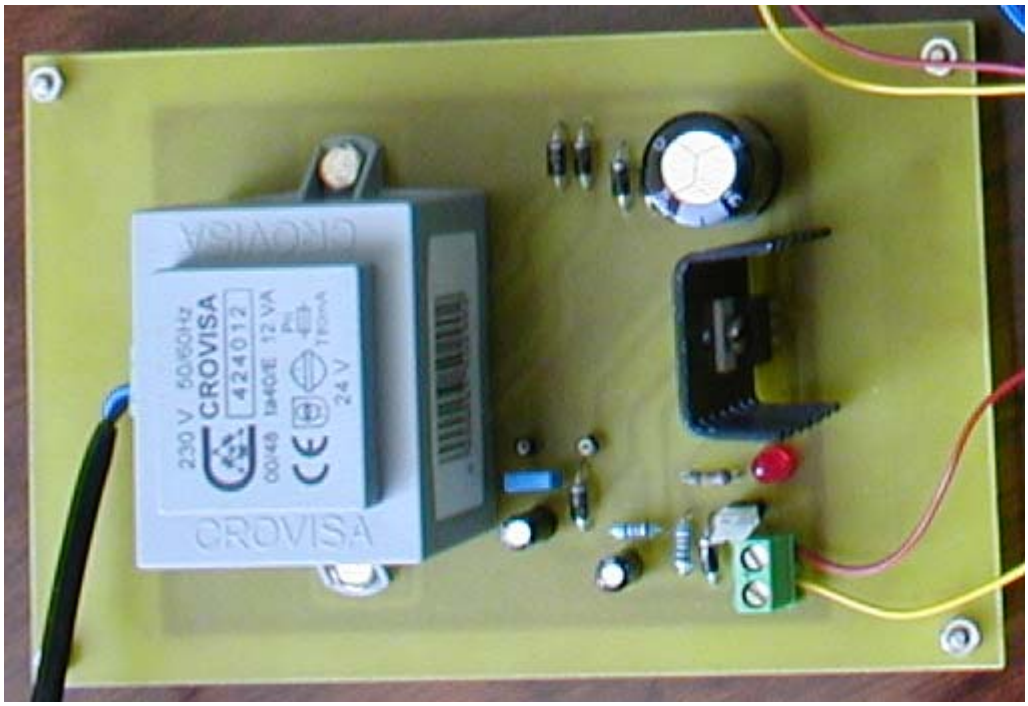
- Posibilidad de debugging de aplicaciones
- Mayor velocidad de grabación de aplicaciones

Además del debugger y la placa grabadora es necesario un software específico que monitorice tanto el proceso de grabación de las aplicaciones, como el de debugging. Dicho software será comentado en el siguiente apartado del documento.

La placa para grabar aplicaciones en el PIC mediante el ICD consta fundamentalmente del conector RJ-12, un circuito de alimentación y otro de reloj. El circuito de reloj es exactamente igual que el visto en las placas microcontroladores ya comentadas. En cuanto al circuito de alimentación se trata de un simple regulador alimentado con una tensión de 9V a 12V aproximadamente; el cual da a su salida la tensión de 5V requerida por el PIC.

2.4 PLACA ALIMENTACIÓN

Esta placa se encarga de alimentar a 24 voltios, a la placa de ensayos y a la microcontroladora. Consta de un transformador, un puente de diodos y un regulador de tensión ajustable LM317. El funcionamiento de este regulador así como su esquemático esta explicado en la sección de la fuente de alimentación de la placa microcontroladora.



3 SOFTWARE

Parte muy importante en el proyecto es sin duda el software. Bajo esta denominación nos referimos tanto al conjunto de programas que hacen posible la grabación del código en el PIC o la compilación de dicho código; como el propio código de programa que alberga el PIC y que gobierna el funcionamiento del sistema.

En primer lugar se tratarán los programas que hacen posible la creación, compilación, debugging y grabación del código de programa que ha de ejecutar el PIC. La creación, compilación y debugging del código se realiza con el programa Picc Compiler del fabricante CCS. Se explicará a continuación el funcionamiento básico del programa; así como las opciones y utilidades más interesantes.

Más tarde se hablará del programa ICD que es el que permite grabar las aplicaciones en el PIC.

Tras estos apartados se comentará el código de programa que ejecutará el PIC y que permitirá que el sistema funcione de acuerdo a los requerimientos del mismo. Se explicará con detalle el trato particular a cada una de las entradas y salidas del sistema. . Pasamos pues a ver todos estos aspectos del software con detalle.

3.1 APLICACIONES

Como ya se ha dicho en primer lugar hablaremos de los programas o herramientas software que han facilitado la elaboración y puesta en marcha del código de programa que ejecutará el PIC de cada una de las placas, para el correcto funcionamiento del sistema.

Se trata en concreto de dos aplicaciones. Por un lado el compilador Picc Compiler y por otro el programa de grabación de PIC's ICD. Pasamos a ver cada uno de ellos.

3.1.1 COMPILADOR PCWH

Tradicionalmente se ha recurrido al lenguaje ensamblador para la programación de microcontroladores. En concreto el fabricante del PIC, Microchip; ofrece una herramienta gratuita llamada MPLAB para el desarrollo de aplicaciones a ejecutar en cualquiera de los PIC's que ofrece su amplia gama. En este caso el programa emplea el código ensamblador para la realización de aplicaciones destinadas al PIC.

Está sobradamente reconocida la eficiencia del código escrito en ensamblador con respecto a códigos escritos en lenguajes de alto nivel. En cambio también es evidente que el programar en ensamblador supone una tarea tediosa y en el caso de aplicaciones complejas que manejen gran cantidad de variables, interrupciones, etc…, puede ser fuente de muchos errores difíciles de depurar. Es en estos casos en los que; a costa de una ligera pérdida de eficiencia, cobran importancia los lenguajes de alto nivel como es el caso del lenguaje C. Además de facilitar la tarea de programar, el lenguaje C ofrece la posibilidad del empleo de punteros y estructuras de datos que hacen aún más sencilla la elaboración de rutinas.

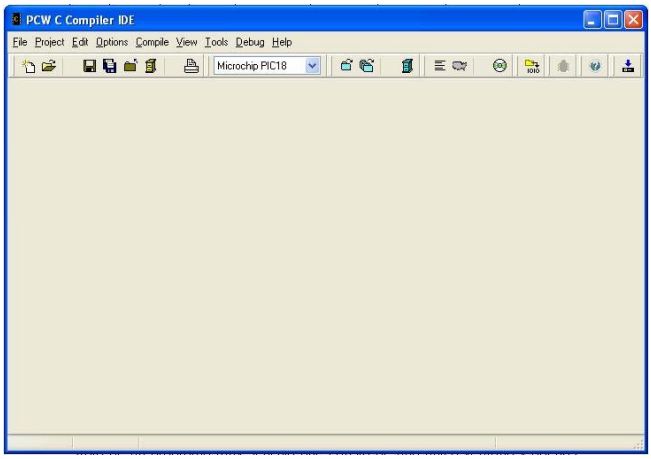
En nuestro caso se ha empleado el lenguaje C. Para ello la herramienta escogida ha sido el compilador Picc de la empresa CCS. Este compilador ofrece diferentes versiones del mismo. Por un lado se dispone de un compilador en línea de comandos válido para el sistema operativo MS-DOS; y por otro, uno válido para windows. Dentro de cada uno de estos compiladores existe otra distinción según sean válidos para PIC's de 12 o 14 bits; o para el PIC18. Así pues tenemos las siguientes versiones:

Característica	PCB	PCM	PCH	PCW	PCWH
Compilador en línea de comandos	Disponible	Disponible	Disponible	Disponible	Disponible
Programas ejemplo	Disponible	Disponible	Disponible	Disponible	Disponible

Drivers de dispositivos	Disponible	Disponible	Disponible	Disponible	Disponible
Interfaz MPLAB	Disponible	Disponible	Disponible	Disponible	Disponible
Windows IDE				Disponible	Disponible
Asistente de nuevo proyecto				Disponible	Disponible
Debugger				Disponible	Disponible
Árbol de llamadas y mapa de memoria	Disponible	Disponible	Disponible	Disponible	Disponible
Estadísticas				Disponible	Disponible
Utilidad de puerto serie				Disponible	Disponible
Soporte de 12 bits	Disponible			Disponible	Disponible
Soporte de 14 bits		Disponible		Disponible	Disponible
Soporte de PIC18			Disponible		Disponible

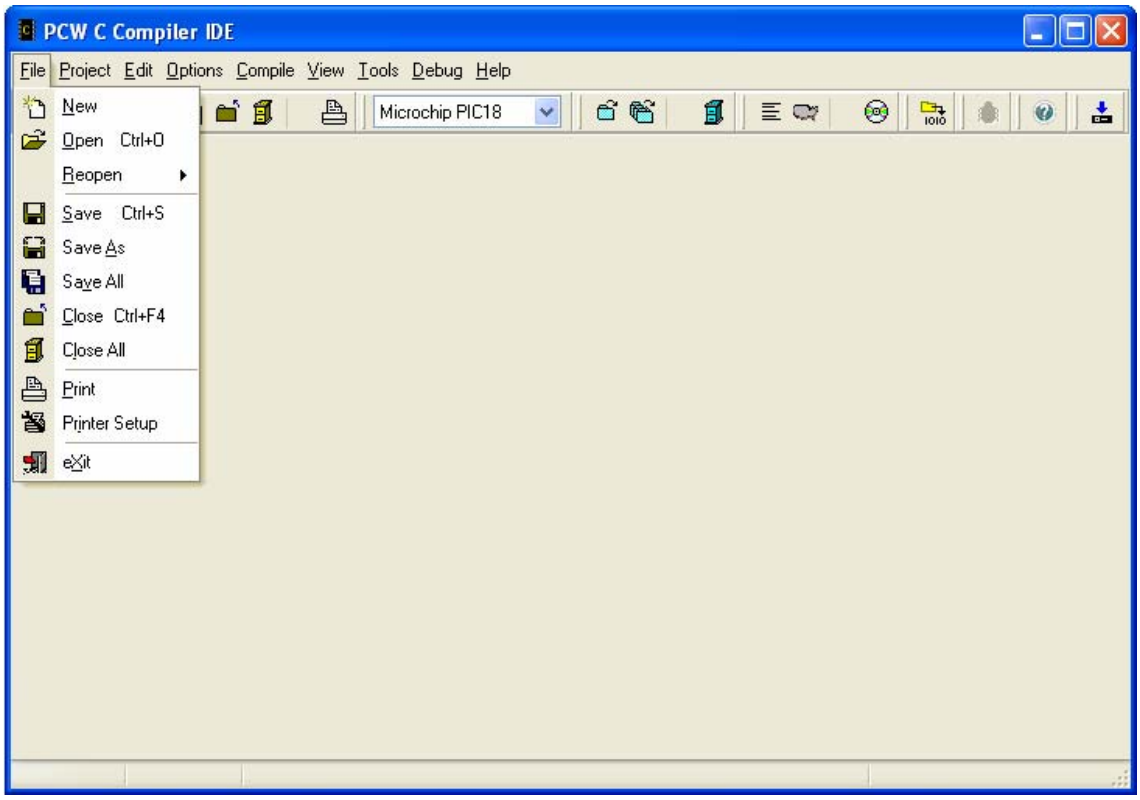
Para el proyecto se ha empleado el compilador PCWH puesto que se ha trabajado en entorno Windows y con el PIC18F458. Pasamos ya a comentar los aspectos más relevantes del programa.

La ventana principal del programa tiene el aspecto que podemos ver a continuación:



En la parte superior de la ventana se tiene el menú principal del programa y la barra de herramientas con algunas de las opciones más utilizadas como *Archivo Nuevo*, *Abrir Fichero*, *Guardar Fichero*, *Compilar*, *Debugger* o *Llamada al programa ICD.* Pasemos a ver aquellos submenús del menú principal más importantes.

