Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

DISEÑO Y DESARROLLO DE UN MÓDULO DE CONEXIÓN A CANOPEN DE UN SENSOR COMERCIAL FUERZA/PAR

Autor: Alberto López Esteban

Tutor: Alberto Jardón Huete

Director: Juan Carlos González Víctores

I.T.I.ELECTRÓNICA INDUSTRIAL

LEGANÉS, MADRID

FEBRERO DE 2011

TÍTULO: DISEÑO Y DESARROLLO DE MÓDULO DE CONEXIÓN A CANOPEN DE UN SENSOR COMERCIAL FUERZA/PAR

AUTOR: ALBERTO LÓPEZ ESTEBAN

DIRECTOR: JUAN CARLOS GONZÁLEZ VÍCTORES

EL TRIBUNAL

PRRESIDENTE: PAOLO PIERRO

SECRETARIO: ÁLVARO CASTRO GONZÁLEZ

VOCAL: MARIO GARCÍA VALDERAS

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 11 de Febrero de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

PRESIDENTE SECRETARIO VOCAL

Resumen

En los últimos tiempos la investigación en robótica se está centrando en el perfeccionamiento e incremento de la autonomía de los sistemas robotizados. Para ello, los robots, tanto industriales como asistenciales, incorporan cada vez más elementos de sensorización externa, para conseguir una mayor eficacia y flexibilidad a la hora de realizar sus tareas.

A pesar de que el sector de la sensorización está muy desarrollado y que existen infinidad de dispositivos con muchas y diferentes características, la mayoría de estos han sido diseñados para trabajar en entornos industriales en los que prima la robustez por encima de otras características como tamaño, peso o flexibilidad. Este problema surge también a la hora de equipar al robot asistencial ASIBOT con una unidad de control de fuerza y par.

Esta circunstancia ha dado lugar al diseño y desarrollo de un módulo que se compone del sistema de adquisición de datos de un sensor comercial de fuerza/par y la electrónica necesaria para integrarlo en un sistema de comunicaciones CAN-Bus como es el que posee el robot asistencial ASIBOT. Con estas premisas, se ha diseñado un dispositivo de tamaño reducido, muy ligero y que permite una gran flexibilidad para ser utilizado en diferentes plataformas.

Para el desarrollo del proyecto se ha partido de un sensor comercial de fuerza/par de JR3®, que facilita la lectura de fuerza y par a través de una comunicación serie. Estos datos serán recogidos por un microcontrolador y serán tratados para posteriormente poder ser enviados a través del CAN-Bus, para lo que se ha diseñado e implementado toda la electrónica necesaria. El diseño software se ha realizado utilizando las herramientas facilitadas por el entorno de desarrollo de Microchip®.

Abstract

Recently robotics research is focusing on improving and increasing the autonomy of robotic systems. To do this, robots, both industrial and assistive, increasingly incorporating elements of external sensor network to achieve greater efficiency and flexibility to perform their tasks.

Although the field of sensor network is very developed and there are plenty of devices with many different features, most of these have been designed to work in industrial environments where robustness premium over other features such as size, weight and flexibility. This problem also arises when wanting to equip the assistive robot ASIBOT with a force/torque unit control.

This has resulted in the design and development of a module that consists of data acquisition system of a commercial sensor force/torque and electronics needed to integrate a communications system such as CAN-Bus. With these premises, we designed a small, very lightweight device, that allows great flexibility to be used on different platforms.

The starting point of the development of this project is a JR3® commercial force/torque sensor, which facilitates the reading of force and torque through a serial communication. Data will be collected by a microcontroller and will be treated and later to be sent through the CAN-Bus, for which we designed and implemented all the necessary electronics. The software design was performed using the tools provided by Microchip's® development environment.

Agradecimientos

En primer lugar, me gustaría agradecer al director de mi proyecto Juan González toda la ayuda ofrecida y el tiempo dedicado al proyecto.

Gracias a mi novia Teresa por su apoyo incondicional, por haberme escuchado en los malos momentos y saber darme siempre el consejo adecuado para seguir adelante. Con ella llegar hasta aquí ha sido mucho más fácil.

A mi gran amigo Dani, por lo buena persona que es, porque una hoja de agradecimientos se quedaría corta para explicar todo lo que me ha ayudado en la realización de este proyecto.

Gracias a mis padres porque me han respaldado en todo momento y porque sin ellos nada de esto hubiera sido posible. A mi hermana y mi cuñado porque siempre están dispuestos a ayudarme y, cómo no, a mi sobrinita Laura.

A todos los compañeros de la universidad que ya son amigos, Miguel, Joni, Sergio, Jesús, por todas las penas y alegrías (sobre todo alegrías) que hemos vivido y compartido durante estos años. A mis amigos de toda la vida Joni, Iván y Linares, por haber estado, por estar y porque sé que estarán, y a toda la familia de mi novia, por ser como son y hacerme sentir siempre uno más de la familia.

Agradecer también a Sito y a Iván por esas sesiones de pruebas CAN y ayudarme de manera desinteresada cuando estaba un poco perdido. También a todo el departamento de Ingeniería de Sistemas y Automática, en especial a Ángela, Fernando y al laboratorio 1.3.C13.

¡Muchas gracias a todos!

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años se han conseguido grandes avances en el campo de la robótica asistencial, área de la robótica especializada en el diseño y desarrollo de equipos que interactúan directamente con el individuo para su rehabilitación, bien sea por la pérdida de capacidad en la movilidad de sus miembros o bien por la pérdida física de uno o más de ellos. El desarrollo de este tipo de robótica tiene como objetivo principal mejorar la calidad de vida de personas con algún tipo de discapacidad o de avanzada edad, ya que como demuestran recientes estudios demográficos, el número de personas mayores de 65 años se incrementa año tras año de una forma muy significativa [1].

Es de allí donde surge el nombre de ASIBOT, robot de asistencia portátil realizado por el grupo *RoboticsLab*, en el departamento de Sistemas y Automática de la Universidad Carlos III de Madrid. El robot ASIBOT, figura 1, permite realizar una gran diversidad de tareas domésticas, tales como dar de comer, afeitar, lavar los dientes, etc.

En términos generales se puede describir el ASIBOT como un robot de cinco grados de libertad, de unos 10 kg de peso, 1,3 m de alcance y capaz de soportar 2 kg en cada uno de sus extremos. Su principal característica es que todo el sistema de control necesario para su funcionamiento se encuentra a bordo del robot, con lo que tan solo es necesario conectarlo a unos puntos de anclaje (*Docking Stations*) que se encontrarán en el entorno de trabajo del robot y que proporcionarán a éste puntos de apoyo y la tensión de alimentación necesaria para empezar a trabajar (24 V_{CC}).

Figura 1. Brazo robótico ASIBOT anclado en una Docking Station

1.1 Motivaciones

Como es bien sabido, los controladores de los robots, tanto industriales como asistenciales, permiten desarrollar programas para que el robot realice de forma repetitiva una secuencia de movimientos prefijados. Para estas operaciones, sólo es necesario disponer de una serie de sensores que permitan conocer la posición del robot en cada uno de los ejes, como pueden ser encoders [2]. Sin embargo, en la actualidad existe un amplio rango de tareas y procesos, tales como por ejemplo los ya mencionados de afeitado o maquillaje, donde no basta con una resolución muy alta de encoders, sino que se necesita un conocimiento del medio donde actuará el robot. En estos casos, además de los habituales sensores de posición, se suele necesitar un conjunto de sensores externos para poder realizar tareas donde es indispensable determinar el instante y lugar de contacto del robot con el entorno o usuario. Los sensores externos más comunes utilizados en robótica se pueden dividir en dos grandes grupos: los de contacto (p. ej. sensores de fuerza) y los de no contacto (p. ej. sistemas de visión).

Los sensores de fuerza proporcionan información sobre las fuerzas y pares aplicados en el extremo del manipulador, permitiendo obtener mejoras en las soluciones para una gran variedad de problemas. Así, podemos encontrar desde aplicaciones muy simples en las que los sensores de fuerza garantizan una operación segura del robot, detectando que no hay colisión entre el robot y el entorno, hasta aplicaciones más complejas que pueden incluir la detección de diferentes tipos de objetos, el manejo de piezas orientadas de forma aleatoria, o mejorar una interacción robot-usuario donde se necesite de un contacto suave (afeitado, maquillaje…).

Este proyecto nace con el objetivo de dotar al sistema robótico ASIBOT de un sensor de fuerza/par y del diseño del hardware que lo gobierne, aportándole la sensibilidad necesaria para realizar con mayor precisión tareas como las descritas anteriormente.

1.2 Objetivos y línea de desarrollo

Para ello, se ha propuesto una solución en la que se implementa un nuevo hardware de adquisición de datos del sensor para que el robot no pierda portabilidad a la hora de equiparlo con un sensor de fuerza/par.

Para ello el primer objetivo es el estudio del sensor elegido, analizando su comportamiento y sus diferentes características.

A continuación se realizará el diseño del módulo a nivel hardware, primero seleccionando los componentes adecuados y, segundo, creando una placa en circuito impreso que implementa todos los circuitos diseñados.

El siguiente paso será desarrollar el software necesario para realizar la adquisición de datos del sensor y darles el tratamiento adecuado, para posteriormente ser enviados a través del bus de comunicaciones CAN.

Finalmente se realizarán una serie de pruebas en sistemas de control con comunicación CAN para comprobar el comportamiento del módulo diseñado en tiempo real de ejecución.

1.3 Estructura del documento

Este documento se divide en cinco capítulos de los cuales, este primero se ha dedicado a introducir el marco en el cual se encuadra el proyecto, así como las motivaciones que dieron lugar al diseño y desarrollo del módulo de comunicaciones y objetivos que inicialmente se pretenden alcanzar.

El segundo capítulo está dedicado al estado del arte del actual proyecto, donde se describirá los diferentes controles de fuerza en robótica y los protocolos de comunicaciones basados en CAN.

En el tercer capítulo se trata en detalle el desarrollo del diseño del módulo de comunicaciones, tanto a nivel *hardware*, teniendo que realizar una placa de circuito impreso, como a nivel *software*, usando las herramientas de Microchip®.

En el cuarto capítulo se explicarán las pruebas realizadas y los resultados obtenidos conforme a éstas.

En el capítulo quinto se hará una evaluación del proyecto explicando las conclusiones a las que se ha llegado una vez finalizado éste y si cumple todos los objetivos inicialmente presentados al inicio de este proyecto. Aparte, se incluirán y expondrán posibles ampliaciones y futuras mejoras que pueden ser llevadas a cabo en posteriores versiones del dispositivo.

Para terminar se dedicarán los anexos para detallar otros aspectos del proyecto como el presupuesto o el listado de componentes utilizados, incluyendo la información más relevante sobre éstos.