En primer lugar veremos el menú *File*. Al desplegarlo tenemos lo siguiente:

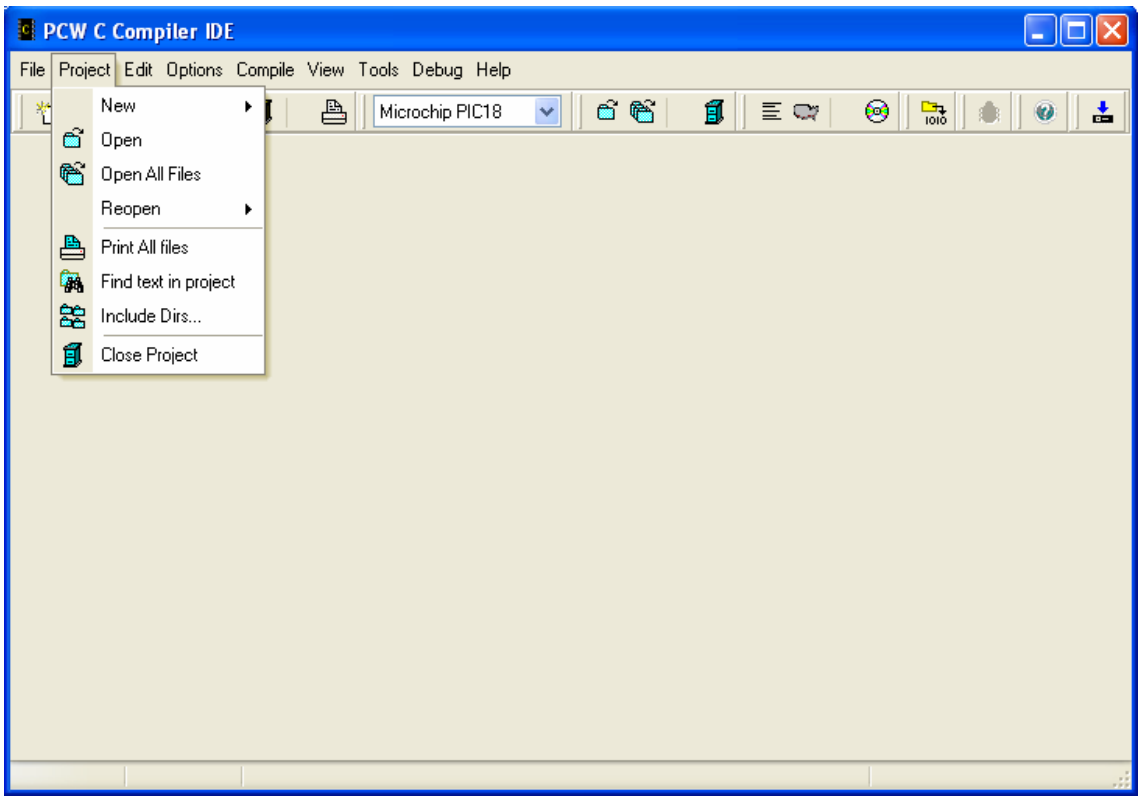


Las distintas opciones que se nos ofrecen son las que se detallan a continuación:

- New: crea un nuevo fichero fuente
- Open: abre un fichero guardado
- Reopen: abre uno de los diez últimos ficheros abiertos que son mostrados en una lista
- Save: guarda la versión actual del fichero abierto sobrescribiendo la anterior

- Save as: guarda la versión actual del fichero abierto pudiendo especificarse tanto ubicación como nombre del mismo
- Save all: guarda todos los ficheros asociados a un proyecto
- Close: cierra el fichero actual. Si ha habido cambios preguntará si éstos se salvan o no
- Close all: cierra todos los ficheros asociados a un proyecto. Si ha habido cambios en alguno irá preguntando que cambios queremos salvar y cúales no
- Print: Imprime el fichero actual
- Printer setup: permite configurar las opciones de impresión
- Exit: salir del programa

Otro submenú destacado es el submenú *Project*. Al desplegarlo pueden verse las siguientes opciones:



Nuevamente detallaremos la funcionalidad de cada una de las opciones:

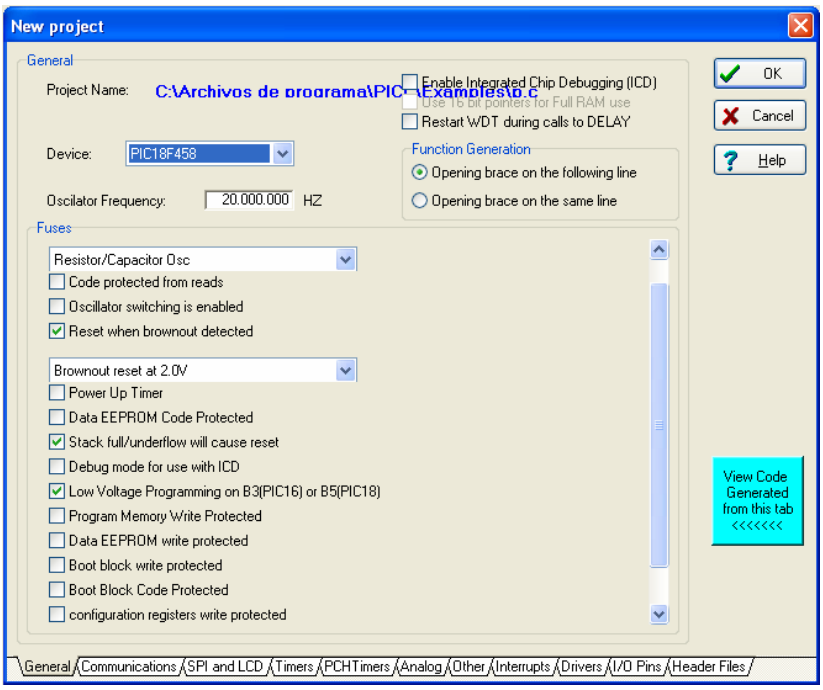
- New: permite crear un nuevo proyecto. Ofrece dos opciones; emplear el asistente, o bien crear los ficheros manualmente. Más tarde se verá con detalle la opción de creación de proyectos

mediante el asistente.

- Open: permite abrir un proyecto creado previamente
- Open all files: abre todos los ficheros asociados a un proyecto
- Reopen: permite abrir uno de los últimos diez proyectos abiertos que son mostrados en una lista
- Print all files: imprime todos los ficheros asociados a un proyecto
- Find text in Project: permite buscar una cadena de caracteres entre todos los ficheros de un proyecto. Útil si se quiere buscar una definición de una variable o una sentencia concreta por ejemplo.
- Include Dirs: establece el directorio donde el programa busca los ficheros de dispositivos y drivers
- Close Project: cierra el proyecto actual

Antes de pasar al siguiente submenú conviene detenerse en el comentario del asistente de creación de proyectos. Constituye una herramienta muy útil para la configuración de los periféricos del PIC, configuración de las E/S, elaboración de rutinas de interrupción, etc…

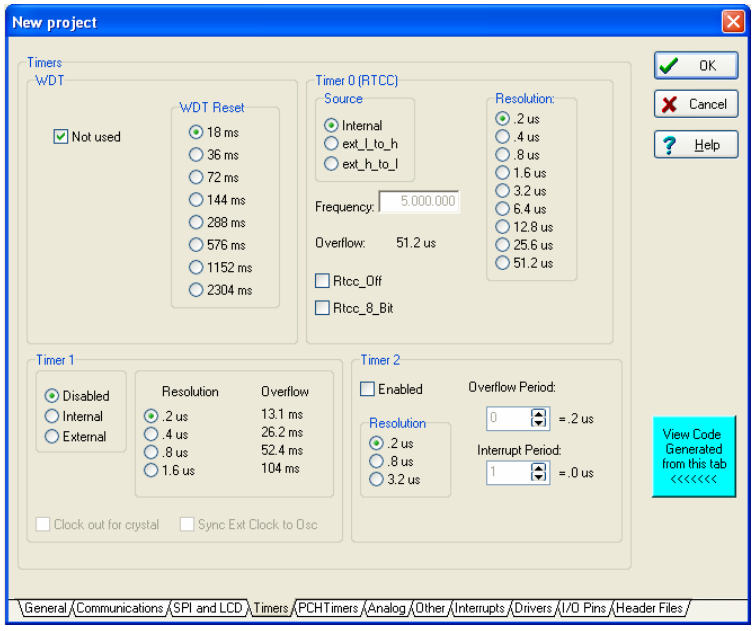
Para acceder al asistente se elige la opción *New* dentro del menú *Project* y posterioremente *PIC Wizard*. A continuación se nos preguntará por el nombre y ubicación del proyecto que se creará. Una vez elegidos aparece la siguiente ventana:

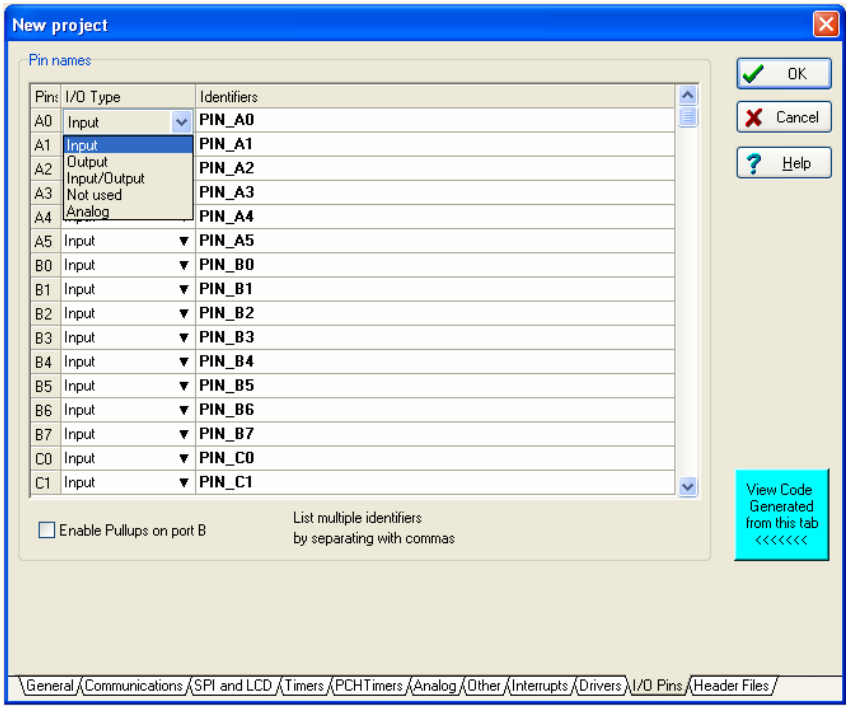


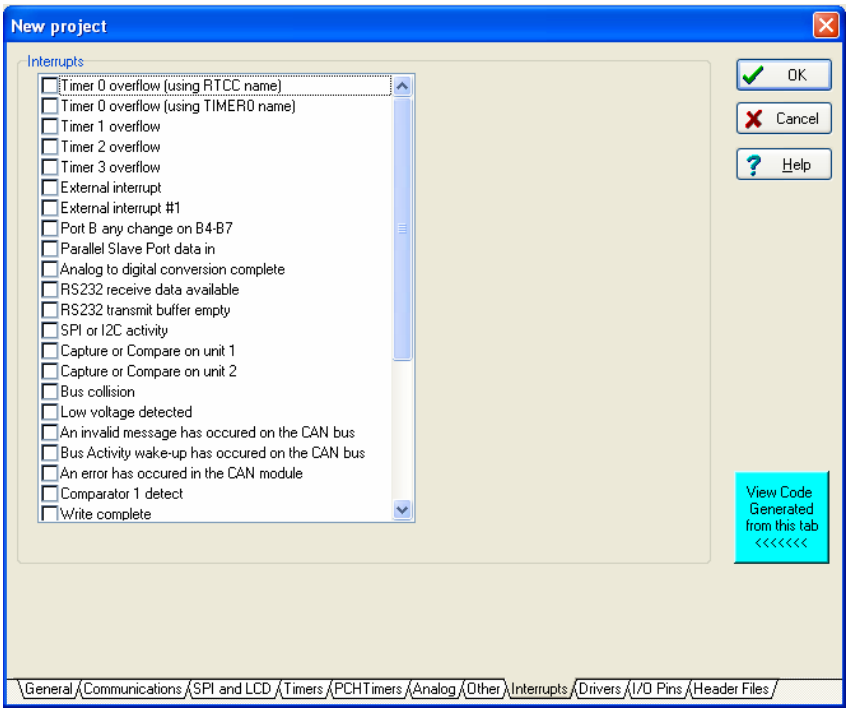
Esta ventana mostrada corresponde a la pestaña *General* de todas las que ofrece el asistente y que pueden verse en la figura. Aquí se indica el nombre del proyecto elegido anteriormente y se permite configurar ciertos parámetros. Entre ellos el PIC a utilizar, la frecuencia y tipo de reloj; y otras funciones auxiliares como habilitación del debugger, protección contra escritura de la memoria de programas, etc… Además de estas opciones que son particulares de cada pestaña; a la derecha de la ventana tenemos botones para aceptar la configuración del asistente, rechazarla y salir, pedir ayuda o ver el código generado por la configuración que se elige en cada momento. Estos botones de la derecha están disponibles en todas las pestañas.

Además de la pestaña *General* están disponibles otras diez que nos permiten como ya se ha dicho múltiples configuraciones de los periféricos del PIC. No obstante conviene revisar siempre el manual del PIC pues existen configuraciones no contempladas por el compilador como ocurre en el caso del módulo SPI. En esos casos habría que escribir bit a bit la configuración deseada en los registros correspondientes.

Entre estas diez pestañas destacar por ejemplo la de activación de interrupciones por su utilidad así como la de Timers o módulo analógico. Veamos cómo serían estas ventanas:

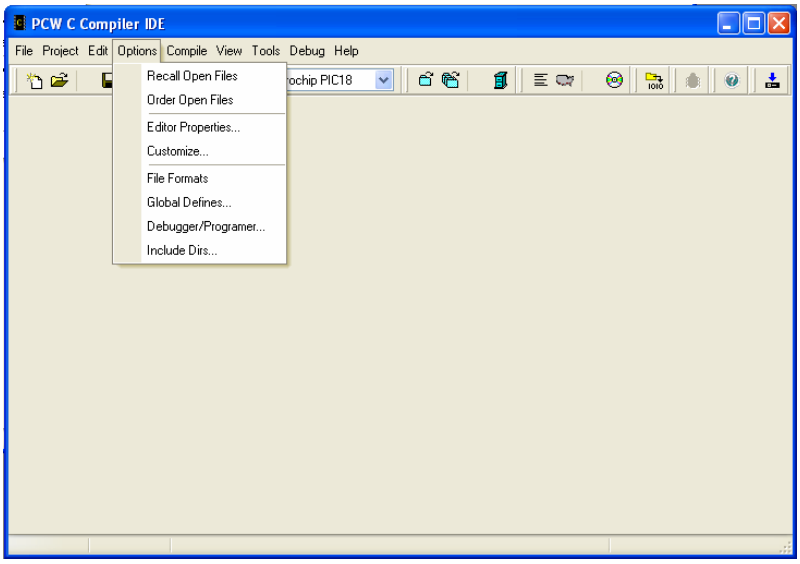


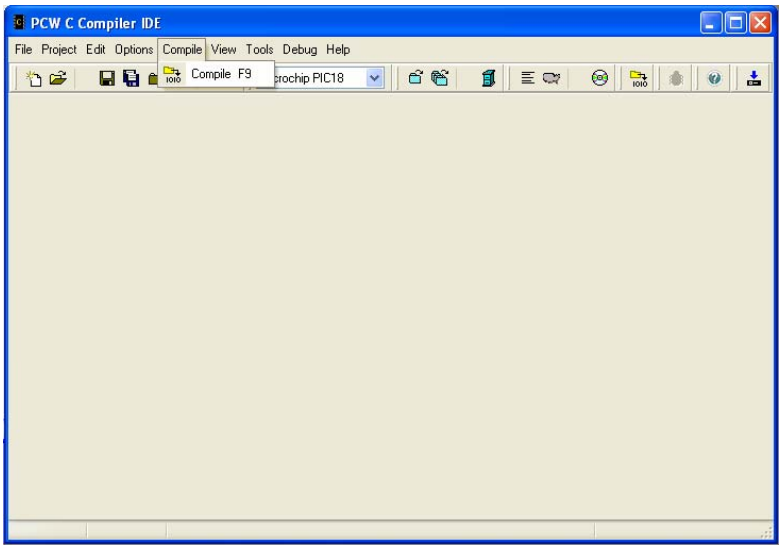




Como puede verse todas estas pestañas son fáciles de configurar y entender por lo que no se comentarán con detalle.

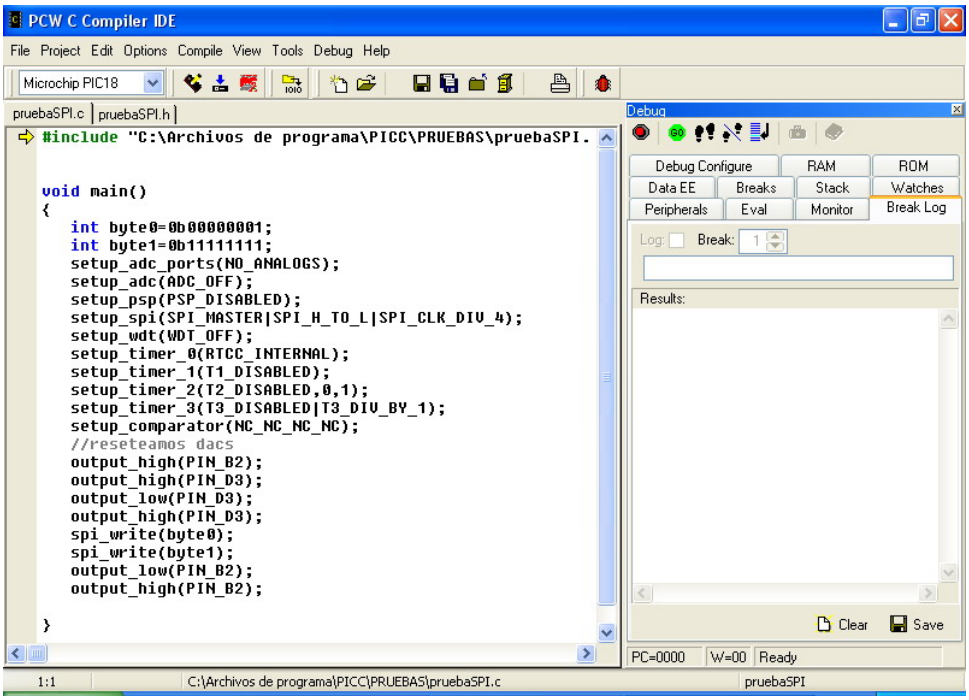
Seguimos viendo algunos de los submenús del programa comentando en este caso los submenús *Options* y *Compile*:





El submenú *Compile* sólo tiene la opción de compilar el código abierto bien sea un fichero individual o bien un proyecto. También puede realizarse la compilación del código de forma más rápida a través del botón de la barra de herramientas. En cuanto al submenú *Options* sólo se destacará la opción *Debugger/programmer* que permite configurar el acceso tanto al programa de grabación de aplicaciones ICD, como al debugger; desde el compilador.

Por último hablaremos del submenú *Debug*.



Como vemos seleccionado el debugger a partir del menú *Debug* nos aparece a la derecha una nueva ventana en la que se ofrecen diversos botones de acción y pestañas para consultar información sobre el PIC. También se puede acceder a esta ventana de debugger a partir del acceso directo que se tiene en la barra de herramientas del programa. En concreto se trata del último icono de la derecha.

Importante es tener en cuenta que para que pueda realizarse debugging es necesario disponer de un hardware específico bien en placa o bien externo a ella. En caso de usar un hardware externo (como es el caso del módulo ICD-U empleado para la programación en este proyecto) se pierde la posibilidad de simular el sistema real; ya que las E/S del circuito no las tendríamos disponibles.

En cambio en el caso de tener dispuesto en la placa el hardware necesario para el empleo del debugger, se podrá observar desde el ordenador el funcionamiento real de nuestro sistema. Para poder realizar esto en nuestro proyecto fue necesaria una modificación en el hardware inicial; modificación que fue comentada al final del apartado correspondiente al hardware de la

placa de la cabeza y que se realizó también en la de la cabina.

Pasamos ya a comentar la ventana del debugger. Veremos algunas de las posibilidades que nos ofrece:

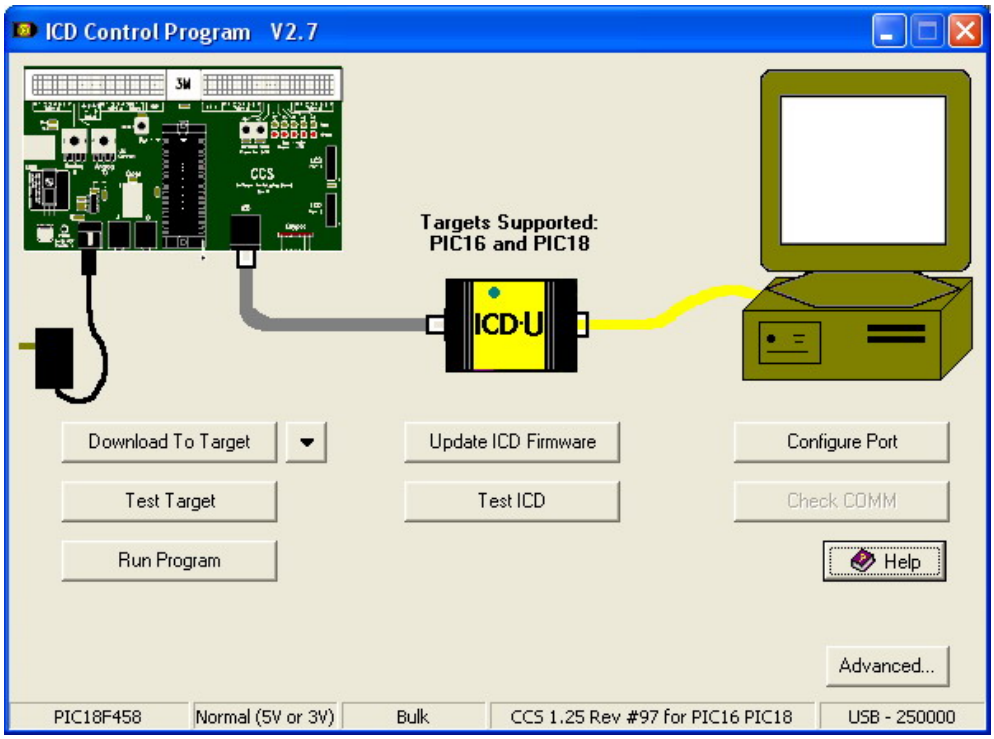
- Reset de la placa: permite resetear el sistema desde el ordenador
- Go: lanza la ejecución del programa que alberga el PIC
- Ejecución paso a paso: permite realizar una ejecución del programa que alberga el PIC sentencia a sentencia de manera que resulta muy útil para depurar errores en la programación. Entre cada paso puede consultarse el estado de los registros, cosa que no se podía realizar con la opción Go.
- Ejecución desde un punto marcado: permite realizar algo similar a la ejecución paso a paso pero de manera más eficiente si cabe. Mediante esta opción se realiza la ejecución del programa albergado en el PIC desde el principio hasta un punto marcado por nosotros en el código. Llegados a este punto podemos analizar el contenido de los registros y posteriormente marcarle otro nuevo punto de ruptura o bien continuar con la ejecución total del programa o paso a paso.

Una vez vistas las opciones de ejecución más destacadas; decir además de esto, que la ventana de debugger ofrece una serie de pestañas que nos permiten ver el estado de cada uno de los registros del PIC agrupados por categorías (RAM, ROM, periféricos, etc…); el estado de la pila, la configuración del debugger, un monitor del puerto serie, etc…

Asimismo en la parte inferior de la ventana se ofrece la posibilidad de salvar como fichero de texto el estado del PIC en un instante.

3.1.2 PROGRAMA GRABADOR DEL PIC

Para monitorizar el kit ICD de grabación de aplicaciones en el PIC es necesario el uso del programa que lleva el mismo nombre desde el PC. Se trata de un programa muy sencillo que consta de una única ventana y apenas opciones de configuración. El aspecto de dicha ventana es el siguiente:

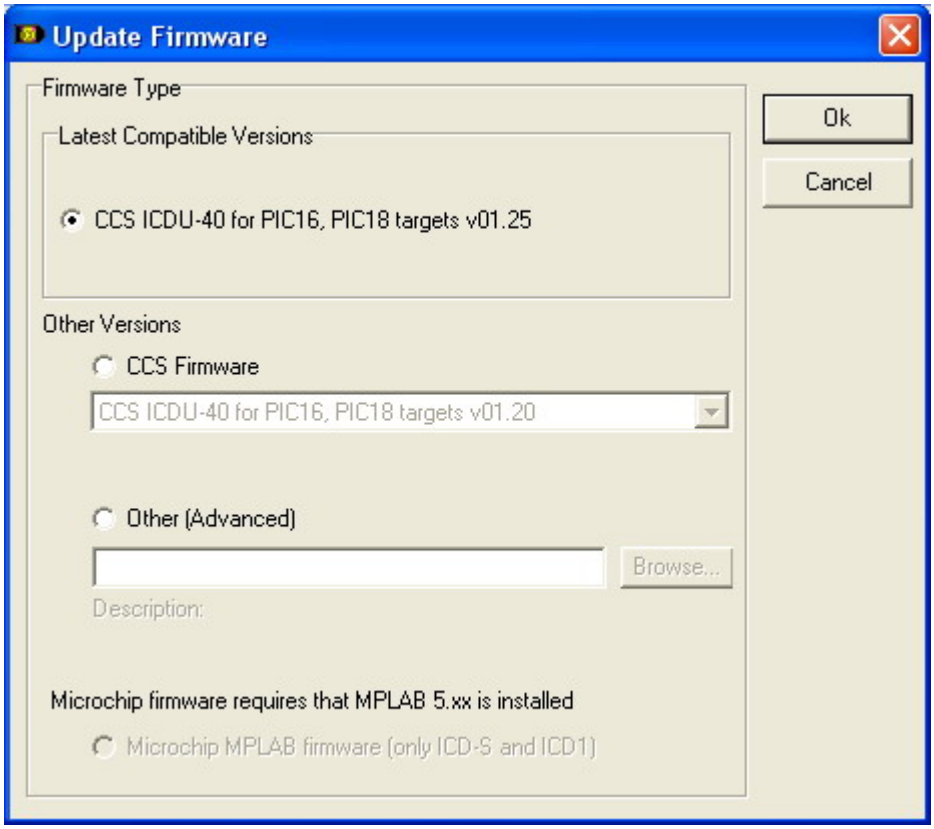


Como puede verse se trata de una herramienta muy intuitiva y de fácil manejo que se divide en tres bloques en cuanto a su funcionalidad y configuración.

Por un lado tenemos la parte correspondiente a la tarjeta o placa donde se encuentra ubicado el PIC. Puede ser el sistema definitivo o simplemente una placa programadora. El botón *Test Target* nos permitirá testear dicha placa para detectar posibles malas conexiones o falta de alimentación de la misma. Una vez aceptada y detectada la placa se puede proceder a la carga del programa. Para ello el botón *Download To Target* nos hará aparecer un cuadro donde indicar la ruta en la que se encuentra el fichero de texto con el código del programa a cargar o bien el proyecto completo que se quiere ubicar en el PIC. Por último odemos ejecutar desde aquí el programa cargado mediante el botón *Run Program* aunque resulta más conveniente hacerlo desde el debugger debido a las funcionalidades adicionales que éste ofrece. No obstante es otra forma de hacer funcionar nuestro PIC de acuerdo al programa cargado.