Capítulo 2

Estado del arte

A continuación se introducirá al lector en las diferentes disciplinas que abarca este proyecto, presentándole para cada una de ellas los conceptos básicos, generalidades, evolución a lo largo de los años, clasificaciones, diseños de interés, etc., a fin de que la lectura del Proyecto de Fin de Carrera pueda realizarse sin dificultad, incluso para aquellos que nunca hayan tratado con los temas aquí abordados. Las disciplinas que vamos a revisar son las siguientes: control de fuerza en robótica, donde se detallarán tanto las diferentes estrategias existentes de control de fuerzas como las arquitecturas de control de éstas; un pequeño estudio comparativo entre diversos sensores de fuerza/par existentes en el mercado; una descripción del protocolo CAN-Bus, incluyendo la topología y dispositivos necesarios para crear un nodo de comunicaciones basado en dicho bus; y una descripción del protocolo CANopen, describiendo los servicios necesarios para implementar su modelo de comunicación.

2.1 Control de fuerza en robótica

Existen un gran número de tareas desempeñadas por robots manipuladores (industriales y asistenciales) en las que es deseable tener un control de fuerza de contacto. Ejemplos típicos de tareas que exigen control de fuerza e interacción en el sector de la robótica asistencial son: cepillado dental, recogida, transporte y colocación de objetos, etc. Las fuerzas de contacto generadas que dependen de la rigidez de la superficie de contacto de las herramientas, deben ser debidamente controladas, pudiendo ser utilizadas dos estrategias para controlar las fuerzas de contacto: el control activo de fuerza y el control pasivo de fuerza. A modo ilustrativo, y a fin de mostrar a grandes rasgos los campos de investigación que abre el desarrollo principal de este proyecto, se describirán algunos de los modelos de control de fuerza más utilizados en robótica.

2.1.1 Control pasivo de fuerza

En el control pasivo de fuerza, las fuerzas de contacto deben ser controladas para que las tareas sean ejecutadas con éxito, bastando que éstas se mantengan dentro de un determinado rango de valores seguros, no siendo necesario conocer su valor exacto. En este caso las fuerzas de contacto son un efecto “indeseado”, inherente a

las tareas. En el control pasivo no existe medición de fuerza, siendo la trayectoria del elemento final modificada por las fuerzas de interacción. En esta situación la estrategia usada es normalmente la de agregar alguna flexibilidad en el elemento terminal, con el objeto de amortiguar los impactos y aumentar la tolerancia a eventuales errores de posicionamiento, complementada con una planificación cuidadosa de la trayectoria y de la aproximación a los objetos. Este procedimiento implica un conocimiento detallado de todo el espacio de trabajo del robot, teniendo perfectamente identificados todos los posibles obstáculos a su movimiento. Conociendo con exactitud y anticipación la posición de cada elemento de dicho espacio de trabajo, es posible definir las trayectorias que los eviten, así como aproximarse con cautela a aquellos con los que es necesario un contacto. En el control pasivo de fuerza, no es necesario utilizar un sensor de fuerza o introducir alteraciones en la programación del sistema de control. Este tipo de control es barato y simple, y constituye la aplicación más común en robótica industrial. Existen dispositivos en el mercado de tipo RCC (*Remote Centre Compliance*), que permiten añadir flexibilidad al elemento terminal protegiendo la herramienta contra impactos y errores de posicionamiento, permitiendo también compensar errores de alineamiento. Un RCC efectúa correcciones cinéticas de las desviaciones del elemento terminal [3].

2.1.2 Control activo de fuerza

En el control activo, las fuerzas de contacto deben ser controladas porque de eso depende la eficiencia y el grado de éxito de la tarea a efectuar. En este caso, las fuerzas de contacto no son indeseables y son necesarias para el éxito de la tarea. Una característica de la tarea a efectuar es que las fuerzas de contacto asuman un valor determinado, o que obedezcan a un determinado perfil. En el control activo, el valor de la fuerza de contacto es retroalimentado a un controlador, siendo usado para modificar o generar en línea las trayectorias deseadas. El control activo de fuerza es fundamental en tareas industriales de pulimento y lijado de superficies rígidas, así como en la robótica asistencial. La manipulación de objetos frágiles o fácilmente deformables, se pueden beneficiar mucho con este método, centrándose especialmente en tareas tecnológicas en las que el factor fuerza/momento de trabajo es esencial.

2.1.3 Arquitectura de control de fuerzas

El estudio del control de fuerza en robots se inició en la década de los 50 y 60 del siglo pasado, surgiendo obviamente grandes problemas de estabilidad de difícil solución. El avance en los recursos computacionales y en algoritmos sofisticados permitió enfrentar los problemas de estabilidad, desarrollándose así diversas arquitecturas de controladores de fuerza, entre los que se encuentran: el control

explícito de fuerza, el de impedancia [4], el implícito de fuerza, el híbrido [5], el de rigidez, y el de amortiguamiento.

2.1.3.1 Control explícito de fuerza

Las dos principales estrategias para el control de fuerza son: el control explícito de fuerza y el control de impedancia. Sin embargo se ha demostrado, teórica y experimentalmente, que un control de impedancia contiene, o es equivalente, a un controlador explícito de fuerza [6]. El control explícito de fuerza, traduce en el comando directo, explícito, la fuerza de referencia con el objetivo de minimizar la función de error de fuerza, siguiendo la ecuación 1.

$$E_f = f_d - f_{md}$$

Ecuación 1

Donde:

- E_f: Es el comando explícito de la fuerza.
- f_d: Es la fuerza de referencia o deseada.
- f_{md}: Es la fuerza medida.

Se han propuesto dos tipos de controles explícitos de fuerza: El control explícito directo (basado en fuerza) y el control explícito indirecto (basado en posición).

Control explícito directo

El control explícito de fuerza directo, ilustrado en la figura 2, tiene como base un controlador de fuerza, F, que compara las señales de fuerza deseada, *fd*, con la medida, *fmd*, y las procesa, suministrando una señal de comando directamente a la planta (robot), G. Una fuerza de referencia también puede ser retroalimentada positivamente y sumada a la señal de comando de la planta. El controlador F normalmente cuenta con un subconjunto controlador PID (p.ej.: P, I, PD, PI, y PID). Si los resultados son adecuados, no es necesario recurrir a técnicas más sofisticadas y matemáticamente más exigentes. En caso de que resulte que el controlador PID (o un subconjunto de este) es inadecuado, la comprensión del sistema de control con PID proporciona información muy útil para análisis más complejos.

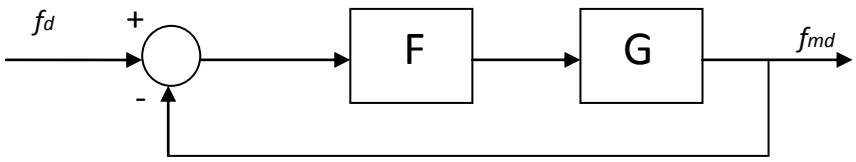


Figura 2. Control explícito directo de fuerza (G representa la planta y F es un controlador).

Control explícito indirecto

En el control explícito indirecto de fuerza, se utiliza un lazo exterior de fuerza, que suministra comandos de posición a un lazo interior, constituido por un controlador de posición. El control explícito indirecto de fuerza fue el primero en ser implementado, por razones prácticas, en la medida que la mayoría de los robots comerciales no permiten el acceso directo a los actuadores. En este tipo de control se consideran dos alternativas las cuales se describen a continuación. La primera alternativa, ilustrada en la figura 3, el lazo exterior suministra referencias de posición a un lazo interior, constituido por un controlador basado en posición, W. La fuerza de referencia es transformada en una posición de referencia a través de una admitancia, la cual es descrita por la inversa de la impedancia de segundo orden según la ecuación 2.

$$A = Z^{-1} = (m_f s^2 + c_f s + k_f)^{-1}$$

Ecuación 2

Donde:

- m_f: Es la masa efectiva.
- c_f: Es la constante de amortiguamiento.
- k_f: Es la rigidez de contacto.

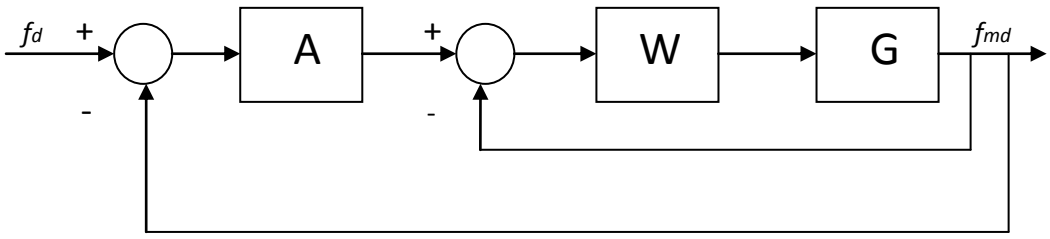


Figura 3. Control explícito indirecto usando relación de admitancia (W es el controlador, G es la planta a controlar: robot).

En la segunda alternativa, ilustrada en la figura 4, el lazo exterior incluye un controlador de fuerza, el que permite suministrar referencias de posición al controlador, G, del sistema de control del robot. Normalmente las referencias de posición son suministradas en la forma de error de posición. Los métodos de control explícito indirecto de fuerza, pueden ser reescritos como métodos de control explícitos directos de fuerza. En realidad los métodos indirectos, basados en posición, difieren de los métodos directos, basados en fuerza, por la adición de rigidez a la planta.

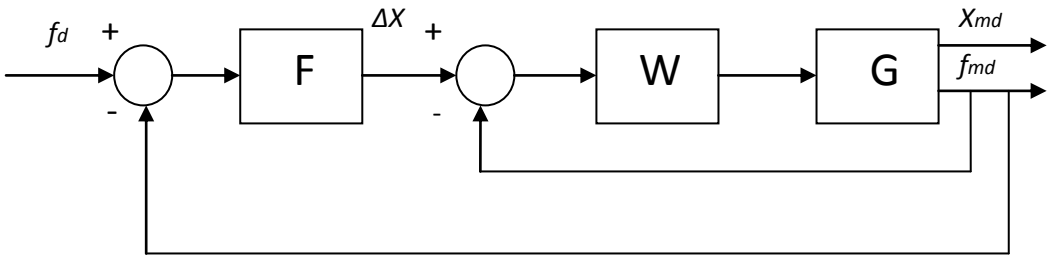


Figura 4. Controlador explícito indirecto usando un controlador de fuerza en el lazo exterior (F es el controlador de fuerza, ΔX es el error de posición, W es el controlador de posición, G es la planta a controlar: robot).