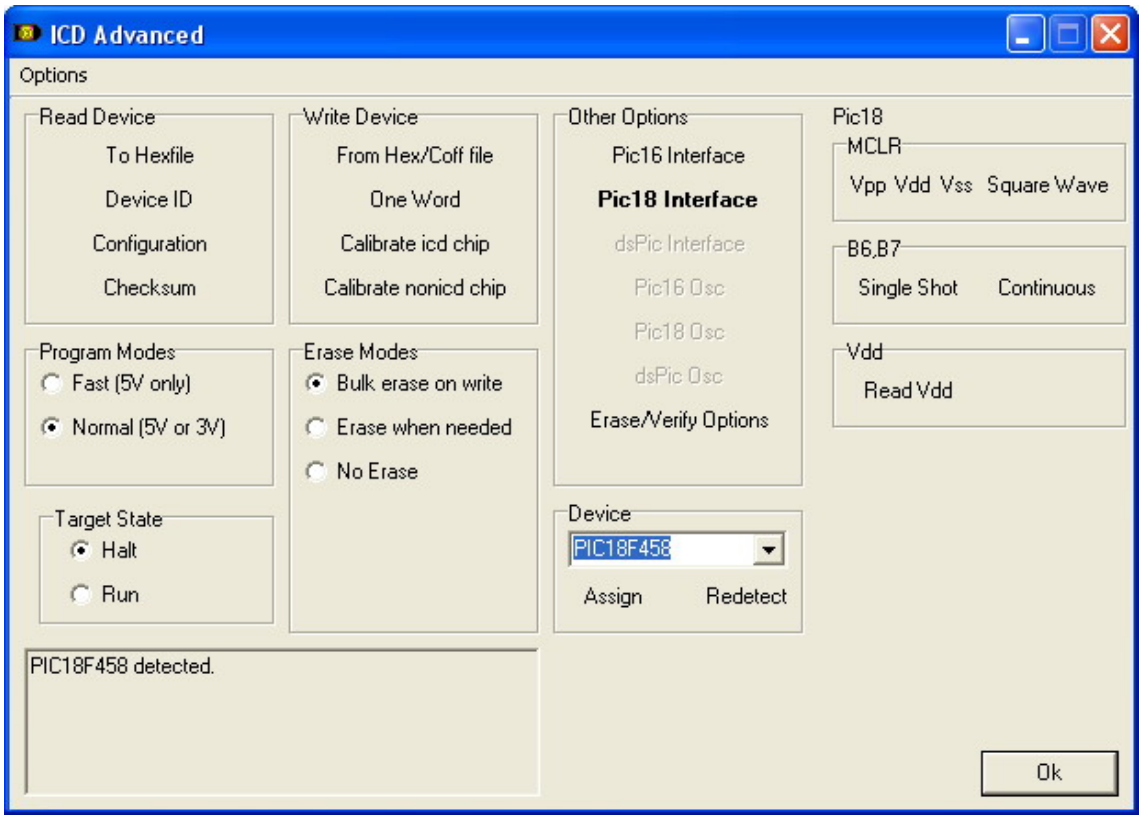
A continuación pasamos a hablar de lo referente al kit ICD-U. En este

caso las opciones son menos que en el caso anterior. Se nos ofrece comprobar que el kit se encuentra en buen estado mediante el botón *Test ICD*; o bien actualizar el firmware del dispositivo. Destacar llegados a este punto, la importancia de la actualización del firmware para el correcto funcionamiento del dispositivo sobre todo en cuanto a su funcionalidad como debugger. A continuación se muestra la ventana que resulta de la actualización del firmware tras usar el botón *Update ICD Firmware*:



Por último en cuanto al ordenador destacar que los botones *Check COMM* y *Configure Port* nos permiten elegir el tipo de puerto por el que el ICD-U se conecta al ordenador y configurar dicha comunicación. En nuestro caso se ha usado el puerto USB del PC.

Además de estos aspectos relativos a la placa, ICD-U y ordenador; podemos configurar el programa mediante el botón *Advanced*, que nos abrirá otra ventana en la que se pueden modificar algunos parámetros.



3.2 SOLUCIÓN AL PROBLEMA

Pasamos ya a describir el código de programa albergado en el PIC y que será el encargado de gestionar el funcionamiento del sistema de acuerdo a las especificaciones deseadas. Se incluirá tanto el código de programa como una breve explicación del por qué del mismo en base a un planteamiento inicial del problema.

Recordemos en primer lugar cúal es la funcionalidad requerida a nuestro sistema. Buscábamos la sustitución de la manguera de 40 hilos por otro sistema compuesto por sólo 2 hilos (en realidad serán 3 para permitir la unión de las tierras de cada placa aunque el protocolo CAN sólo indique el empleo de los hilos CANL y CANH); de manera que esto fuera transparente al usuario de la máquina forestal. Por tanto nuestro sistema debe de procesar las señales recibidas a ambos lados (esto es en la cabina y en el cabezal) y enviarlas al lado opuesto. Habrá que analizar cada una de las señales recibidas a cada lado del sistema y estudiar la manera más óptima de tratarlas para codificarlas

y enviarlas por el bus CAN al otro extremo.

En adelante se analizará cada una de estas señales y se explicará el procedimiento seguido para conseguir leerla en un extremo y reproducirla exactamente en el otro. Asimismo se detallará la configuración de cada uno de los periféricos involucrados en el proyecto incluído el PIC.

Por último decir que será en el siguiente apartado donde se incluirá el código grabado en cada uno de los PIC's de las placas de cabeza y cabina.

Comencemos por tanto por describir la solución software a la que se ha llegado. Para ello definiremos en primer lugar algunos aspectos generales al proyecto y que afectan al funcionamiento global; para pasar más tarde a ver cada una de las señales con detenimiento.

3.2.1 ASPECTOS GLOBALES

El elemento fundamental en torno al cual gira el diseño software es el bus CAN. Dicho protocolo se basa en el paso de mensajes, los cuales escuchan todos los nodos conectados al bus. En nuestro caso sólo serán dos los nodos conectados de manera que existirán únicamente dos identificadores de mensajes. Estos identificadores servirán para evitar colisiones a la hora de escribir en el bus aunque ya en nuestro diseño se prevee que esto no ocurra.

Por tanto se defne un nodo que actuará como "director" o "maestro" de la comunicación y que en este caso será el situado en la cabeza de la máquina forestal. Dicho nodo enviará mensajes de 8 bytes a través del bus CAN con un período de 500µs. Dichos mensajes contendrán la información recibida por los sensores situados en el cabezal de la máquina; la cual llega mediante las señales de entrada de nuestro circuito de cabeza.

Así pues cada 500µs se envía por el bus la información recibida en la placa de la cabeza; la cual es recogida y procesada por la placa de la cabina.

Para conseguir esto se emplea una rutina de interrupción asociada al Timer1 de manera que cuando se desborda el contador de dicho Timer sabemos que el tiempo entre mensajes ha transcurrido y es necesario enviar otro con la nueva información.

Por otra parte en la placa de la cabina el funcionamiento es distinto. El envío de mensajes tiene por objeto hacer llegar a los actuadotes situados en la cabeza la acción que deben realizar de acuerdo a las órdenes dadas por el operario situado en cabina. Al igual que antes esta información viaja por el CAN y es recogida y procesada por la placa de la cabeza. La diferencia está en que estos envíos tienen lugar después de una recepción. De ahí que hayamos definido la placa de la cabeza como “maestra” de la comunicación dado que tras un envío de la placa de la cabeza tiene lugar un envío de la placa de la cabina pero no al reves.

Ahora bien, hay que tener en cuenta un aspecto importante al tratar el tipo de máquina que tenemos. Ese aspecto es la seguridad. Por defecto las salidas de la placa de la cabeza están a cero de manera que no se realice ninguna actividad por parte de los actuadores. Esto evita el que un tronco pueda precipitarse sobre la cabina por ejemplo. El estado de los actuadotes viene definido por los mensajes que en la cabeza se reciben de la cabina por lo que en caso de pérdida de la comunicación con la cabina; dicho estado de los actuadotes podría ser no deseado y ocasionar como decimos algún perjuicio en la seguridad del operario.

Se define entonces otro Timer (en concreto el Timer3) en la rutina de la placa de la cabeza de manera que si pasados 850μs no se recibe contestación por parte de la placa de la cabina las salidas se vuelven a poner a cero y el sistema nuevamente empieza a enviar un mensaje desde la placa de la cabeza a la de la cabina. En caso contrario el Timer se resetea y no ocurre esto.

Resumiendo decir que en la rutina de la placa de la cabeza el funcionamiento es el que describimos a continuación. Se envía un mensaje

cada 500µs con la información recogida por los sensores de la cabeza y se inicia el Timer de 500µs (Timer1) para el envío del siguiente. A continuación se está a la espera de recibir un mensaje de vuelta de la cabina. Si este mensaje llega se procesa y se pone a cero el Timer de seguridad (Timer3) con lo que el funcionamiento sigue así indefinidamente. En caso de no llegar, transcurridos los 850µs la rutina de seguridad saltará y las salidas y Timer1 se inicializan comenzando de nuevo el proceso descrito antes. Todo esto se consigue con la interrupción asociada a mensaje recibido por el módulo CAN.

Para la rutina de la placa de la cabina el funcionamiento es distinto. Continuamente se está a la espera de recibir un mensaje por el módulo CAN. Cuando la interrupción asociada a recepción por CAN avisa el mensaje recibido se procesa y se envía la información que hasta ese momento se ha recogido.

Vamos viendo ya por tanto que las interrupciones juegan un papel fundamental en el diseño, siendo su complicado control uno de los puntos delicados del proyecto. Además de las ya explicadas habrá alguna más específica de cada rutina (cabeza y cabina). Esto hará que sea inviable la opción de realizar debugging con el sistma completo dado que al incluir rutinas temporales se pierde totalmente el control del funcionamiento de nuestro programa.

Otro aspecto global que conviene indicar además de los ya mencionados es que emplearemos el identificador decimal 20 para los mensajes dirigidos a la placa de la cabina y el 10 para los destinados a la placa de la cabeza. Asimismo en ambas rutinas (cabeza y cabina) habrá que incluir el fichero de dispositivo correspondiente al PIC18F458.

Por último decir que al inicio del fichero asociado a la placa de la cabeza y cabina se incluyen aquellos ficheros auxiliares que se requieren como son el fichero del dispositivo (PIC18F458), el fichero de funciones auxiliares al programa principal y el fichero de extensión .h con las definiciones de variables y constantes así como la configuración del reloj del sistema, el watchdog o el

tipo de justificación de los registros del convertidor A/D. Asimismo se añade también el fichero que hace las funciones de driver del CAN con las definiciones y cuerpo de las funciones asociadas a este periférico. No obstante dedicaremos un apartado a este respecto.

Una vez comentados estos aspectos sólo queda especificar cómo se ha realizado el tratamiento particular de cada señal y el gobierno que se ha realizado del bus CAN.

3.2.2 CONFIGURACIÓN PIC

Respecto al PIC habrá de realizarse tanto la configuración de los periféricos (Timers, convertidores A/D, módulo CAN y módulo PWM, SPI, reloj del sistema, etc…); como la de las distintas interrupciones involucradas en el proceso.

Para esta labor resulta fundamental el empleo del compilador ya que facilita la escritura de los registros de cada uno de los periféricos de una manera muy sencilla e intuitiva sobre todo si se recurre al empleo del asistente de configuración.

El primer paso consiste en definir para cada pin del PIC si se trata de entrada o salida. Para ello se puede proceder a escribir los registros tris_a, tris_b, tris_c, tris_d y tris_e; de manera que un ’1’ implica que el pin en cuestión del puerto es entrada, y un ’0’ implica salida. Para el caso de la placa de la cabeza la configuración sería como sigue:

```
set_tris_a(0x0B);
set_tris_b(0x38);
set_tris_c(0x90);
set_tris_d(0x00);
set_tris_e(0x00);
```

Además de esta configuración de los puertos es necesario también configurar los periféricos del PIC. Esta tarea es bastante sencilla mediante el empleo del asistente, ya que gracias a un conjunto de ventanas en las que elegimos los parámetros deseados; se genera el código correspondiente. No

obstante en ocasiones puede resultar necesario configurar manualmente bit a bit los registros de algún periférico; pues el compilador no contempla todas las posibilidades. Nuevamente veremos a modo de ejemplo las líneas de código que permiten configurar los periféricos del PIC para la placa de la cabeza:

```
setup_adc_ports(RA0_RA1_RA3_ANALOG);
setup_adc(ADC_CLOCK_DIV_32);
setup_psp(PSP_DISABLED);
setup_spi(SPI_MASTER|SPI_XMIT_L_TO_H|SPI_CLK_DIV_4);
setup_wdt(WDT_OFF);
setup_timer_0(RTCC_OFF);
setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
setup_timer_2(T2_DIV_BY_16,255,1);
setup_timer_3(T3_INTERNAL|T3_DIV_BY_1);
setup_ccp1(CCP_PWM);
setup_ccp2(CCP_PWM);
setup_comparator(NC_NC_NC_NC);
```

Una vez configurados los periféricos habrá que inicializarlos antes de iniciar el bucle principal del programa. En este caso tendríamos:

```
can_init();
configuracion_expansor();
inicializacion_variables();
set_adc_channel(0);
set_timer1(0xF63C);
set_timer3(0x3CB0);
```

Por ultimo habrá también que habilitar las interrupciones necesarias para el correcto funcionamiento del sistema; algunas de las cúales ya han sido comentadas y otras serán comentadas a continuación. El código (generado en este caso por el asistente para el caso de la placa de la cabeza) será:

```
enable_interrupts(INT_TIMER1);
enable_interrupts(INT_TIMER3);
enable_interrupts(INT_CANRX0);
enable_interrupts(INT_CANRX1);
enable_interrupts(global);
```

Una vez habilitadas las interrupciones será necesario definir el cuerpo de la rutina asociada a cada una de estas interrupciones. Esto se realiza fuera del bucle principal de programa y su ejecución está asociada en cada caso a un

evento. Para nuestro sistema las interrupciones empleadas serán:

- TIMER1: se activa cuando se desborda el contador asociado a este Timer. Dicho contador se cargará de manera que transcurrido el tiempo deseado llegue a su valor máximo, con lo que en el siguiente ciclo de reloj se desbordará activando así la rutina correspondiente. Este Timer1 es empleado en la placa de la cabeza para la generación del período de 500µs entre cada envío de mensajes.
- TIMER0: funcionamiento análogo al anterior. Es empleada esta rutina para la generación en la placa de la cabina de las señales de encoger recibidas en la cabeza.
- TIMER3: funcionamiento análogo a los dos casos anteriores. En este caso la interrupción se emplea para generar los 850 µs de la rutina de seguridad.
- CAN_RX0, CAN_RX1: ambas albergan las mismas líneas de código ya que se ha activado la opción de doble buffer de manera que si uno de los buffer del módulo CAN se llena un nuevo mensaje sea guardado en el otro buffer evitando así la pérdida de información. Para la placa de la cabeza esta rutina permite comprobar que ha habido respuesta por parte de la cabina y se puede inicializar el Timer3 de la rutina de seguridad. En el caso de la placa de cabina la rutina de interrupción es la que permite detectar que se ha recibido un mensaje nuevo de la cabeza y ha de enviarse un nuevo mensaje por la cabina a la misma. A pesar de que sólo tenemos 2 nodos en el sistema y no existe ambigüedad en cuanto al destinatario del mensaje; en esta rutina se realiza en ambos casos la comprobación de que el identificador del mensaje es correcto de acuerdo con la placa que recibe dicho mensaje.
- RB: rutina que se ejecuta cuando en algunos de los pines altos (4 a 7) del puerto B se detecta un cambio. En nuestro caso se usa para la placa de la cabeza y permite detectar cualquiera de los flancos que se reciben de las señales de encoder. Además de incrementar la cuenta de pulsos en cada iteración, se determina el sentido de giro en cada caso.

El cuerpo completo de cada una de las rutinas puede analizarse con detalle en el listao completo del código que se encuentra en el próximo apartado. A modo de jemplo veamos la rutina asociada al Timer1:

```
#int_TIMER1
TIMER1_isr()
{
  #asm
```

```
  MOVWF W_TEMP
  MOVFF STATUS, STATUS_TEMP
  MOVFF BSR, BSR_TEMP
  #endasm

  set_timer1(0xF63C);

  dato[0]=ent_dig_exp;
  dato[1]=ent_dig_pic;
  dato[2]=can_analog_0_alto;
  dato[3]=can_analog_1_alto;
  dato[4]=can_analog_3_alto;
  dato[5]=can_analog_0_1_3;
  dato[6]=encoder;
  can_transmit(20,dato,8,3,0);

  contador=0;

  #asm
  MOVFF BSR_TEMP, BSR
  MOVF W_TEMP, W
  MOVFF STATUS_TEMP, STATUS
  #endasm
}
```

El cuerpo completo de la función o bucle principal de programa es aquel que se encuentra comprendido dentro de la función *main*. Por ejemplo:

```
void main()
{
  //Cuerpo de la función o bucle principal
}
```

3.2.3 CAN BUS

El protocolo CAN Bus constituye la base del diseño software. Su control eficiente será fundamental para la viabilidad del sistema. Por tanto la programación del CAN constituye la parte más importante de todo el código de programa.

Se basa fundamental en el empleo de tres funciones cuya utilidad es inicializar el bus, transmitir un mensaje y recibir un mensaje. A su vez estas

funciones necesitan de otras para el establecimiento de las máscaras y filtros de mensaje, configuración de la velocidad del bus o el modo de operación, conversión de identificadores de mensajes, etc.

El conjunto de todas estas funciones puede verse en el fichero *can-18f458_2.c* y que se encuentra en el cd adjunto al proyecto. No las listaremos aquí debido a su extensión. Además el fichero mencionado contiene una serie de comentarios que hacen fácil la comprensión del código de programa.

3.2.4 E/S DIGITALES

Dado que el tráfico de información proveniente de E/S digitales tiene lugar tanto en el sentido cabina-cabeza como cabeza-cabina se detallará sólo el funcionamiento en el segundo caso y dicha explicación será extensiva al otro sin más que invertir los papeles de cada placa.

Para el tratamiento de dichos datos es fundamental la correcta programación del expansor MAX6957; pues será el encargado de serializar los datos recibidos para enviarlos mediante SPI al PIC que posteriormente los empaquetará en cada mensaje CAN que se envíe a la cabina. Del mismo modo realizará el proceso inverso para aquellos datos que llegan mediante el bus CAN al PIC y han de ser enviados a los actuadores del cabezal de la máquina.

El fichero *funciones.c* contiene la función encargada de la configuración del expansor de E/S de la placa de la cabeza. Conviene recordar que para esta placa se tenían 15 salidas digitales por tan sólo 6 entradas. Por tanto han de configurarse los puertos del expansor conectados a las salidas como tales y lo mismo para las entradas. Previamente habremos configurado el expnasor para su modo de funcionamiento normal. La función encargada de estas tareas se denomina *configuracion_expansor* y su definición es la siguiente:

```
void configuracion_expansor(void){

  output_low(CS);
  spi_write(0x04);              //Modo normal de operación
  spi_write(0x01);
```

```
    output_high(CS);

    output_low(CS);
    spi_write(0x0B);              //Pines 12-15 como salidas
    spi_write(0x55);
    output_high(CS);

    output_low(CS);
    spi_write(0x0C);              //Pines 16-19 como salidas
    spi_write(0x55);
    output_high(CS);

    output_low(CS);
    spi_write(0x0D);              //Pines 20-23 como salidas
    spi_write(0x55);
    output_high(CS);

    output_low(CS);
    spi_write(0x0E);      //Pines 24-26 como salidas y pin 27 como entrada
    spi_write(0x95);
    output_high(CS);

    output_low(CS);
    spi_write(0x0F);              //Pines 28-31 como entradas
    spi_write(0xAA);
    output_high(CS);

}
```

Se puede observar que se han configurado sólamente 5 de las 6 entradas digitales. La razón es que una de ellas está unida directamente a una entrada del PIC por lo que su procesado no requiere el expansor.

Una vez configurado el expansor estará listo para los procesos de lectura y escritura, que ya fueron comentados en la descripción del MAX6957 que se dio en el apartado de hardware. En concreto para este caso la lectura y escritura se realiza mediante las siguientes sentencias:

```
    //Lectura de entradas digitales

    output_low(CS);
    spi_write(0xDB);
    spi_write(0x00);
    output_high(CS);
    output_low(CS);
    spi_write(0xDB);
```

```
spi_write(0x00);
output_high(CS);
ent_dig_exp=spi_read();
ent_dig_pic=puerto_c.pinc7;

//Escritura de salidas digitales

output_low(CS);
spi_write(0x4C);
spi_write(sal_dig_1);
output_high(CS);

output_low(CS);
spi_write(0x53);
spi_write(sal_dig_2);
output_high(CS);
```

Para el transporte de la información por el bus CAN se utilizan una serie de variables las cuales se almacenan posteriormente en cada uno de los bytes del mensaje CAN y que se detallarán en el apartado de programación del CAN. No obstante podemos adelantar que dichas variables son:

- ent_dig_exp: variable empleada para almacenar la información de las entradas digitales provenientes del expansor
- ent_dig_pic: variable empleada para almacenar la información de la entrada digital conectada directamente al PIC
- sal_dig_1, sal_dig_2: variables empleadas para almacenar los valores que se aplicarán a cada una de las salidas digitales

3.2.5 ENTRADAS ANALÓGICAS

Las señales analógicas serán leídas en la placa del procesador forestal; la cual las envía gracias al CAN-Bus hacia la placa de la cabina para ser sacadas por esta hacia el autómata. Así pues se verán involucrados dos procesos: conversión A/D y conversión D/A.

Para la conversión A/D disponemos de un convertidor interno al PIC y cuya configuración e inicilización ya fue comentada anteriormente. En cambio para la conversión D/A se hace preciso el empleo de un periférico externo; en este caso el MAX5742.

La captura de las señales analógicas se realiza mediante 16 bits, de los cuales sólo 10 son necesarios pues el convertidor D/A tiene esa precisión. Así pues, nosotros enviaremos para cada canal analógico de entrada una variable con los ocho bits menos significativos; dejando otra variable compartida por los tres canales y de la cual sólo ocuparan dos bits cada uno (los dos más significativos).

Por tanto el proceso consiste en leer los 16 bits dejando en una variable los ocho menos significativos, y en la variable compartida los dos más significativos. Por ejemplo para el canal 0 sería como sigue:

```
canal_0=read_adc();
can_analog_0_alto=make8(canal_0,1);
can_analog_0_bajo=make8(canal_0,0);
can_analog_0_aux=can_analog_0_bajo>>6;
set_adc_channel(1);
```

Vemos como también se deja preparado el convertidor para la lectura del próximo canal. Para cada entrada analógica tenemos una variable auxiliar de forma que haciendo una función OR de las tres conseguimos transportar en un solo byte los dos bits más significativos de las tres señales. Así se consigue ahorrar bits en cada mensaje CAN.

En el lado opuesto tenemos el DAC el cual se controla mediante la escritura de palabras de doce bits. De estos doce bits sólo diez llevarán información de la señal a reproducir por lo que en primer lugar habrá que hacer operaciones de desplazamiento e inversión de los bits de cada canal para colocarlos de manera adecuada en la palabra de doce bits que se cargará mediante SPI en el convertidor.

Ha de tenerse en cuenta sila justificación de los bits se hace a la izquierda o a la derecha. Una vez preparada la palabra se carga mediante SPI de forma similar a como se hizo para el expansor de E/S y tendremos nuestra señal analógica reproducida a la salida del DAC. Por ejemplo para el canal 0

sería como sigue:

```
can_dac_a_alto=in_data[2]; // Recibo byte alto: aaaa bbbb
can_dac_a_bajo=in_data[5] & 0x03; // Recibo byte bajo: 0000 00cc
can_dac_a_aux=swap(can_dac_a_alto); // Intercambio bits altos por
                                                    bajos: bbbb aaaa
can_a_dac_temp=can_dac_a_aux & 0xF0; // Cojo los 4 primeros bits;
                                                      bbbb 0000
can_dac_a_bajo=can_dac_a_bajo<<2 ; // Pongo 2 ultimos bits en
                                                    posición: 0000 cc00
DACAH=can_dac_a_aux & 0x0F; // Dato alto listo para cargar en el
                                                  AC: 0000 aaaa
DACAL=can_dac_a_bajo | can_a_dac_temp; // Dato bajo listo para
                                                    cargar en el DAC: bbbb cc00
output_low(CS_DAC);
spi_write(DACAH); // Carga dato alto en DAC A
spi_write(DACAL); // Carga dato bajo en DAC A
output_high(CS_DAC);
```

Del mismo modo se procedería para el resto de canales.

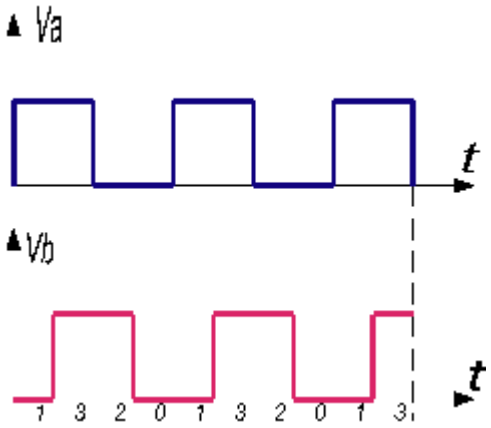
3.2.6 ENTRADAS ENCODER

Se tienen dos señales digitales cuadradas con un desfase entre ellas de 90º. De dichas señales habrá de descifrarse el sentido de giro de los rodillos que manejan los árboles. Asimismo nos interesará también llevar la cuenta de pulsos recibidos durante el intervalo de tiempo entre un mensaje CAN y otro. Así pues, se enviará mediante el bus CAN un byte destinado a las señales de encoder en el que los tres bits menos significativos codifican la cuenta de pulsos recibidos; mientras que los cinco más significativos contendrán el sentido de giro de cada pulso.

Considerando que las señales de encoder presentan una frecuencia máxima de 2kHz, en un intervalo de 500μs se tienen en el peor de los casos cinco pulsos luego el espacio destinado a la información de estas señales es suficiente. En cuanto al sentido decir que un ’1’ significa sentido horario y un ’0’ antihorario.