2.2 Comparativa de prestaciones de sensores de fuerza/par

A fin de justificar la elección del sensor utilizado en este diseño, se realizará una comparativa de tres sensores de fuerza/par con características similares existentes en el mercado y pertenecientes a tres de las compañías más importantes del sector. La comparativa se realizará analizando los rasgos más a tener en cuenta cuando se trata de elegir un sensor de este tipo. Los tres sensores a analizar son:

- Sensor de fuerza/par modelo 50M31a-I25 de JR3®.

Figura 5. Sensor de JR3® modelo 50M31A-I25

- Sensor de fuerza/par modelo Gamma SI-65-5 de ATI®.

Figura 6. Sensor de ATI® modelo Gamma SI-65-5

- Sensor de fuerza/par modelo FS6 de AMTI®.

Figura 7. Sensor de AMTI® modelo FS6

2.2.1 Propiedades físicas

En la tabla 1 se pueden observar las diferencias según las propiedades físicas mostradas por cada modelo.

Tabla 1. Comparativa según propiedades físicas

Modelo	50M31A-I25 de JR3®	Gamma SI-65-5 de ATI®	FS6 de AMSI®
Peso[g]	140	254	100
Diámetro[mm]	100	75,4	37,8
Altura[mm]	40	33,3	63,5
Montaje en superficie	Sí	Sí	No
Capacidad de carga (load capacities)			
Fx, Fy[N]	100	65	220
Fz[N]	200	200	440
Mx, My[Nm]	5	5	11
Mz[Nm]	5	5	6
Rigidez (stiffness)			
Fx, Fy[N/m]	$5{,}07 \cdot 10^6$	$9{,}1 \cdot 10^6$	$2 \cdot 10^6$
Fz[N/m]	$51{,}62 \cdot 10^6$	$18 \cdot 10^6$	$3 \cdot 10^6$
Mx, My[Nm/rad]	$11{,}8 \cdot 10^3$	$11 \cdot 10^3$	$20 \cdot 10^4$
Mz[Nm/rad]	$2{,}9 \cdot 10^3$	$16 \cdot 10^3$	$20 \cdot 10^4$

2.2.2 Propiedades eléctricas

A la hora de diseñar un módulo de conexión a un sensor hay que tener muy en cuenta las especificaciones eléctricas mostradas por éste, para, por ejemplo, diseñar el circuito de alimentación de forma adecuada. En la tabla 2 se muestra la comparativa según este tipo de características.

Tabla 2. Comparativa según propiedades eléctricas

Modelo	50M31A-I25 de JR3®	Gamma SI-65-5 de ATI®	FX1901 de Measurement Industries®
Vexcitación[V]	8	10	15
Consumo[mA]	250	400	150
Linealidad[± % sobre Fondo de Escala]	0,5	0,5	0,2
Histéresis[± % sobre Fondo de Escala]	0,05	0.06	0,2

2.2.3 Especificaciones técnicas

En este apartado se pretende mostrar otras prestaciones de diferente campo pero de igual importancia a la hora de elegir el sensor más adecuado para el diseño que se quiere llevar a cabo. Por ejemplo, según el tipo de salida, ya que si ésta es digital no requiere conversión A/D y la transmisión muestra mayor fiabilidad y exactitud. En la tabla 3 se muestran estas otras características.

Tabla 3. Comparativa según especificaciones técnicas

Modelo	50M31a-I25 de JR3®	Gamma SI-65-5 de ATI®	FX1901 de Measurement Industries®
Tipo de salida	Digital	Digital	Analógica
Tipo de conector	RJ-11	DB-15	-
Frecuencia de muestreo[Khz]	64	28,5	-
Resolución del ADC	16	16	-
Tarjeta receptora de datos	Sí	Sí	-

2.3 Protocolo CAN-Bus

CAN (acrónimo del inglés ***Controller Area Network***) es un protocolo de comunicación serie desarrollado por la firma alemana Robert Bosch GmbH[7], basado en una topología bus para la transmisión de mensajes en entornos distribuidos. En la figura 8 se muestra una visión esquemática de un sistema CAN-Bus.

CAN fue desarrollado, inicialmente para aplicaciones en los automóviles y por lo tanto la plataforma del protocolo es resultado de las necesidades existentes en el área de la automoción. La Organización Internacional para la Estandarización (ISO, *International Organization for Standarization*) define dos tipos de redes CAN: una red de alta velocidad (hasta 1 Mbps), bajo el estándar ISO 11898-2, destinada para controlar el motor e interconectar la unidades de control electrónico (ECU); y una red de baja velocidad tolerante a fallos (menor o igual a 125 Kbps), bajo el estándar ISO 11519-2/ISO 11898-3, dedicada a la comunicación de los dispositivos electrónicos internos de un automóvil como son control de puertas, techo corredizo, luces y asientos.

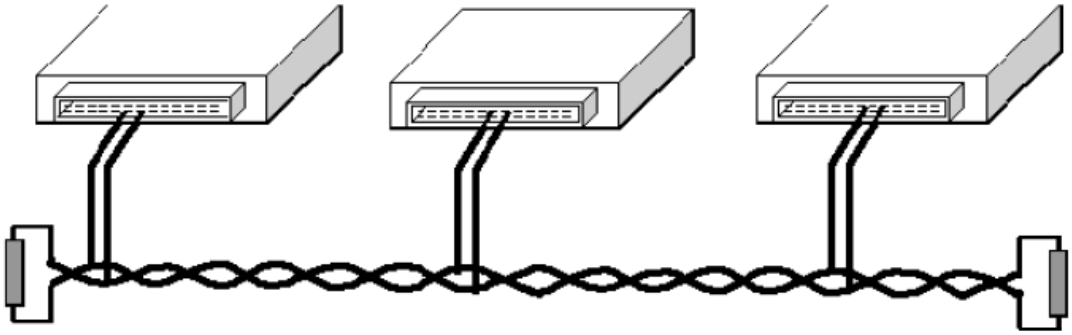


Figura 8. Sistema CAN-Bus

2.3.1 Características principales de CAN

CAN se basa en el modelo productor/consumidor, el cual es un concepto, o paradigma de comunicaciones de datos, que describe una relación entre un productor y uno o más consumidores. CAN es un protocolo orientado a mensajes, es decir la información que se va a intercambiar se descompone en mensajes, a los cuales se les asigna un identificador y se encapsulan en tramas para su transmisión. Cada mensaje tiene un identificador único dentro de la red, con el cual los nodos deciden aceptar o no dicho mensaje.

Dentro de sus principales características se encuentran:

- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción por multidifusión (*multicast*) con sincronización de tiempos.
- Sistema robusto en cuanto a consistencia de datos.
- Sistema multimaestro.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas
- Distinción entre errores temporales y fallas permanentes de los nodos de la red, y desconexión autónoma de nodos defectuosos.

2.3.2 Protocolo de comunicaciones CAN

CAN [16] es un protocolo de comunicaciones serie que soporta control distribuido en tiempo real con un alto nivel de seguridad y multiplexación.

El establecimiento de una red CAN para interconectar dispositivos tiene la finalidad de sustituir o eliminar el cableado. Los sensores, sistemas antideslizantes, etc. se conectan mediante una red CAN a velocidades de transferencia de datos de hasta 1 Mbps.

De acuerdo al modelo de referencia OSI (*Open Systems Interconnection*, Modelo de Interconexión de Sistemas Abiertos), la arquitectura de protocolos CAN incluye tres capas: **física**, **de enlace de datos** y **aplicación**, además de una capa especial para gestión y control del nodo llamada **capa de supervisor**.

- **Capa física:** define los aspectos del medio físico para la transmisión de datos entre nodos de una red CAN, los más importantes son niveles de señal, representación, sincronización y tiempos en los que los bits se transfieren al bus. La especificación del protocolo CAN no define una capa física, sin embargo, los estándares ISO 11898 establecen las características que deben cumplir las aplicaciones para la transferencia en alta y baja velocidad.

- **Capa de enlace de datos:** define las tareas independientes del método de acceso al medio, además debido a que una red CAN brinda soporte para procesamiento en tiempo real a todos los sistemas que la integran, el intercambio de mensajes que demanda dicho procesamiento requiere de un sistema de transmisión a frecuencias altas y retrasos mínimos. En redes multimaestro, la técnica de acceso al medio es muy importante ya que todo nodo activo tiene los derechos para controlar la red y acaparar los recursos. Por lo tanto la capa de enlace de datos define el método de acceso al medio así como los tipos de tramas para el envío de mensajes.

Cuando un nodo necesita enviar información a través de una red CAN, puede ocurrir que varios nodos intenten transmitir simultáneamente. CAN resuelve lo anterior al asignar prioridades mediante el identificador de cada mensaje, donde dicha asignación se realiza durante el diseño del sistema en forma de números binarios y no puede modificarse dinámicamente. El identificador con el menor número binario es el que tiene mayor prioridad.

El método de acceso al medio utilizado es el de Acceso Múltiple por Detección de Portadora, con Detección de Colisiones y Arbitraje por Prioridad de Mensaje (CSMA/CD+AMP, *Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority*). De acuerdo con este método, los nodos en la red que necesitan transmitir información deben esperar a que el bus esté libre (detección de portadora); cuando se cumple esta condición, dichos nodos transmiten un bit de inicio (acceso múltiple). Cada nodo lee el bus bit a bit durante la transmisión de la trama y comparan el valor transmitido con el valor recibido; mientras los valores sean idénticos, el nodo continúa con la transmisión; si se detecta una diferencia en los valores de los bits, se lleva a cabo el mecanismo de arbitraje(espera de tiempo aleatorio).

CAN establece dos formatos de tramas de datos (*data frame*) que difieren en la longitud del campo del identificador, las tramas estándares (*standard frame*) con un identificador de 11 bits definidas en la especificación CAN 2.0A, y las tramas extendidas (*extended frame*) con un identificador de 29 bits definidas en la especificación CAN 2.0B.

Para la transmisión y control de mensajes CAN, se definen cuatro tipos de tramas: de datos, remota (*remote frame*), de error (*error frame*) y de sobrecarga (*overload frame*). Las tramas remotas también se establecen en ambos formatos, estándar y extendido, y tanto las tramas de datos como las remotas se separan de tramas precedentes mediante espacios entre tramas (*interframe space*).

En cuanto a la detección y manejo de errores, un controlador CAN cuenta con la capacidad de detectar y manejar los errores que surjan en una red. Todo error detectado por un nodo, se notifica inmediatamente al resto de los nodos.

- **Capa de supervisor:** La sustitución del cableado convencional por un sistema de bus serie presenta el problema de que un nodo defectuoso puede bloquear el funcionamiento del sistema completo. Cada nodo activo transmite una bandera de error cuando detecta algún tipo de error y puede ocasionar que un nodo defectuoso pueda acaparar el medio físico. Para eliminar este riesgo el protocolo CAN define un mecanismo autónomo para detectar y desconectar un nodo defectuoso del bus, dicho mecanismo se conoce como aislamiento de fallos.