Por tanto para concretar llamaremos a una de las señales de encoder A, y a la otra B de manera que si es A la que se detecta antes que B el sentido es horario y viceversa. Una representación de este tipo de señales sería la

siguiente:



En la figura vemos marcados los distintos estados en que pueden encontrarse el par de señales que tratamos. Según la secuencia entre uno y otro se estará produciendo un giro en un sentido u otro. Tendremos por tanto que contemplar los posibles cambios de giro en el encoder. Para ello disponemos de la función *calcula_sentido* que forma parte del fichero *funciones_cabeza.c* que se lista en el siguiente apartado. Debido a su extensión nos remitimos a dicho apartado para consultar las sentencias exactas de dicha función en lugar de mostrarla aquí.

Simplemente comentar que con cada interrupción del puerto B se ejecuta la función *calcula_sentido* que guardará el sentido del último pulso recibido en una variable auxiliar; y se incrementará el contador de pulsos. Según el valor de dicho contador se escribe en el byte del mensaje CAN correspondiente al encoder, el nuevo valor del contador y el sentido del pulso recibido. Esto puede verse en la función *componer_byte_CAN*.

Una vez se recibe esta información en la placa de la cabina se procede a la generación de dos señales digitales. La primera de ellas será una señal cuadrada con tantas transiciones o pulsos como indique la cuenta enviada desde la cabeza. La segunda se corresponde con el sentido de giro de cada uno de los pulsos que se generan con la primera de las señales.

Cabe preguntarse el por qué del empleo de estas dos señales en lugar

de las originales recibidas en la cabeza. El motivo es simplificar el código ya que el autómata admite ambos formatos de información. Así pues resulta más cómodo a la hora de programar optar por la solución comentada.

3.2.7 SALIDAS PWM

En este caso recibimos por el lado de la cabina dos señales analógicas y una vez enviadas mediante el CAN son presentadas en el otro extremo como señales de salida PWM.

Este tipo de modulación es muy empleada en el control de motores y para el caso que nos ocupa resultó una manera muy óptima de controlar las electroválvulas proporcionales.

El modo de hacerlo es muy simple ya que en este caso ambos módulos (convertidor A/D y PWM) son internos al PIC. La configuración e inicialización se supone ya realizada como se comentó con anterioridad por lo que sólo habrá que explicar el tipo de tratamiento que se realiza sobre cada señal.

En cuanto al lado de la cabina poco tenemos que añadir a lo que ya se dijo anteriormente respecto a las entradas analógicas de la cabeza del robot. El proceso es exactamente el mismo y sólo se diferencia en el otro extremo pues en lugar de reconstruir la señal analógica se sacrá modulada mediante PWM.

Para la modulación PWM se recurre al módulo de captura / comparación / modulación del PIC. Para su manejo sólo habrá que establecer el ciclo de trabajo teniendo en cuenta que los optoacopladores invierten la señal. Así pues al máximo restaremos el valor deseado; de manera que al pasar por los optoacopladores tengamos la señal que buscamos. Para establecer el ciclo de trabajo habrá que realizar una labor de justificación e inversión de los bits recibidos mediante el CAN para adaptarlos al formato que requiere el módulo PWM. Para ambos canales el código que realiza la escritura de las señales sería:

```
pwm1_alto=in_data[2];
pwm1_bajo=in_data[4] & 0x03;
pwm1_bajo=pwm1_bajo<<6;
pwm1_aux=make16(pwm1_alto,pwm1_bajo);
pwm1=pwm1_aux>>6;
pwm2_alto=in_data[3];
pwm2_bajo=in_data[4] & 0x0C;
pwm2_bajo=pwm2_bajo<<4;
pwm2_aux=make16(pwm2_alto,pwm2_bajo);
pwm2=pwm2_aux>>6;
set_pwm1_duty(1023-pwm1);
set_pwm2_duty(1023-pwm2);
```

3.3 CÓDIGO

A continuación listaremos sin más los distintos códigos implicados en el funcionamiento de nuestro sistema. Se trata de los códigos principales de las placas de cabina y cabeza; así como los ficheros de configuración de periféricos y funciones auxiliares.

CÓDIGO CABEZA

FICHERO CABEZA.C

```
#include "C:\Archivos de programa\PICC\Examples\e_s digitales.h"
#include "C:\Archivos de programa\PICC\Examples\funciones.c"

#int_TIMER1
TIMER1_isr()
{
  #asm
  MOVWF W_TEMP
  MOVFF STATUS, STATUS_TEMP
  MOVFF BSR, BSR_TEMP
  #endasm

  set_timer1(0xF63C);

  dato[0]=ent_dig_exp;
  dato[1]=ent_dig_pic;
  dato[2]=can_analog_0_alto;
  dato[3]=can_analog_1_alto;
  dato[4]=can_analog_3_alto;
  dato[5]=can_analog_0_1_3;
  dato[6]=encoder;
  can_transmit(20,dato,8,3,0);
```

```
   contador=0;

   #asm
   MOVFF BSR_TEMP, BSR
   MOVF W_TEMP, W
   MOVFF STATUS_TEMP, STATUS
   #endasm
}

#int_TIMER3
TIMER3_isr()
{

   inicializacion_variables();
   can_init();
   set_pwm1_duty(0xFF);
   set_pwm2_duty(0xFF);
   set_timer1(0xF63C);
   set_timer3(0x3CB0);

}

#int_CANRX0
CANRX0_isr()
{
   #asm
   MOVWF W_TEMP
   MOVFF STATUS, STATUS_TEMP
   MOVFF BSR, BSR_TEMP
   #endasm

   can_receive(rx_id,&in_data[0],rx_len);
   if(rx_id==10)
   {

   sal_dig_1=in_data[0];
   sal_dig_2=in_data[1];

   }
   set_timer3(0x3CB0);

   #asm
   MOVFF BSR_TEMP, BSR
   MOVF W_TEMP, W
   MOVFF STATUS_TEMP, STATUS
   #endasm
}

#int_CANRX1
CANRX1_isr()
```

```
{
  #asm
  MOVWF W_TEMP
  MOVFF STATUS, STATUS_TEMP
  MOVFF BSR, BSR_TEMP
  #endasm

  can_receive(rx_id,&in_data[0],rx_len);
  if(rx_id==10)
  {

  sal_dig_1=in_data[0];
  sal_dig_2=in_data[1];

  }
  set_timer3(0x3CB0);

  #asm
  MOVFF BSR_TEMP, BSR
  MOVF W_TEMP, W
  MOVFF STATUS_TEMP, STATUS
  #endasm
}

#int_RB
RB_isr()
{

  int auxiliar;
  set_tris_b(0x38);
  auxiliar=input_b();

  contador++;

  calcula_sentido();
  componer_byte_CAN();

}

void main()
{

  //Configuración de los puertos del PIC

  set_tris_a(0x0B);
  set_tris_b(0x38);
  set_tris_c(0x90);
  set_tris_d(0x00);
  set_tris_e(0x00);
```

```
//Configuración de los periféricos del PIC

setup_adc_ports(RA0_RA1_RA3_ANALOG);
setup_adc(ADC_CLOCK_DIV_32);
setup_psp(PSP_DISABLED);
setup_spi(SPI_MASTER|SPI_XMIT_L_TO_H|SPI_CLK_DIV_4);
setup_wdt(WDT_OFF);
setup_timer_0(RTCC_OFF);
setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
setup_timer_2(T2_DIV_BY_16,255,1);
setup_timer_3(T3_INTERNAL|T3_DIV_BY_1);
setup_ccp1(CCP_PWM);
setup_ccp2(CCP_PWM);
setup_comparator(NC_NC_NC_NC);

//Inicialización de periféricos y variables

can_init();
configuracion_expansor();
inicializacion_variables();
set_adc_channel(0);
set_timer1(0xF63C);
set_timer3(0x3CB0);

//Habilitación de interrupciones

enable_interrupts(INT_TIMER1);
enable_interrupts(INT_TIMER3);
enable_interrupts(INT_CANRX0);
enable_interrupts(INT_CANRX1);
enable_interrupts(global);

//Programa principal

while(TRUE){

//Lectura de entradas digitales

output_low(CS);
spi_write(0xDB);
spi_write(0x00);
output_high(CS);
output_low(CS);
spi_write(0xDB);
spi_write(0x00);
output_high(CS);
ent_dig_exp=spi_read();
ent_dig_pic=puerto_c.pinc7;

//Escritura de salidas digitales
```

```
output_low(CS);
spi_write(0x4C);
spi_write(sal_dig_1);
output_high(CS);

output_low(CS);
spi_write(0x53);
spi_write(sal_dig_2);
output_high(CS);

//Lectura de los canales analógicos

canal_0=read_adc();
can_analog_0_alto=make8(canal_0,1);
can_analog_0_bajo=make8(canal_0,0);
can_analog_0_aux=can_analog_0_bajo>>6;
set_adc_channel(1);
delay_us(10);
canal_1=read_adc();
can_analog_1_alto=make8(canal_1,1);
can_analog_1_bajo=make8(canal_1,0);
can_analog_1_aux=can_analog_1_bajo>>4;
set_adc_channel(3);
delay_us(10);
canal_3=read_adc();
can_analog_3_alto=make8(canal_3,1);
can_analog_3_bajo=make8(canal_3,0);
can_analog_3_aux=can_analog_3_bajo>>2;
set_adc_channel(0);
can_analog_0_1_3=can_analog_0_aux | can_analog_1_aux |
 can_analog_3_aux;

//Escritura de las señales PWM

pwm1_alto=in_data[2];
pwm1_bajo=in_data[4] & 0x03;
pwm1_bajo=pwm1_bajo<<6;
pwm1_aux=make16(pwm1_alto,pwm1_bajo);
pwm1=pwm1_aux>>6;
pwm2_alto=in_data[3];
pwm2_bajo=in_data[4] & 0x0C;
pwm2_bajo=pwm2_bajo<<4;
pwm2_aux=make16(pwm2_alto,pwm2_bajo);
pwm2=pwm2_aux>>6;
set_pwm1_duty(1023-pwm1);
set_pwm2_duty(1023-pwm2);

}
```

```
}
```

FICHERO CABEZA.H

```
#include <18F458.h>
#device adc=10
#use delay(clock=20000000)
#fuses NOWDT,HS, NOPROTECT,NOBROWNOUT,NOLVP,WRT
#use fast_io(A)
#use fast_io(B)
#use fast_io(C)
#use fast_io(D)
#use fast_io(E)
#bit ADFM = 0xFC1.7
#include <can-18f458_2.c>
#byte PORTA = 0xF80
#byte PORTB = 0xF81
#byte PORTC = 0xF82
#byte PORTD = 0xF83
#byte PORTE = 0xF84

int in_data[8];
int rx_len,sync[1],syncrec[1];
int i,j,a,b,c,d,z,buf,out;
int32 rx_id;
byte dato[8];

long pwm1,pwm2,pwm1_aux,pwm2_aux;
int pwm1_alto,pwm1_bajo,pwm2_alto,pwm2_bajo;

int ent_dig_exp,ent_dig_pic,sal_dig_1,sal_dig_2;

long canal_0,canal_1,canal_3;
int can_analog_0_alto,can_analog_1_alto,can_analog_3_alto;
int can_analog_0_1_3;
int can_analog_0_bajo,can_analog_1_bajo,can_analog_3_bajo;
int can_analog_0_aux,can_analog_1_aux,can_analog_3_aux;

int contador,etiqueta;
boolean fsa;
boolean fsb;
boolean fba;
boolean fbb;
boolean sentido;
boolean error;

struct{

   int cuenta:3;
   int1 sentido1;
```

```
   int1 sentido2;
   int1 sentido3;
   int1 sentido4;
   int1 sentido5;
}encoder;

struct{

   int1 pinc0;
   int1 pinc1;
   int1 pinc2;
   int1 pinc3;
   int1 pinc4;
   int1 pinc5;
   int1 pinc6;
   int1 pinc7;
}puerto_c;
#byte puerto_c = 0xF82

struct{

   int1 pind0;
   int1 pind1;
   int1 pind2;
   int1 pind3;
   int1 pind4;
   int1 pind5;
   int1 pind6;
   int1 pind7;
}puerto_d;
#byte puerto_d = 0xF83

int W_TEMP;
#locate W_TEMP=0x100
int STATUS_TEMP;
#locate STATUS_TEMP=0x200
int BSR_TEMP;
#locate BSR_TEMP=0x300
#byte STATUS = 0xFD8
#byte BSR = 0xFE0

#define RELOJ PIN_C3
#define CS PIN_C6
#define DIN PIN_C5

#define GPIN1 PIN_B4
#define GPIN2 PIN_B5

boolean sh,sah;
int temp;
```

FICHERO FUNCIONES_CABEZA.C

```
void configuracion_expansor(void){

  output_low(CS);
  spi_write(0x04);
  spi_write(0x01);
  output_high(CS);

  output_low(CS);
  spi_write(0x0B);
  spi_write(0x55);
  output_high(CS);

  output_low(CS);
  spi_write(0x0C);
  spi_write(0x55);
  output_high(CS);

  output_low(CS);
  spi_write(0x0D);
  spi_write(0x55);
  output_high(CS);

  output_low(CS);
  spi_write(0x0E);
  spi_write(0x95);
  output_high(CS);

  output_low(CS);
  spi_write(0x0F);
  spi_write(0xAA);
  output_high(CS);

}

void inicializacion_variables (void){

  ADFM=0;

  for(i=0;i<8;i++)
  {
  in_data[i]=0x00;
  dato[i]=0x00;
  }

  pwm1=0x00;
  pwm2=0x00;
```

```
   canal_0=0x00;
   canal_1=0x00;
   canal_3=0x00;
   can_analog_0_alto=0x00;
   can_analog_0_bajo=0x00;
   can_analog_0_aux=0x00;
   can_analog_1_alto=0x00;
   can_analog_1_bajo=0x00;
   can_analog_1_aux=0x00;
   can_analog_3_alto=0x00;
   can_analog_3_bajo=0x00;
   can_analog_3_aux=0x00;
   can_analog_0_1_3=0x00;

   ent_dig_exp=0x00;
   ent_dig_pic=0x00;

   sal_dig_1=0x00;
   sal_dig_2=0x00;

   encoder.cuenta=0;
   encoder.sentido1=0;
   encoder.sentido2=0;
   encoder.sentido3=0;
   encoder.sentido4=0;
   encoder.sentido5=0;

   fsa=FALSE;
   fsb=FALSE;
   fba=FALSE;
   fbb=FALSE;
   sentido=TRUE;
   error=FALSE;

   contador=0;

}

void calcula_sentido()
{

if((fsa==FALSE)&&(fsb==FALSE)&&(fba==FALSE)&&(fbb==FALSE))
{
   if(((input(GPIN1))==1) && ((input(GPIN2))==0))
   {
      fsa=TRUE;
      sentido=TRUE;
      error=FALSE;
   }
   else{
```

```
      if(((input(GPIN1))==0) && ((input(GPIN2))==1))
      {
        fsb=TRUE;
        sentido=FALSE;
        error=FALSE;
      }
      else{
        fsa=FALSE;
        fsb=FALSE;
        fba=FALSE;
        fbb=FALSE;
        error=TRUE;
      }
    }
    goto fin;
  }

  if((fsa==TRUE) && ((fsb==FALSE)&&(fba==FALSE)&&(fbb==FALSE)))
  {
    if(((input(GPIN1))==1) && ((input(GPIN2))==1))
    {
      fsb=TRUE;
      error=FALSE;
    }
    else{

      if(((input(GPIN1))==0) && ((input(GPIN2))==0))
      {
        fsa=FALSE;
        fsb=FALSE;
        fba=FALSE;
        fbb=FALSE;
        sentido=FALSE;
        error=FALSE;
      }
      else{
        fsa=FALSE;
        fsb=FALSE;
        fba=FALSE;
        fbb=FALSE;
        error=TRUE;
      }
    }
    goto fin;
  }

  if((fsb==TRUE) && ((fsa==FALSE)&&(fba==FALSE)&&(fbb==FALSE)))
  {
    if(((input(GPIN1))==1) && ((input(GPIN2))==1))
```

```
    {
      fsa=TRUE;
      error=FALSE;
    }
    else{

      if(((input(GPIN1))==0) && ((input(GPIN2))==0))
      {
        fsa=FALSE;
        fsb=FALSE;
        fba=FALSE;
        fbb=FALSE;
        sentido=TRUE;
        error=FALSE;
      }
      else{
        fsa=FALSE;
        fsb=FALSE;
        fba=FALSE;
        fbb=FALSE;
        error=TRUE;
      }
    }
    goto fin;
}

if(((fsa==TRUE)&&(fsb==TRUE)) && ((fba==FALSE)&(fbb==FALSE)))
{
    if(((input(GPIN1))==0) && ((input(GPIN2))==1) && (sentido==TRUE))
    {
      fba=TRUE;
      error=FALSE;
    }
    else{

      if(((input(GPIN1))==1)&&((input(GPIN2))==0) && (sentido==FALSE))
      {
        fbb=TRUE;
        error=FALSE;
      }
      else{

        if(((input(GPIN1))==1)&&((input(GPIN2))==0) && (sentido==TRUE))
        {
          fbb=TRUE;
          sentido=FALSE;
          error=FALSE;
        }
        else{
```

```
        if(((input(GPIN1))==0)&&((input(GPIN2))==1)&&(sentido==FALSE))
          {
            fba=TRUE;
            sentido=TRUE;
            error=FALSE;
          }
          else{
            fsa=FALSE;
            fsb=FALSE;
            fba=FALSE;
            fbb=FALSE;
            error=TRUE;
          }
        }
      }
    }
    goto fin;
}

if(((fsa==TRUE)&&(fsb==TRUE)&&(fba==TRUE)) && ((fbb==FALSE)))
{
  if(((input(GPIN1))==0) && ((input(GPIN2))==0))
  {
    fsa=FALSE;
    fsb=FALSE;
    fba=FALSE;
    fbb=FALSE;
    error=FALSE;
  }
  else{

    if(((input(GPIN1))==1) && ((input(GPIN2))==1))
    {
      if(sentido==TRUE)
      {
        sentido=FALSE;
        fba=FALSE;
        error=FALSE;
      }
      else
      {
        sentido=TRUE;
        fba=FALSE;
        error=FALSE;
      }
    }
    else{
      fsa=FALSE;
      fsb=FALSE;
      fba=FALSE;
```

```
        fbb=FALSE;
        error=TRUE;
      }
    }
    goto fin;
}

if(((fsa==TRUE)&&(fsb==TRUE)&&(fbb==TRUE)) && ((fba==FALSE)))
{
  if(((input(GPIN1))==0) && ((input(GPIN2))==0))
  {
    fsa=FALSE;
    fsb=FALSE;
    fba=FALSE;
    fbb=FALSE;
    error=FALSE;
  }
  else{
    if(((input(GPIN1))==1) && ((input(GPIN2))==1))
    {
      if(sentido==TRUE)
      {
        fbb=FALSE;
        sentido=FALSE;
        error=FALSE;
      }
      else
      {
        fbb=FALSE;
        sentido=TRUE;
        error=FALSE;
      }
    }
    else{
      fsa=FALSE;
      fsb=FALSE;
      fba=FALSE;
      fbb=FALSE;
      error=TRUE;
    }
  }
}

fin: etiqueta=1;

}

void componer_byte_CAN()
{
```

```
   switch(contador){

   case 1: encoder.sentido1=sentido;
     break;
   case 2: encoder.sentido2=sentido;
     break;
   case 3: encoder.sentido3=sentido;
     break;
   case 4: encoder.sentido4=sentido;
     break;
   case 5: encoder.sentido5=sentido;
     break;
   default: encoder.sentido5=sentido;
     break;
   }

   encoder.cuenta=contador;

}
```

CÓDIGO CABINA

FICHERO CABINA.C

```
#include "C:\Archivos de programa\PICC\Examples\rx_e_s digitales.h"
#include "C:\Archivos de programa\PICC\Examples\funciones_cabina.c"

#int_RTCC
RTCC_isr()
{
  #asm
  MOVWF W_TEMP
  MOVFF STATUS, STATUS_TEMP
  MOVFF BSR, BSR_TEMP
  #endasm

  switch(MENSAJE.cuenta){

    case 1: un_pulso();
          break;
    case 2: dos_pulsos();
          break;
    case 3: tres_pulsos();
          break;
    case 4: cuatro_pulsos();
          break;
    case 5: cinco_pulsos();
          break;
    default: { if (puerto_c.pinc1==1) puerto_c.pinc1=1;
            else puerto_c.pinc1=0;
```

```
            }
    }
    #asm
    MOVFF BSR_TEMP, BSR // Restaurar BSR
    MOVF W_TEMP, W // Restaurar WREG
    MOVFF STATUS_TEMP, STATUS // Restaurar STATUS
    #endasm
}

#int_CANRX0
CANRX0_isr()
{
    #asm
    MOVWF W_TEMP
    MOVFF STATUS, STATUS_TEMP
    MOVFF BSR, BSR_TEMP
    #endasm

    can_receive(rx_id,&in_data[0],rx_len);
    if(rx_id==20)
    {

    sal_dig_exp=in_data[0];
    sal_dig_pic=in_data[1];
    MENSAJE=in_data[6];

    }

    dato[0]=ent_dig_1;
    dato[1]=ent_dig_2;
    dato[2]=can_analog_0_alto;
    dato[3]=can_analog_1_alto;
    dato[4]=can_analog_0_1;

    can_transmit(10,dato,8,3,0);

    switch(MENSAJE.cuenta){

      case 1: { contador=11;
                un_pulso();
                break;}
      case 2: { contador=21;
                dos_pulsos();
                break;}
      case 3: { contador=31;
                tres_pulsos();
                break;}
      case 4: { contador=41;
                cuatro_pulsos();
                break;}
```

```
      case 5: { contador=51;
                cinco_pulsos();
                break;}
      default: set_timer0(0);
               break;
    }

    #asm
    MOVFF BSR_TEMP, BSR
    MOVF W_TEMP, W
    MOVFF STATUS_TEMP, STATUS
    #endasm
}

#int_CANRX1
CANRX1_isr()
{
    #asm
    MOVWF W_TEMP
    MOVFF STATUS, STATUS_TEMP
    MOVFF BSR, BSR_TEMP
    #endasm

    can_receive(rx_id,&in_data[0],rx_len);
    if(rx_id==20)
    {

    sal_dig_exp=in_data[0];
    sal_dig_pic=in_data[1];
    MENSAJE=in_data[6];

    }

    dato[0]=ent_dig_1;
    dato[1]=ent_dig_2;
    dato[2]=can_analog_0_alto;
    dato[3]=can_analog_1_alto;
    dato[4]=can_analog_0_1;

    can_transmit(10,dato,8,3,0);

    switch(MENSAJE.cuenta){

      case 1: { contador=11;
                un_pulso();
                break;}
      case 2: { contador=21;
                dos_pulsos();
                break;}
      case 3: { contador=31;
```

```
                tres_pulsos();
                break;}
        case 4: { contador=41;
                cuatro_pulsos();
                break;}
        case 5: { contador=51;
                cinco_pulsos();
                break;}
        default: set_timer0(0);
                break;
    }

    #asm
    MOVFF BSR_TEMP, BSR
    MOVF W_TEMP, W
    MOVFF STATUS_TEMP, STATUS
    #endasm
}

void main()
{

    //Configuración puertos del PIC

    set_tris_a(0x0B);
    set_tris_b(0x08);
    set_tris_c(0x90);
    set_tris_d(0x00);
    set_tris_e(0x00);

    //Configuración periféricos del PIC

    setup_adc_ports(RA0_RA1_ANALOG_RA3_REF);
    //setup_adc_ports(RA0_RA1_RA3_ANALOG);
    setup_adc(ADC_CLOCK_DIV_32);
    //ADCON0.ADCS1_0=0b10;
    //ADCON1.ADCS2=0b0;
    //ADCON1.PCFG=0b0101;
    setup_psp(PSP_DISABLED);
    setup_spi(SPI_MASTER|SPI_XMIT_L_TO_H|SPI_CLK_DIV_4);
    setup_wdt(WDT_OFF);
    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_16|RTCC_8_bit);
    setup_timer_1(T1_DISABLED);
    setup_timer_2(T2_DISABLED,0,1);
    setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
    setup_comparator(NC_NC_NC_NC);

    //Inicializaciones de variables y periféricos

    can_init();
```

```
configuracion_expansor();
configuracion_dac();
set_adc_channel(0);
set_timer0(0);

//Habilitación de interrupciones

enable_interrupts(INT_CANRX0);
enable_interrupts(INT_CANRX1);
enable_interrupts(INT_RTCC);
enable_interrupts(global);

//Programa principal

while(TRUE){

//Lectura de entradas digitales

output_low(CS_EXP);
spi_write(0xCC);
spi_write(0x00);
output_high(CS_EXP);
output_low(CS_EXP);
spi_write(0xCC);
spi_write(0x00);
output_high(CS_EXP);
ent_dig_1=spi_read();

output_low(CS_EXP);
spi_write(0xD3);
spi_write(0x00);
output_high(CS_EXP);
output_low(CS_EXP);
spi_write(0xD3);
spi_write(0x00);
output_high(CS_EXP);
ent_dig_2=spi_read();

//Escritura de salidas digitales

output_low(CS_EXP);
spi_write(0x5B);
spi_write(sal_dig_exp);
output_high(CS_EXP);

puerto_c.pinc0=sal_dig_pic;

//Escritura de datos del DAC

can_dac_a_alto=in_data[2]; // Recibo byte alto: aaaa bbbb
```

```
can_dac_a_bajo=in_data[5] & 0x03; // Recibo byte bajo: 0000 00cc
can_dac_a_aux=swap(can_dac_a_alto); // Intercambio bits altos por
                                        bajos: bbbb aaaa
can_a_dac_temp=can_dac_a_aux & 0xF0; // Cojo los 4 primeros bits;
                                        bbbb 0000
can_dac_a_bajo=can_dac_a_bajo<<2 ; // Pongo 2 ultimos bits en
                                        posición: 0000 cc00
DACAH=can_dac_a_aux & 0x0F; // Dato alto listo para cargar en el
                                        AC: 0000 aaaa
DACAL=can_dac_a_bajo | can_a_dac_temp; // Dato bajo listo para
                                        cargar en el DAC: bbbb cc00

can_dac_b_alto=in_data[3]; // Recibo byte alto: aaaa bbbb
can_dac_b_bajo=in_data[5] & 0x0C; // Recibo byte bajo: 0000 cc00
can_dac_b_aux=swap(can_dac_b_alto); // Intercambio bits altos por
                                        bajos: bbbb aaaa
can_b_dac_temp=can_dac_b_aux & 0xF0; // Cojo los 4 primeros bits;
                                        bbbb 0000
DACBH=can_dac_b_aux & 0x0F; // Dato alto listo para cargar en el
                                        DAC: 0000 aaaa
DACBL=can_dac_b_bajo | can_b_dac_temp; // Dato bajo listo para
                                        cargar en el DAC: bbbb cc00

can_dac_c_alto=in_data[4]; // Recibo byte alto: aaaa bbbb
can_dac_c_bajo=in_data[5] & 0x30; // Recibo byte bajo: 00cc 0000
can_dac_c_aux=swap(can_dac_c_alto); // Intercambio bits altos por
                                        bajos: bbbb aaaa
can_c_dac_temp=can_dac_c_aux & 0xF0; // Cojo los 4 primeros bits;
                                        bbbb 0000
can_dac_c_bajo=can_dac_c_bajo>>2 ; // Pongo 2 ultimos bits en
                                        posición: 0000 cc00
DACCH=can_dac_c_aux & 0x0F; // Dato alto listo para cargar en el
                                        DAC: 0000 aaaa
DACCL=can_dac_c_bajo | can_b_dac_temp; // Dato bajo listo para
                                        cargar en el DAC: bbbb cc00

bit_set(DACBH,4);
bit_set(DACCH,5);
output_low(CS_DAC);
spi_write(DACAH); // Carga dato alto en DAC A
spi_write(DACAL); // Carga dato bajo en DAC A
output_high(CS_DAC);
output_low(CS_DAC);
spi_write(DACBH); // Carga dato alto en DAC B
spi_write(DACBL); // Carga dato bajo en DAC B
output_high(CS_DAC);
output_low(CS_DAC);
spi_write(DACCH); // Carga dato alto en DAC C
spi_write(DACCL); // Carga dato bajo en DAC C
output_high(CS_DAC);
```

```
  //Lectura de las entradas analógicas

  canal_0=read_adc();
  can_analog_0_alto=make8(canal_0,1);
  can_analog_0_bajo=make8(canal_0,0);
  can_analog_0_aux=can_analog_0_bajo>>6;
  set_adc_channel(1);
  delay_us(10);
  canal_1=read_adc();
  can_analog_1_alto=make8(canal_1,1);
  can_analog_1_bajo=make8(canal_1,0);
  can_analog_1_aux=can_analog_1_bajo>>4;
  set_adc_channel(0);
  can_analog_0_1=can_analog_0_aux | can_analog_1_aux;

  }

}
```