- **Capa de aplicación:** Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación. Entre las capas de aplicación más utilizadas cabe mencionar CAL (*CAN Application Layer*) en la que se basa el protocolo **CANopen**, pero también existen otras como DeviceNet, SDS (*Smart Distributed System*), OSEK o CANKingdom.

2.3.3 Capa de aplicación, CAL

Fue una de las primeras especificaciones producidas por CiA (CAN in Automation) [8] basada en un protocolo existente desarrollado originalmente por Philips Medical Systems. Ofrece un ambiente orientado a objetos para el desarrollo de aplicaciones distribuidas de cualquier tipo, basadas en CAN.

Esta capa está compuesta por los siguientes cuatro servicios:

- CMS (*CAN-based Message Specification*): ofrece objetos de tipo variable, evento y dominio para diseñar y especificar cómo se accede a la funcionalidad de un dispositivo través de su interfaz CAN.

- NMT (*Network ManagemenT*): proporciona servicios para la gestión de la red. Realiza las tareas de inicializar, arrancar, parar o detectar fallos en los nodos. Se basa en el concepto de maestro-esclavo, habiendo sólo un NMT maestro en la red.

- DBT (*DistriBuTor*): se encarga de asignar de forma dinámica los identificadores CAN (de 11 bits), también llamados **COB-ID** (*Communication Object Identifier*). También se basa en el modelo maestro-esclavo, existiendo sólo un DBT maestro.

- LMT (*Layer ManagemenT*): permite cambiar ciertos parámetros de las capas como por ejemplo el identificador de un nodo (**Node-ID**) o la velocidad del bus CAN.

2.3.4 Elementos que componen un sistema CAN-Bus

Existen un número determinado de elementos que deben estar presentes en cualquier sistema CAN-Bus que se desee implementar y que se tratarán a continuación.

2.3.4.1 Medio físico

La información circula por dos cables trenzados que unen todas las unidades de control que forman el sistema. Esta información se trasmite por diferencia de tensión entre los dos cables, de forma que un valor alto de tensión representa un "1" y un valor bajo de tensión representa un "0". La combinación adecuada de unos y ceros conforman el mensaje a trasmitir.

Como se puede observar en la figura 9, en un cable de CAN-Bus los valores de tensión oscilan entre 0V y 2.25V, por lo que se denomina cable *L* (*Low*) y en el otro, el cable *H* (*High*) lo hacen entre 2.75V y 5V. En caso de que se interrumpa la línea *H* o que se derive a masa, el sistema trabajará con la señal de Low con respecto a masa, en el caso de que se interrumpa la línea *L*, ocurrirá lo contrario. Esta situación permite que el sistema siga trabajando con uno de los cables cortados o comunicados a masa, incluso con ambos comunicados también sería posible el funcionamiento, quedando fuera de servicio solamente cuando ambos cables se cortan. Es importante tener en cuenta que el trenzado entre ambas líneas sirve para anular los campos magnéticos, por lo que no se debe modificar en ningún caso ni el paso ni la longitud de dichos cables.

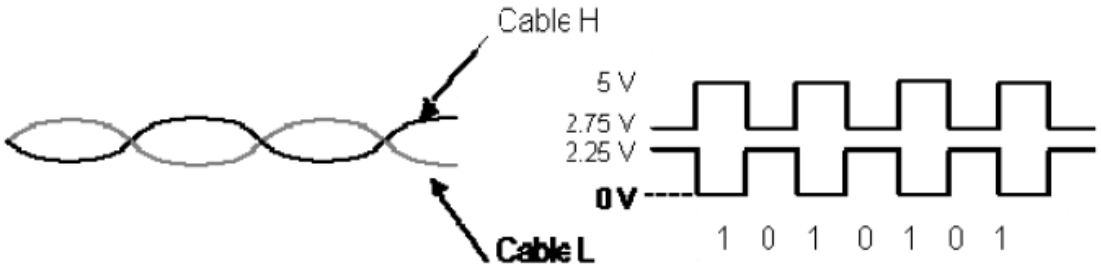


Figura 9. Niveles de tensión presentes en CAN-Bus

2.3.4.2 Elemento de cierre o terminador

Son resistencias conectadas a los extremos de los cables H y L. Sus valores se obtienen de forma empírica y permiten adecuar el funcionamiento del sistema a diferentes longitudes de cables y número de unidades de control abonadas, ya que impiden fenómenos de reflexión que pueden perturbar el mensaje. Estas resistencias están alojadas en el interior de algunas de las unidades de control del sistema por cuestiones de economía y seguridad de funcionamiento.

2.3.4.3 Controlador

Es el elemento encargado de la comunicación entre el microprocesador de la unidad de control y el trasmisor-receptor. Trabaja acondicionando la información que entra y sale entre ambos componentes.

El controlador está situado en el nodo, por lo que existen tantos como unidades estén conectados al sistema. Este elemento trabaja con niveles de tensión muy bajos y es el que determina la velocidad de trasmisión de los mensajes, que será más o menos elevada según el compromiso del sistema. Este elemento también interviene en la necesaria sincronización entre las diferentes unidades de mando para la correcta emisión y recepción de los mensajes.

2.3.4.4 Transmisor / Receptor

El trasmisor-receptor es el elemento que tiene la misión de recibir y de trasmitir los datos, además de acondicionar y preparar la información para que pueda ser utilizada por los controladores. Esta preparación consiste en situar los niveles de tensión de forma adecuada, amplificando la señal cuando la información se vuelca en la línea y reduciéndola cuando es recogida de la misma y suministrada al controlador.

El trasmisor-receptor es básicamente un circuito integrado que está situado en cada una de las unidades de control abonadas al sistema, trabaja con intensidades próximas a 0.5 A y en ningún caso interviene modificando el contenido del mensaje. Funcionalmente está situado entre los cables que forman la línea Can-Bus y el controlador. En la figura 10 se muestran todos los componentes que constituyen un nodo de conexión al bus.

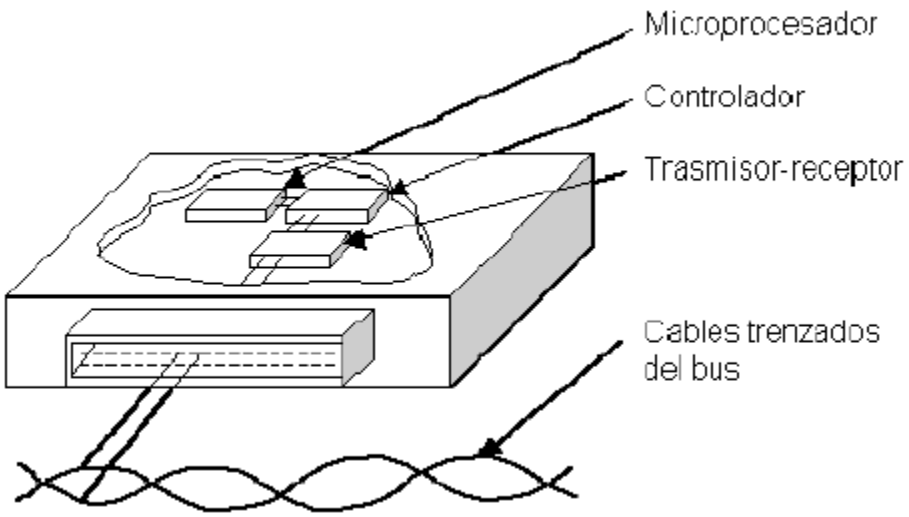


Figura 10. Componentes de un nodo conectado a CAN-Bus

2.4 Protocolo CANopen

Como ya habíamos comentado en la sección 2.3.3, CAL es la capa de aplicación en que se basa CANopen y la que proporciona todos los servicios de gestión de red y mensajes del protocolo, pero no define los contenidos de los objetos CMS ni los tipos de objetos. Es decir, define cómo pero no qué. Aquí es donde CANopen entra en juego. CANopen [9] está por encima de CAL y utiliza un subconjunto de sus servicios y protocolos de comunicación. Proporciona una implementación de un sistema de control distribuido utilizando los servicios y protocolos de CAL.

El concepto central de CANopen es el diccionario de objetos (OD, *Device Object Dictionary*). Es un concepto utilizado por otros buses de campo. No forma parte de CAL.

2.4.1 Diccionario de objetos de CANopen

Es un grupo ordenado de objetos. Describe completamente y de forma estandarizada la funcionalidad de cada dispositivo y permite su configuración mediante mensajes a través del propio bus.

Cada objeto se direcciona utilizando un índice de 16 bits. Para permitir el acceso a elementos individuales de las estructuras de datos también existe un subíndice de 8 bits. En la tabla 4 se puede ver la estructura general del diccionario.

Tabla 4. Estructura de un diccionario de objetos estándar en CANopen

Indice	Objeto	Descripción
0x0000	No usado	
0x0001 – 0x001F	Tipos de dato estáticos	Contiene definiciones de tipos de dato estándar como boolean, enteros, string, punto flotante, etc. Se incluyen como referencia, no pueden ser leídas ni escritas.
0x0020 – 0x003F	Tipos de dato complejos	Contiene definiciones de estructuras predefinidas, compuestas de tipos estáticos, comunes a todos los dispositivos.
0x0040 – 0x005F	Tipos de dato específicos del fabricante	Contiene definiciones de estructuras predefinidas, compuestas de tipos estáticos, específicas de un dispositivo en particular.
0x0060 – 0x007F	Tipos de dato estáticos específicos del perfil del dispositivo	Definiciones de tipos de dato básicos específicos para el perfil del dispositivo.
0x0080 – 0x009F	Tipos de dato complejos específicos del perfil del dispositivo	Definiciones de estructuras específicas para el perfil del dispositivo.
0x00A0 – 0x0FFF	Reservado	
0x1000 – 0x1FFF	Rango para el perfil de comunicaciones	Contiene parámetros de configuración del bus CAN. Estas entradas del diccionario son comunes a todos los dispositivos.
0x2000 – 0x5FFF	Rango para el perfil específico del fabricante	Contiene las extensiones al perfil estándar, realizadas por el fabricante.
0x6000 – 0x9FFF	Rango para perfiles de dispositivo estandarizados	Contiene todos los objetos de datos comunes a un tipo de perfil que pueden ser leídos o escritos desde la red. Algunas de las entradas son obligatorias (funcionalidad requerida) mientras que otras son opcionales (funcionalidad opcional).
0xA000 – 0xFFFF	Reservado	

El rango relevante de objetos va desde el índice 1000 al 9FFF. Para cada nodo de la red existe un OD, diccionario de objetos, que contiene todo los parámetros que describen el dispositivo y su comportamiento en la red.

En CANopen hay documentos que describen perfiles. Hay un perfil de comunicaciones (*communication profile*) donde están descritos todos los parámetros relacionados con las comunicaciones. Además hay varios perfiles de dispositivos (*device profiles*) donde se definen los objetos de un dispositivo en particular.