FICHERO CABINA.H

```
#include <18F458.h>
#device adc=10
#use delay(clock=20000000)
#fuses NOWDT,HS, NOPROTECT,NOBROWNOUT,NOLVP,WRT
#use fast_io(A)
#use fast_io(B)
#use fast_io(C)
#use fast_io(D)
#use fast_io(E)
#bit ADFM = 0xFC1.7
#include <can-18f458_2.c>
#byte PORTA = 0xF80
#byte PORTB = 0xF81
#byte PORTC = 0xF82
#byte PORTD = 0xF83
#byte PORTE = 0xF84

#define VALOR 0xFB

int in_data[8];
int rx_len,sync[1],syncrec[1];
int i,j,a,b,c,d,z,buf,inp,out;
int32 rx_id;
byte dato[8];

long canal_0,canal_1;
int can_analog_0_alto,can_analog_1_alto,can_analog_0_1;
int can_analog_0_bajo,can_analog_1_bajo;
```

```
int can_analog_0_aux,can_analog_1_aux;

int ent_dig_1,ent_dig_2,sal_dig_exp,sal_dig_pic;

struct{

   int1 pinc0;
   int1 pinc1;
   int1 pinc2;
   int1 pinc3;
   int1 pinc4;
   int1 pinc5;
   int1 pinc6;
   int1 pinc7;
}puerto_c;
#byte puerto_c = 0xF82

struct {

int1 pind0;
int1 pind1;
int1 pind2;
int1 pind3;
int1 pind4;
int1 pind5;
int1 pind6;
int1 pind7;

} puerto_d;
#byte puerto_d = 0xF83

struct{
   int cuenta:3;
   int1 sentido1;
   int1 sentido2;
   int1 sentido3;
   int1 sentido4;
   int1 sentido5;
}MENSAJE;

int contador;

int W_TEMP;
#locate W_TEMP=0x100
int STATUS_TEMP;
#locate STATUS_TEMP=0x200
int BSR_TEMP;
#locate BSR_TEMP=0x300
#byte STATUS = 0xFD8
#byte BSR = 0xFE0
```

```
#define RELOJ PIN_C3
#define CS_EXP PIN_E0
#define CS_DAC PIN_E1
#define DIN PIN_C5

int DACAH,DACAL,can_a_dac_temp,can_dac_a_alto;// Variables DAC A
int can_dac_a_bajo,can_dac_a_aux; // Variables DAC A
int DACBH,DACBL,can_b_dac_temp,can_dac_b_alto;// Variables DAC B
int can_dac_b_bajo,can_dac_b_aux; // Variables DAC B
int DACCH,DACCL,can_c_dac_temp,can_dac_c_alto;//Variables DAC C
int can_dac_c_bajo,can_dac_c_aux; // Variables DAC C

struct{

   int1 ADON;
   int1 void1;
   int1 GODONE;
   int CHS:3;
   int ADCS1_0:2;

} ADCON0;
#byte ADCON0=0xFC2

struct{

   int PCFG:4;
   int1 void4;
   int1 void5;
   int1 ADCS2;
   int1 ADFM;

} ADCON1;
#byte ADCON1=0xFC1
```

FICHERO FUNCIONES_CABINA.C

```
void configuracion_expansor(void){

   output_low(CS_EXP);
   spi_write(0x04);
   spi_write(0x01);
   output_high(CS_EXP);

   output_low(CS_EXP);
   spi_write(0x0B);
   spi_write(0xAA);
   output_high(CS_EXP);

   output_low(CS_EXP);
```

```
   spi_write(0x0C);
   spi_write(0xAA);
   output_high(CS_EXP);

   output_low(CS_EXP);
   spi_write(0x0D);
   spi_write(0xAA);
   output_high(CS_EXP);

   output_low(CS_EXP);
   spi_write(0x0E);
   spi_write(0x6A);
   output_high(CS_EXP);

   output_low(CS_EXP);
   spi_write(0x0F);
   spi_write(0x55);
   output_high(CS_EXP);

}

void configuracion_dac(void){

   output_high(CS_DAC);
   output_low(CS_DAC);
   spi_write(0xF0);
   spi_write(0x00);
   output_high(CS_DAC);
   output_low(CS_DAC);
   spi_write(0xF0);
   spi_write(0x04);
   output_high(CS_DAC);
   output_low(CS_DAC);
   spi_write(0xF0);
   spi_write(0x08);
   output_high(CS_DAC);

}

void inicializaciones(void){

   ADFM=0;

   for(i=0;i<8;i++)
   {
   in_data[i]=0x00;
   dato[i]=0x00;
   }

   contador=0;
```

```
}

void un_pulso (void){

switch (contador){

  case 11: {contador=12;
        if (puerto_c.pinc1==1) puerto_c.pinc1=0;
        else puerto_c.pinc1=1;
        puerto_c.pinc2=MENSAJE.sentido1;
        set_timer0(0);
        break;}

  default: set_timer0(0);

  }
}
void dos_pulsos (void){

switch (contador){

  case 21: {set_timer0(0);
        contador=22;
        if (puerto_c.pinc1==1) puerto_c.pinc1=0;
        else puerto_c.pinc1=1;
        puerto_c.pinc2=MENSAJE.sentido1;
        set_timer0(VALOR);
        break;}

  case 22: {set_timer0(0);
        contador=23;
        if (puerto_c.pinc1==1) puerto_c.pinc1=0;
        else puerto_c.pinc1=1;
        puerto_c.pinc2=MENSAJE.sentido2;
        set_timer0(0);
        break;}

  default: set_timer0(0);

  }
}
void tres_pulsos (void){

switch (contador){

  case 31: {set_timer0(0);
        contador=32;
        if (puerto_c.pinc1==1) puerto_c.pinc1=0;
        else puerto_c.pinc1=1;
```

```
            puerto_c.pinc2=MENSAJE.sentido1;
            set_timer0(VALOR);
            break;}

    case 32: {set_timer0(0);
            contador=33;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido2;
            set_timer0(VALOR);
            break;}

    case 33: {set_timer0(0);
            contador=34;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido3;
            set_timer0(0);
            break;}

    default: set_timer0(0);

    }
}
void cuatro_pulsos (void){

switch (contador){

    case 41: {set_timer0(0);
            contador=42;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido1;
            set_timer0(VALOR);
            break;}

    case 42: {set_timer0(0);
            contador=43;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido2;
            set_timer0(VALOR);
            break;}

    case 43: {set_timer0(0);
            contador=44;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido3;
            set_timer0(VALOR);
```

```
            break;}

    case 44: {set_timer0(0);
            contador=45;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido4;
            set_timer0(0);
            break;}

    default: set_timer0(0);

    }

}
void cinco_pulsos (void){

switch (contador){

    case 51: {contador=52;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido1;
            set_timer0(VALOR);
            break;}

    case 52: {contador=53;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido2;
            set_timer0(VALOR);
            break;}

    case 53: {contador=54;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido3;
            set_timer0(VALOR);
            break;}

    case 54: {contador=55;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
            puerto_c.pinc2=MENSAJE.sentido4;
            set_timer0(VALOR);
            break;}

    case 55: {contador=56;
            if (puerto_c.pinc1==1) puerto_c.pinc1=0;
            else puerto_c.pinc1=1;
```

```
            puerto_c.pinc2=MENSAJE.sentido5;
            set_timer0(0);
            break;}

      default: set_timer0(0);

      }
   }
```

4 CONCLUSIONES

Para terminar se comentarán aquellos objetivos de los iniciales que han sido cubiertos con la realización del presente proyecto; así como las futuras líneas de desarrollo del trabajo realizado hasta ahora.

4.1 OBJETIVOS CUBIERTOS

Inicialmente nos planteábamos seis objetivos fundamentales a conseguir con la realización del proyecto. En mayor o menor medida todos han sido cubiertos, si bien existe margen de mejora en cuanto a la programación del código o velocidad de los dispositivos.

Comentar a grandes rasgos que el uso del compilador a sido fundamental parapoder desarrollar el proyecto teniendo en cuanta la coexistencia de múltiples interrupciones. El aprovechamiento de la herramienta ha sido importante y ha reducido de manera drástica los tiempos de programación y depuración.

En cuanto al CAN-Bus decir que se han alcanzado unas tasas binarias altas (cercanas al máximo) con una fiabilidad alta y robustez.

Por lo que respecta al PIC, la migración de un modelo a otro no ha supuesto ningún handicap ya que se trataba de dispositivos compatibles el cuanto a los pines.

La unificación de los protocolos serie se ha realizado en base al protocolo SPI desechando el I2C ya que teniendo en cuenta que se ha trabajado con muestras gratuitas; los fabricantes ofrecian mejores prestaciones en SPI que en I2C. Esto asimismo ha permitido eliminar algunos cuellos de botella del sistema.

Por último las señales de encoder se han añadido al procesado de

cabeza sin ningún problema. No obstante el desarrollo final puede optimizarse en cuanto a tiempo, a pesar de que su funcionalidad es correcta.

4.2 LÍNEAS DE DESARROLLO

Para finalizar se señalarán algunas de las líneas futuras de desarrollo o posibilidades de ampliación que se presentan con la realización del presente proyecto. Las más destacadas serían las siguientes:

- Montaje SMD para reducción del tamaño de la placa
- Aprovechamiento de las entradas y salidas digitales y analógicas de previsión para por ejemplo incluir sensores de ángulo o similares
- Optimización y mejora en cuanto a tiempos del código de programa
- Eliminación de optoacopladores lineales por no presentar igual relación de transformación. Empleo en su lugar de amplificadores
- Bluetooth para monitorización externa sin cables
- Empleo de nuevo PIC con módulo de procesado de señales de encoder incorporado
- Eliminación del autómata de cabina mediante el procesado y actuación directa de cada una de las placas del sistema
- Utilización de Pocket PC en cabina para la monitorización del sistema, permitiendo así conectividad GSM/GPRS/UMTS de la máquina con las instalaciones de la empresa

En cuanto a estas mejoras del proyecto comentadas decir que se trabaja actualmente sobre alguna de ellas esperando que en un breve espacio de tiempo se conviertan en una realidad, enriqueciendo de este modo el trabajo realizado tanto en este proyecto como en aquellos en los que se apoya y que se realizaron cona anterioridad.

5 DATASHEETS DE COMPONENTES

A continuación se tendrán las siguientes hojas de catálogo de los componenetes externos al PIC. Podremos encontrarlos siguiendo elorden que se expone:

- MAX6957
- MAX5742
- LM358A
- HCPL2531
- DCR022405
- MCP6142
- MCP2551