Un perfil define para cada objeto del diccionario su función, nombre, índice, subíndice, tipos de datos, si es obligatorio u opcional, si es de “sólo lectura”, “sólo escritura” o “lectura-escritura”, etc.

2.4.2 Modelo de comunicaciones

El modelo de comunicaciones de CANopen define cuatro tipos de mensajes (objetos de comunicación):

- Objetos administrativos: son mensajes administrativos que permiten la configuración de las distintas capas de la red así como la inicialización, configuración y supervisión de la misma. Se basa en los servicios NMT, LMT y DBT de la capa CAL.

- *Service Data Objects* (SDO): objetos o mensajes de servicio utilizados para leer y escribir cualquiera de las entradas del diccionario de objetos de un dispositivo. Corresponden a mensajes CAN de baja prioridad.

- *Process Data Objects* (PDO): objetos o mensajes de proceso utilizados para el intercambio de datos de proceso, es decir, datos de tiempo real. Por este motivo, típicamente corresponden a mensajes CAN de alta prioridad.

- Mensajes predefinidos: de sincronización, de emergencia y *time stamp*. Permiten la sincronización de los dispositivos (objetos SYNC) y generar notificaciones de emergencia en forma opcional.

CANopen soporta los modelos de comunicación productor-consumidor en sus variantes *push* y *pull* (figura 11) y punto-a-punto, maestro-esclavo (figura 12). En el modelo *push* los productores colocan los eventos en el canal de eventos y éste se los envía a los consumidores. En el *pull* el flujo de eventos ocurre en el sentido contrario, es decir, los consumidores solicitan eventos al canal de eventos y éste los solicita a los productores.

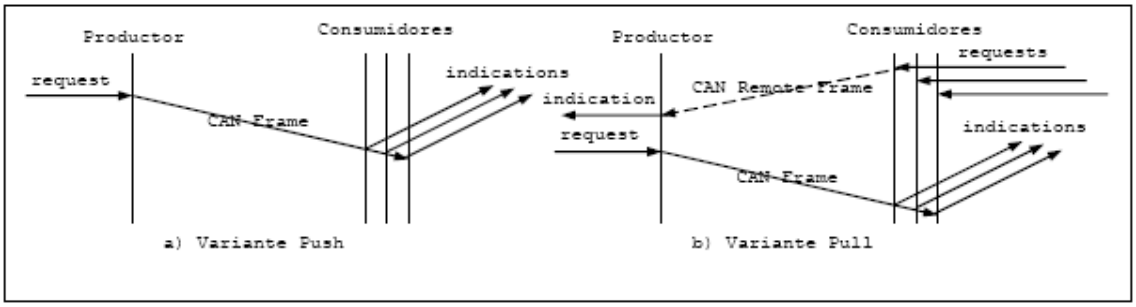


Figura 11. Modelo de comunicación productor-consumidor en CANopen

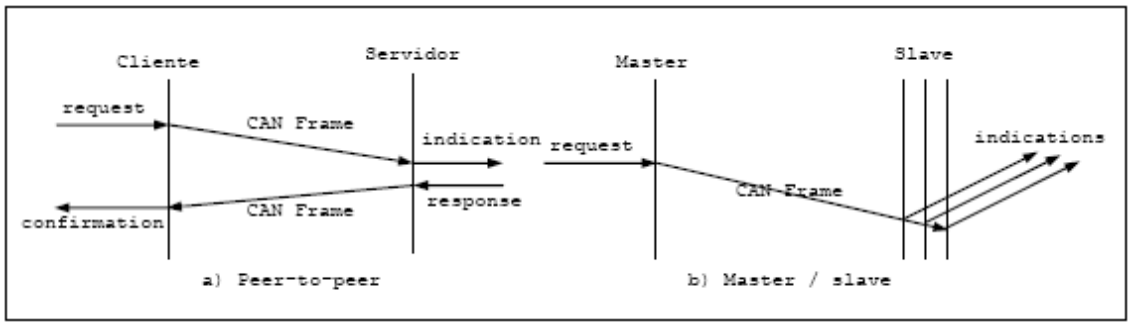


Figura 12. Modelos de comunicación punto-a-punto y maestro-esclavo en CANopen

2.4.3 Identificadores de mensaje

CANopen define la distribución de los identificadores de mensaje (*Predefined Connection Set*) de manera que hay un mensaje de emergencia por nodo, mensajes de sincronización y *time stamp*, un SDO (ocupando dos identificadores), mensajes NMT y cuatro PDOs de transmisión y cuatro de recepción por dispositivo.

El identificador de 11 bits, como se puede ver en la figura 13, se divide en dos partes:

- 4 bits para el código de función
- 7 bits para el identificador de nodo (Node-ID)

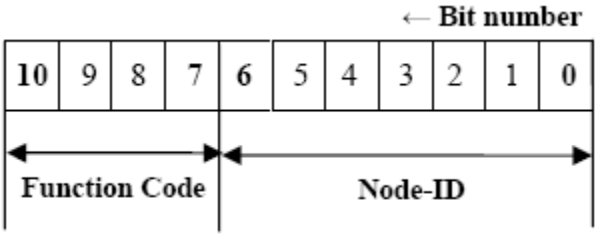


Figura 13. Estructura del identificador de mensajes CAN

La distribución de los identificadores se corresponde con una estructura del tipo maestro-esclavo. El maestro conoce los Node-IDs de todos los esclavos conectados (máximo 127) pero dos esclavos no pueden comunicarse entre sí porque no conocen sus identificadores.

En la tabla 5 se puede ver la distribución general de los identificadores.

Tabla 5. Asignación de los identificadores CAN en CANopen

Broadcast objects of the CANopen Predefined Master/Slave Connection Set			
Object	**Function code (ID-bits 10-7)**	**COB-ID**	**Communication parameters at OD index**
NMT Module Control	0000	000h	–
SYNC	0001	080h	1005h, 1006h, 1007h
TIME STAMP	0010	100h	1012h, 1013h

Peer-to-Peer objects of the CANopen Predefined Master/Slave Connection Set			
Object	**Function code (ID-bits 10-7)**	**COB-ID** *	**Communication parameters at OD index**
EMERGENCY	0001	081h - 0FFh	1024h, 1015h
PDO 1 (transmit)	0011	181h - 1FFh	1800h
PDO 1 (receive)	0100	201h - 27Fh	1400h
PDO 2 (transmit)	0101	281h - 2FFh	1801h
PDO 2 (receive)	0110	301h - 37Fh	1401h
PDO 3 (transmit)	0111	381h - 3FFh	1802h
PDO 3 (receive)	1000	401h - 47Fh	1402h
PDO 4 (transmit)	1001	481h - 4FFh	1803h
PDO 4 (receive)	1010	501h - 57Fh	1403h
SDO (transmit/server)	1011	581h - 5FFh	1200h
SDO (receive/client)	1100	601h - 67Fh	1200h
NMT Error Control	1110	701h - 77Fh	1016h, 1017h

2.4.4 Service Data Objects (SDO)

Normalmente este tipo de objetos es usado para configuración de dispositivos y transferencia de grandes cantidades de datos no relevantes en forma directa para el control del proceso. En comparación con los PDOs, son mensajes de baja prioridad.

Los objetos de servicio o SDOs permiten implementar los modelos de comunicación cliente-servidor o punto-a-punto, para acceder a los diccionarios de objetos de los dispositivos. Para un cierto SDO, el dispositivo cuyo diccionario está siendo accedido es el servidor mientras que el otro dispositivo, el que inicia la actividad, es el cliente. Este tipo de objetos ofrece transferencia de datos sin conexión y con confirmación. Por este motivo, cada SDO involucra el intercambio de dos tramas CAN con diferentes identificadores. Un SDO es representado en CMS como un objeto de tipo *Multiplexed Domain*.

Se definen una serie de protocolos petición-respuesta que se pueden aplicar a los SDOs para su transferencia:

- *Initiate Domain Download*
- *Initiate Domain Upload*
- *Download Domain Segment*
- *Upload Domain Segment*
- *Abort Domain Transfer Download* significa escribir en el diccionario de objetos y *upload* leer del él.

Al ser *Multiplexed Domains* desde el punto de vista de CMS, los SDOs pueden transferir datos de cualquier longitud. Sin embargo CANopen define dos tipos de transferencia para los SDOs basándose en el tamaño de los datos a transferir: transferencia expédita, para datos de longitud menor o igual a 4 *bytes*, y transferencia en segmentos, para datos de longitud mayor de 4 *bytes*.

Los mensajes tanto del cliente como del servidor siempre tienen una longitud de 8 *bytes* aunque no todos contengan información significativa. Es importante tener en cuenta que en CANopen los parámetros de más de un *byte* se envían siempre en la forma ***little endian***, es decir, primero el *byte* menos significativo (LSB).

2.4.4.1 Transferencia expédita

Usada para transmitir mensajes con longitud de datos menor o igual a 4 *bytes*, para lo cual se usan los protocolos *Initiate Domain Download* o *Initiate Domain Upload*.
No se aplica fragmentación, se envía un único mensaje CAN y se recibe la confirmación del servidor.

Un SDO que se transmite del cliente al servidor bajo este protocolo tiene el siguiente formato:

- *Command Specifier* (1 *byte*): contiene información sobre si es subida o descarga de datos, petición o respuesta, transferencia expédita o en segmentos, tamaño de los datos y el *toggle* bit:
  - *Client Command Specifier* (3 bits): 001 para *download* y 010 para *upload*.
  - Ignorado (1 bit): se pone a 0.
  - Número de *bytes* / Ignorado (2 bits): número de *bytes* que no contienen datos. Es válido si los siguientes dos bits son 1. Si no, vale 0. En *upload* se ignora.

- *Transfer Expedited* / Ignorado (1 bit): indica si se trata de una transferencia expédita (a 1) o una transferencia en segmentos (a 0). Tiene que estar siempre a 1 para este tipo de transferencia. En *upload* se ignora.

- *Block Size Indicated* / Ignorado (1 bit): si está a 1 es que el tamaño de los datos está especificado, si no, no. En *upload* se ignora.

- *Index* (2 *bytes*): índice de la entrada del diccionario de objetos del servidor que el cliente desea acceder mediante el SDO actual. Este campo y el siguiente forman el multiplexor del dominio.

- *Subindex* (1 *byte*): subíndice de la entrada del diccionario de objetos del servidor que el cliente desea acceder. Sólo tiene sentido si la entrada es de un tipo complejo. Si se trata de una entrada con tipo estático, debe ser 0.

- Datos / Ignorado (4 *bytes*): datos que se desean enviar al servidor (el valor de una entrada en el diccionario si se está escribiendo). Si es *upload* se ignora.

El servidor, al recibir el mensaje anterior, y si ha accedido con éxito al diccionario, contesta con un mensaje con el siguiente formato:

- *Command Specifier* (1 *byte*):

  - *Server Command Specifier* (3 bits): 011 para *download* y 010 para *upload*.

  - Ignorado (1 bit): se pone a 0.

  - Ignorado / Número de *bytes* (2 bits): si es un *download* se ignora.

  - Ignorado / *Transfer Expedited* (1 bit): si es un *download* se ignora.

  - Ignorado / *Block Size Indicated* (1 bit): si es un *download* se ignora.

- Ignorado / Datos (4 *bytes*): si es un *download* (escritura) se ignora ya que es una confirmación. Si es una lectura o *upload*, contiene el valor leído del diccionario de objetos del servidor.

2.4.4.2 Transferencia en segmentos

Usada para transmitir mensajes con longitud de datos mayor de 4 *bytes*. Se aplica fragmentación en segmentos partiendo los datos en múltiples mensajes CAN. El cliente espera confirmación del servidor por cada segmento.

Para el primer mensaje tanto del cliente como del servidor, se usan los protocolos *Initiate Domain Download* o *Initiate Domain Upload* según corresponda, con el bit *Transfer Expedited* a 0 (para transferencia en segmentos). Si ese bit está a 0 y el siguiente (*Block Size Indicated*) está a 1, significa que el campo de datos (de 4 *bytes*) contiene el número de *bytes* que se van a transmitir (al ser *little endian* el *byte* 4 del mensaje contiene el LSB y el 7 el MSB).

Para los mensajes siguientes se usan los protocolos *Download Domain Segment* o *Upload Domain Segment*, según se quiera leer o escribir una entrada en el diccionario. Los mensajes CAN bajo estos dos protocolos tienen el siguiente formato:

- *Command Specifier* (1 *byte*):
  - *Client Command Specifier* (3 bits): 000 para *download* y 011 para *upload*.
  - *Toggle Bit* (1 bit): es un bit que vale 0 o 1 de forma alternada en segmentos consecutivos. La primera vez vale 0. Teniendo en cuenta que el mecanismo de intercambio sólo permite un mensaje pendiente de confirmación, un único bit es suficiente para esto. El servidor se limita a hacer un eco del valor recibido.
  - Número de *bytes* / Ignorado (3 bits): indica el número de *bytes* que no contienen datos, o cero si no especifica el tamaño. Se ignora en el *upload*.
  - *Last Segment Indication* / Ignorado (1 bit): vale 1 si se trata del último segmento del SDO que se está transmitiendo y 0 si hay más segmentos. Para *upload* se ignora.
- Datos / Ignorado (7 *bytes*): datos que se desean enviar al servidor. Son los *bytes* que no caben en el mensaje CAN inicial. Si es *upload* se ignora.

El servidor, al recibir el mensaje anterior, contesta con un mensaje con el formato:

- *Command Specifier* (1 *byte*):
  - *Client Command Specifier* (3 bits): 001 para *download* y 000 para *upload*.
  - *Toggle Bit* (1 bit): igual que antes.
  - Ignorado / Número de *bytes* (3 bits): se ignora en el *download*.
  - Ignorado / *Last Segment Indication* (1 bit): se ignora en el *download*.

- Ignorado / Datos (7 *bytes*): si es un *download* (escritura) se ignora porque es una confirmación. Si es una lectura o *upload*, contiene el valor leído del diccionario de objetos del servidor.

2.4.4.3 Abort Domain Transfer

Uno de los protocolos antes mencionados y que aún no se ha explicado es el *Abort Domain Transfer*. Existe la posibilidad que al acceder a las entradas del diccionario de objetos del servidor, se produzca un error. Este protocolo es usado en esos casos para notificar tanto a clientes como a servidores. El formato de los mensajes de este protocolo es el siguiente:

- *Command Specifier* (1 *byte*):
  - *Command Specifier* (3 bits): 100 para *Abort Domain Transfer*.
  - Ignorado (5 bits): se ignoran.
- *Index* (2 *bytes*): índice de la entrada del diccionario de objetos del servidor que causó el error que se está notificando. Este campo y el siguiente forman el multiplexor del dominio.
- *Subindex* (1 *byte*): subíndice de la entrada del diccionario de objetos del servidor que causó el error que se está notificando. Sólo tiene sentido si la entrada es de un tipo complejo. Si se trata de una entrada con tipo estático, debe ser 0.
- Código de error (4 *bytes*): código que identifica el error. En la tabla 6 se pueden ver sus posibles valores.

Tabla 6. Códigos de error para SDO *Abort Domain Transfer*

Abort Code	Description
0503 0000	Toggle bit not alternated
0504 0000	SDO protocol timed out
0504 0001	Client/Server command specifier not valid or unknown
0504 0002	Invalid block size (Block Transfer mode only)
0504 0003	Invalid sequence number (Block Transfer mode only)
0503 0004	CRC error (Block Transfer mode only)
0503 0005	Out of memory
0601 0000	Unsupported access to an object
0601 0001	Attempt to read a write-only object
0601 0002	Attempt to write a read-only object
0602 0000	Object does not exist in the Object Dictionary
0604 0041	Object can not be mapped to the PDO
0604 0042	The number and length of the objects to be mapped would exceed PDO length
0604 0043	General parameter incompatibility reason
0604 0047	General internal incompatibility in the device
0606 0000	Object access failed due to a hardware error
0606 0010	Data type does not match, lengh of service parameter does not match
0606 0012	Data type does not match, lengh of service parameter is too high
0606 0013	Data type does not match, lengh of service parameter is too low
0609 0011	Sub-index does not exist
0609 0030	Value range of parameter exceeded (only for write access)
0609 0031	Value of parameter written too high
0609 0032	Value of parameter written too low
0609 0036	Maximum value is less than minimum value
0800 0000	General error
0800 0020	Data can not be transferred or stored to the application
0800 0021	Data can not be transferred or stored to the application because of local control
0800 0022	Data can not be transferred or stored to the application because of the present device state
0800 0023	Object Dictionary dynamic generation fails or no Object Dictionary is present (e.g. OD is generated from file and generation fails because of a file error)

Los SDOs requieren ser definidos en el diccionario de objetos mediante una estructura que contiene parámetros relacionados con la transmisión de los mismos. La estructura para el primer SDO para servidores tiene un índice de 0x1200 mientras que el primer SDO para clientes, se ubica en la entrada 0x1280. En total, en una red CANopen, se pueden definir hasta 128 SDOs para clientes y 128 SDOs para servidores.

2.4.5 Process Data Objects (PDO)

Este tipo de objetos permite intercambiar datos del proceso en tiempo real. Implementa el modelo de comunicaciones productor-consumidor. Los datos se transmiten desde un productor a varios consumidores. Ofrece un servicio de transferencia de datos sin conexión y sin confirmación. No se aplica un protocolo de fragmentación y reensamble de los objetos. Los PDOs están pensados para tráfico de tiempo real de alta prioridad, por lo que es conveniente evitar la sobrecarga que produciría agregar un protocolo de fragmentación y confirmación como el que se usa en los SDOs.

Los mensajes PDO de un nodo o dispositivo pueden dividirse en dos categorías. Los tPDO son aquello mensajes con información del proceso que el nodo transmite (por ejemplo la lectura de un sensor). Por otro lado, los rPDO son los mensajes con información del proceso que el nodo escucha (por ejemplo un nodo que controle la apertura de una bomba escuchará el bus en busca de órdenes). El contenido de un PDO está definido tan sólo por su identificador. Tanto el emisor como el receptor deben conocerlo para poder interpretar su estructura interna.

Cada PDO se describe mediante dos objetos del diccionario:

- PDO *Communication Parameter*: contiene el COB-ID que utiliza el PDO, el tipo de transmisión, tiempo de inhibición y temporizador.
- PDO *Mapping Parameter*: contiene una lista de objetos del OD contenidos en la PDO, incluyendo su tamaño en bits.

CANopen define varios mecanismos de comunicación para la transmisión de PDOs:

- Transmisión asíncrona:
  - Eventos: la transmisión de un mensaje es causada por la ocurrencia de un evento específico definido en el perfil del dispositivo.
  - Temporizador: existe un temporizador que cada cierto tiempo cause la transmisión.
  - Solicitud remota: la transmisión asincrónica de mensajes PDO puede comenzar al recibir una solicitud remota (trama RTR) enviada por otro dispositivo.
- Transmisión sincrónica: la transmisión sincrónica de mensajes PDO es disparada por la expiración de un período de transmisión, sincronizado mediante la recepción de objetos SYNC. Es decir, cada vez que llega un mensaje SYNC, se abre una ventana de transmisión sincrónica. Los PDOs sincrónicos deben ser enviados dentro de esa ventana. Se distinguen dos modos dentro de este tipo de transmisión:
  - Modo cíclico: son mensajes que se transmiten dentro de la ventana abierta por el objeto SYNC. No se transmiten en todas las ventanas sino con cierta periodicidad, especificada por el campo *Transmission Type* del *Communication Parameter* correspondiente.

- Modo acíclico: son mensajes que se transmiten a partir de un evento de la aplicación. Se transmiten dentro de la ventana pero no de forma periódica.

En la tabla 7 se muestran los distintos modos de transmisión de PDOs, definidos por el *Transmission Type* (entero de 8 bits) del *Communication Parameter*.

Tabla 7. Modos de transmisión de PDOs enCANopen

Trans-Mission Type	Condition to trigger PDO (B=both needed, O=one or both)			PDO Transmission
	SYNC*	RTR*	Event*	
0	B	-	B	Sync, acyclic
1-240	O	-	-	Sync, cyclic
241-251	-	-	-	*reserved*
252	B	B	-	Sync, after RTR
253	-	O	-	Async, after RTR
254	-	O	O	Async, manufacturer specific event
255	-	O	O	Async, device profile specific event

*SYNC = objeto SYNC recibido

*RTR = recibida trama RTR

*Event = cambio de valor de un dato, temporizador…

Un PDO puede tener asignado un tiempo de inhibición que define el tiempo mínimo que debe pasar entre dos transmisiones consecutivas del mismo PDO. Forma parte del *Communication Parameter*. Está definido como un entero de 16 bits en unidades de 100 microsegundos.

2.4.6 Mensajes adicionales

2.4.6.1 Mensaje de Time Stamp

Este tipo de objetos representan una cantidad absoluta de tiempo en milisegundos desde el 1 de Enero de 1984. Proporciona a los dispositivos un tiempo de referencia común. La etiqueta temporal o *time-stamp* se implementa como una secuencia de 48 bits.

2.4.6.2 Mensaje de sincronización (SYNC)

En una red CANopen, hay un dispositivo que es el productor de objetos SYNC y una serie de dispositivos consumidores de objetos SYNC. Cuando los consumidores reciben el mensaje del productor, abren su ventana de sincronismo y pueden ejecutar sus tareas sincrónicas. Este mecanismo permite coherencia temporal y coordinación entre los dispositivos. Por ejemplo, un conjunto de sensores pueden

leer las variables del proceso controlado en forma coordinada y obtener así una imagen consistente del mismo.

El COB-ID usado por este objeto de comunicación puede encontrarse en la entrada 0x1005 del diccionario. Para garantizar el acceso de estos objetos al bus, debería asignárseles un COBID bajo. El conjunto predefinido de conexiones de CANopen sugiere usar un valor de 128. El campo de datos del mensaje CAN de este objeto se envía vacío.

Como se puede observar en la figura 14 el comportamiento de estos mensajes viene determinado por dos parámetros (longitud de ventana y período de transmisión).

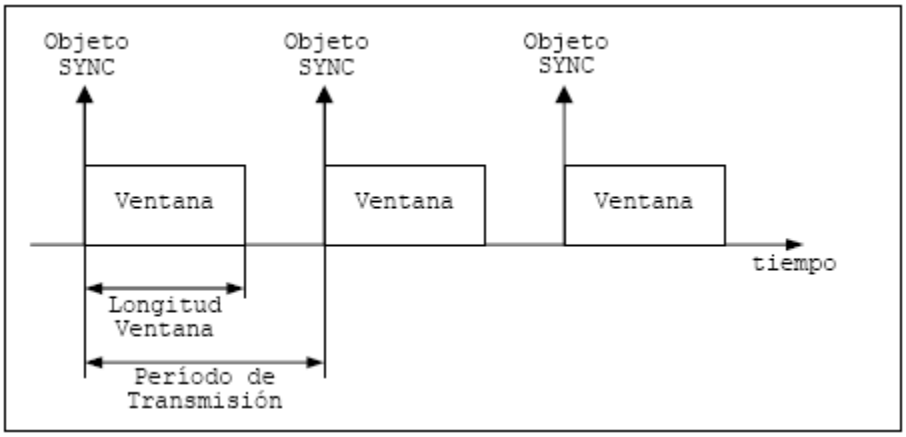


Figura 14. Parámetros de un objeto SYNC en CANopen

La longitud de la ventana o *Synchronous Window Length* puede ubicarse en la entrada 0x1007 del diccionario. El período de transmisión se encuentra en la posición 0x1006.

2.4.6.3 Mensaje de emergencia

Estos mensajes se envían cuando ocurre un error interno en un dispositivo. Se transmiten al resto de dispositivos con la mayor prioridad. Pueden usarse como interrupciones o notificaciones de alertas.

Un mensaje de emergencia tiene 8 *bytes*. En la tabla 8 se muestra su estructura.

Tabla 8. Estructura de un mensaje de emergencia en CANopen

COB-ID	Byte 0-1	Byte 2	Byte 3-7
0x080 + Node_ID	**Emergency Error Code**	**Error Register** (Object 0x1001)	Manufac-turer specific error field

- *Emergency Error Code* (2 *bytes*): código del error que causó la generación del mensaje de emergencia. Se trata de fallos internos de los dispositivos por lo cual, los errores se relacionan con fallos de tensión, de corriente, del software del dispositivo, del adaptador del bus CAN, etc. En la tabla 9 se muestran los códigos correspondientes en hexadecimal.

Tabla 9. Códigos de error para los mensajes de emergencia de CANopen

Emergency Error Code	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	current, device input side
22xx	current, inside the device
23xx	current, device output side
30xx	Voltage
31xx	mains voltage
32xx	voltage inside the device
33xx	output voltage
40xx	Temperature
41xx	ambient temperature
42xx	device temperature
50xx	Device hardware
60xx	Device software
61xx	internal software
62xx	user software
63xx	data set
70xx	Additional modules
80xx	Monitoring
81xx	Communication
8110	CAN overrun
8120	Error Passive
8130	Life Guard Error or Heartbeat Error
8140	Recovered from Bus-Off
82xx	Protocol Error
8210	PDO not processed due to length error
8220	Length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

'xx' es la parte dependiente del perfil del dispositivo

- *Error Register* (1 *byte*): entrada con índice 0x1001 del diccionario de objetos. Cada bit de este registro indica una condición de error distinta cuando está a '1'. En la tabla 10 se observa el significado de cada bit.

Tabla 10. Bits del *Error Register* de los mensajes de emergencia de CANopen

Bit	Significado
0	Error genérico.
1	Problema de corriente.
2	Problema de tensión.
3	Problema de temperatura.
4	Error de comunicaciones.
5	Específico del perfil de dispositivo.
6	Reservado.
7	Específico del fabricante del dispositivo.

- *Manufacturer-specific Error Field* (5 *bytes*): este campo puede usarse para información adicional sobre el error. Los datos incluidos y su formato son definidos por el fabricante del dispositivo.

2.4.6.4 Mensajes de Node/Life Guarding

Se utiliza para saber si un nodo está operativo. La comunicación se basa en el concepto de maestro-esclavo. El NMT maestro monitorea el estado de los nodos. A esto se le llama *node guarding*. Opcionalmente los nodos pueden monitorear el estado del NMT maestro. A esto se le llama *life guarding*. Comienza en el NMT esclavo después de haber recibido el primer mensaje de *node guarding* del NMT maestro. Sirve para detectar errores en las interfaces de red de los dispositivos, pero no fallos en los dispositivos en sí (ya que estos son avisados mediante mensajes de emergencia).

Se puede implementar de dos maneras distintas, pudiendo utilizarse sólo una de ellas a la vez: NMT *Node Guarding* o *Heartbeat*.

NMT *Node Guarding*

El maestro va preguntando a los esclavos si están "vivos" cada cierto tiempo. Si alguno no contesta en un tiempo determinado significa que está caído y se informa de ello. Si los esclavos también monitorean al maestro deben informar de que el maestro está caído si no reciben mensajes de *Node Guarding* de él durante un determinado intervalo. El maestro interroga a los esclavos mediante una trama remota (RTR) con la estructura mostrada en la tabla 11.

Tabla 11. Trama RTR que el NMT maestro envía a los NMT esclavos

COB-ID
0x700 + Node_ID

En la tabla 12 se puede ver el mensaje con el que los NMT esclavos responden a éste.

Tabla 12. Mensaje que los NMT esclavos envían al NMT maestro

COB-ID	Byte 0
0x700 + Node_ID	bit 7: *toggle*, bit 6-0: *state*

El *toggle bit* (bit 7) es un bit que va alternando su valor en cada mensaje de *Node Guarding*. La primera vez vale '0'. Los bits del 0 al 6 indican el estado del nodo. En la tabla 13 se describen sus valores correspondientes.

Tabla 13. Valor del campo *state* en un NMT *Node Guarding* Heartbeat

Value	State
0	Initialising
1	Disconnected *
2	Connecting *
3	Preparing *
4	Stopped
5	Operational
127	Pre-operational

Los estados marcados con * son por si se realiza un *boot-up* extendido

Cada nodo manda un mensaje de *Hearbeat* cada cierto tiempo para informar de que está operativo. En este caso el mensaje de *Boot-up* se considera que es el primer mensaje de *Heartbeat*. Si el NMT maestro deja de recibir estos mensajes durante un tiempo determinado significará que el nodo está caído.

En la tabla 14 se muestra la estructura de este tipo de mensaje, y en la tabla 15 se describe el estado (*state*) en que se encuentra el nodo.

Tabla 14. Estructura de un mensaje de *Heartbeat*

COB-ID	Byte 0
0x700 + Node_ID	*state*

Tabla 15. Valor del campo *state* en un mensaje de *Heartbeat*

state	Meaning
0	Boot-up
4	Stopped
5	Operational
127	Pre-operational

2.4.7 Gestión de la red (NMT)

Aparte de los mensajes predefinidos, CANopen incluye una serie de mensajes para la administración y monitoreo de los dispositivos en la red. Estos están implementados en la capa CAL y reciben el nombre de servicios de gestión de red (*Network ManagemenT*, NMT). Se trabaja con un modelo de comunicaciones maestro-esclavo en el cual un dispositivo es el NMT maestro y el resto los NMT esclavos. Un dispositivo NMT esclavo puede encontrarse en alguno de los siguientes estados:

- ***Initialising***: al encender el dispositivo se pasa directamente a este estado. Después de realizar las labores de inicialización el nodo transmite el mensaje de *Boot-up*, cuya estructura se muestra en la tabla 16, y pasa al estado *Pre-Operational*.

Tabla 16. Estructura del mensaje de *Boot-up*

COB-ID	Byte 0
0x700 + Node_ID	0

Para permitir un reseteo parcial del nodo se subdivide en tres subestados:

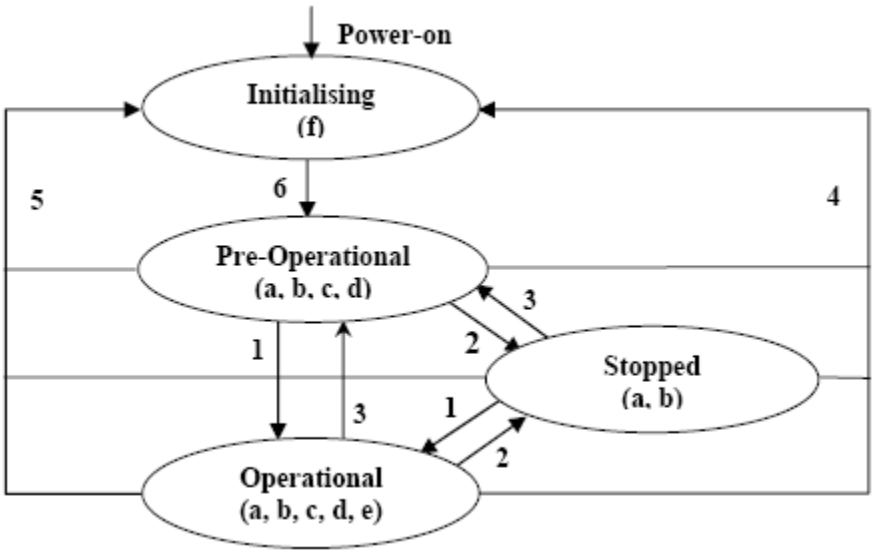
- *Reset-Application*: los parámetros específicos del fabricante y del perfil del dispositivo son fijados a su valor inicial. A continuación el nodo pasa al estado *Reset-Communication*.
- *Reset-Communication*: los parámetros del perfil de comunicaciones sonfijados a su valor inicial. A continuación el nodo pasa al estado *Initialising*.

- ***Pre-operational***: en este estado el dispositivo puede ser configurado mediante SDOs. Puede enviar y recibir SDOs, mensajes de emergencia, de sincronización, *time stamp* y mensajes NMT.

- ***Operational***: el dispositivo ya ha sido configurado y funciona normalmente. Todos los objetos de comunicación están activos así también puede enviar y recibir PDOs.

- ***Stopped***: todos los objetos de comunicación dejan de estar activos. No se pueden enviar ni recibir PDOs ni SDOs, sólo mensajes de NMT.

En la figura 15 se puede ver en detalle el diagrama de los posibles estados de un nodo.



Las letras entre paréntesis indican qué objetos de comunicación están permitidos en cada estado:

a. NMT, b. Node Guard, c. SDO, d. Emergency, e. PDO, f. Boot-up

Transiciones entre estados (mensajes NMT):

1: Start_Remote_Node (command specifier 0x01)
2: Stop_Remote_Node (command specifier 0x02)
3: Enter_Pre-Operational_State (command specifier 0x80)
4: Reset_Node (command specifier 0x81)
5: Reset_Communication (command specifier 0x82)
6: Inicialización terminada, entra en Pre-Operational directamente al mandar el mensaje de Boot-up

Figura 15. Diagrama de transición de estados de un nodo en CANopen

Sólo el NMT maestro puede mandar mensajes del módulo de control NMT, pero todos los esclavos deben dan soporte a los servicios del módulo de control NMT. No hay respuesta para un mensaje NMT. Sólo los envía el maestro y la estructura viene definida en la tabla 17.

Tabla 17. Estructura de un mensaje NMT

COB-ID	Byte 0	Byte 1
0x000	*CS*	Node-ID

Con el Node-ID (identificador de nodo) igual a cero, todos los NMT esclavos son diseccionados (*broadcast*). En la tabla 18 se pueden observar los valores que puede tener el campo CS (*Command Specifier*).

Tabla 18. Valores del campo CS del mensaje NMT

Command Specifier	NMT Service
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-operational State
129	Reset Node
130	Reset Communication

Capítulo 3

Descripción del diseño

En este capítulo se tratarán en detalle todas las especificaciones del sensor que se va a utilizar, la selección de componentes y los pasos seguidos para crear el diseño del sistema encargado del tratamiento y transmisión de datos. En la figura 16 se muestra el esquema inicialmente planteado para llevar a cabo su implementación.

Figura 16. Esquema inicial del diseño

3.1 Selección de componentes

El componente principal y más importante del diseño del dispositivo es el sensor comercial que se utilice. Este elemento va a influir en todos los parámetros relacionados con la adquisición de datos, como pueden ser el protocolo de comunicación, velocidad de transferencia, longitud del paquete de datos, etc. Muy a tener en cuenta es el modo de lectura de los datos que proporcione dicho sensor y la posibilidad de conexión con los demás elementos que formen, como podría ser, un nodo CANopen.

Una vez llevada a cabo la elección del sensor que se va a utilizar, el siguiente paso es seleccionar los distintos componentes que mejor se adapten a las necesidades del diseño. Estos elementos, que posteriormente se tratarán en detalle son:

- Controlador de la comunicación serie
- Microcontrolador
- Controlador CAN-Bus

En lo que se refiere a la lectura del sensor, el 50M31A incorpora una salida serie digital en formato serie síncrono que se rige por el protocolo de comunicaciones serie RS-485, por lo que la comunicación no será el factor más determinante a la hora de elegir el microcontrolador. Sí lo será, por contra, el soporte y herramientas que el fabricante pone a disposición del diseñador, y en este caso **Microchip®** es la mejor opción, ya que además de ofrecer una completa línea de productos para el diseño de aplicaciones embebidas usando el protocolo CAN, como lo es la **Application Note AN945** [10], también ofrece un entorno de desarrollo y programación muy potente como es **MPLAB IDE** y la herramienta **MPLAB C Compiler C18** [11], que incorpora una amplia gama de librerías y funciones predefinidas que facilitan y simplifican de manera significativa el desarrollo de la aplicación deseada.

A la hora de seleccionar los controladores, o también llamados *transceivers*, que gestionan el CAN-Bus y la comunicación serie no se encuentran grandes problemas ya que éstos son componentes muy comunes en el mercado, con diferentes fabricantes que ofrecen prestaciones similares. Dentro de las opciones que Microchip pone a disposición del diseñador a través de su línea de productos CAN, para este diseño concreto se ha optado seleccionar un microcontrolador que integre un módulo CAN y unos controladores de comunicaciones por varias razones:

- Se simplifica la parte hardware ya que no es necesario integrar en la placa del diseño un controlador CAN, ya que este viene incorporado en el microcontrolador.
- Se simplifica la parte software ya que no es necesario programar el puerto serie que comunica el microcontrolador con el controlador CAN.
- Se reducen posibles problemas derivados de la inclusión de otro componente en la placa (conexiones, ruidos, etc.), y se reduce el coste total de los componentes.

3.1.1 Sensor de fuerza/par

Dentro de la amplia gama de productos que ofrece JR3® se ha seleccionado el modelo concreto **50M31A-I25**. Además de caracterizarse por su pequeño tamaño y alta precisión, ofrece una gran facilidad para su montaje sobre distintas superficies, y permite una fácil adaptación a nuestro diseño. Este sensor proporciona una lectura de datos digital, una alta frecuencia de muestreo, y un consumo menor que el de otros sensores de fuerza/par del mercado (ver sección 2.2). En la figura 17 se muestra una imagen del modelo seleccionado.

Figura 17. Sensor comercial de fuerza/par de JR3® modelo 50M31A-I25

Las características eléctricas más importantes son las siguientes:

- Precisión nominal del 1% de Fondo de Escala (FE).
- Linealidad de 0,5% de FE desde +FE a –FE y 0,1% of FE por debajo de 1/4 de FE.
- Electrónica interna para una mayor inmunidad a ruido e interferencias.

Respecto a las propiedades físicas, las más destacables se muestran en detalle en la tabla 19 y 20.

Tabla 19.Propiedades físicas en ejes de fuerza del 50M31A de JR3

	Fx, Fy	Fz
Capacidad de carga (load capacities)[kg]	10,2	20,4
Capacidad de carga (load capacities)[N]	100	200
Rigidez (stiffness)[N/m]	$5{,}07 \cdot 10^6$	$51{,}62 \cdot 10^6$
Sobrecarga permitida (permisible overloads)[N]	471,4	1862,2

Tabla 20. Propiedades físicas en ejes de par del 50M31A de JR3

	Mx, My	Mz
Capacidad de carga (load capacities) [Kg·m]	0,6	0,6
Capacidad de carga (load capacities)[N·m]	5	5
Rigidez (stiffness)[N·m/rad]	$11{,}8 \cdot 10^3$	$2{,}9 \cdot 10^3$
Sobrecarga permitida (permisible overloads)[N·m]	20,31	15,8

Respecto a la orientación de los ejes, los sensores de JR3, tienen orientados los ejes X e Y en el plano del cuerpo del sensor, y el eje Z perpendicular al X e Y. El punto de referencia para todos los datos de carga es el centro geométrico del sensor. Observando desde el punto de vista que ofrece la figura 18 las fuerzas y los momentos se rigen por la regla de la mano derecha.

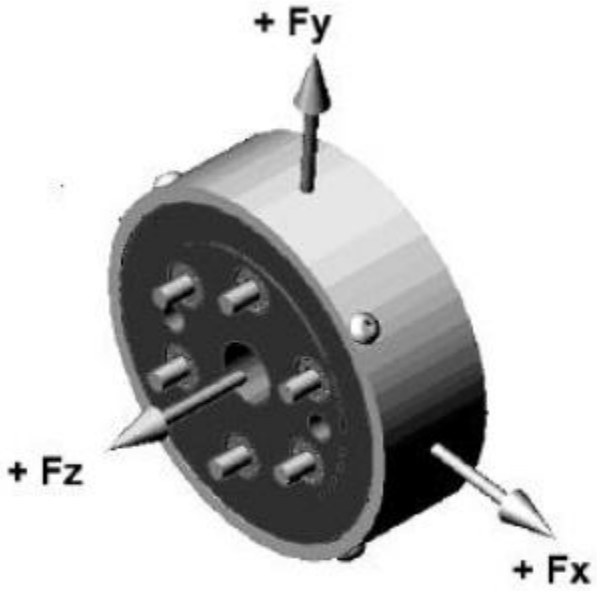


Figura 18. Orientación de los ejes de fuerza/par respecto al cuerpo del sensor

3.1.1.1 Características del sensor de fuerza/par 50M31A de JR3®

Especificaciones

El 50M31A es un miembro de la familia JR3® de transductores de fuerza/par de 6 ejes con electrónica interna. Esta electrónica interna incluye convertidores de tensión y reguladores, amplificadores, un multiplexor, un convertidor analógico-digital (ADC), dos transmisores de par RS-485, una memoria EEPROM de 4k y una PLD (*Programable Logic Device*).

La señal de alimentación del sensor se puede presentar en dos o cuatro líneas. La tensión de alimentación es convertida y filtrada para proporcionar una tensión regulada de +/- 5 V o +/- 10 V. Los amplificadores son usados para amplificar y sumar las señales de tensión calibradas. Las señales sumadas son multiplexadas y alimentan al ADC, el cual digitaliza la señal con una resolución de 14 a 16 bits. La trama de datos de la salida del ADC es serie. El flujo de datos serie es enviado a través de los transmisores RS-485 por el cable externo del sensor para que puedan ser interpretados por el usuario.

Tanto la señal que posee los datos de fuerza/par de los distintos ejes (***DATA***), como la señal digital de reloj de éste (***DCLK***), son transmitidas bajo el protocolo de comunicaciones RS-485. El medio físico es un par de hilos trenzados para mejorar la inmunidad ante posible ruido eléctrico. Las señales de excitación (***EXCITATION***), masa (***GND***), conforman la alimentación del sensor. La tarjeta receptora proporciona la señal de alimentación, la cual debe estar comprendida entre 7 y 9 V.

Paquete de datos

El paquete de datos está formado por 4 bits de direcciones y 16 bits de datos, donde estos últimos están representados en complemento a 2.

La tasa de baudios (*baud rate*), a la que se transmiten las señales *DCLK* y *DATA* es variable. *DCLK* es producido por el transmisor (sensor), y de esta manera la tasa de baudios puede variar durante la utilización del sensor. El sensor asume un tipo de acuerdo según el cual el receptor (microcontrolador) será capaz de aceptar la tasa de datos. Actualmente JR3® diseña receptores capaces de manejar *DCLKs* de hasta 4 MHz.

La longitud del paquete de datos es aproximadamente 24 períodos *DCLK* con el pulso de comienzo (*start pulse*) incluido, como se aprecia en la figura 19. Esto permitiría unos 165k de paquetes por segundo. Este ancho de banda es aceptable

mas allá de lo que actualmente es necesario para las medidas de carga de los 6 ejes.

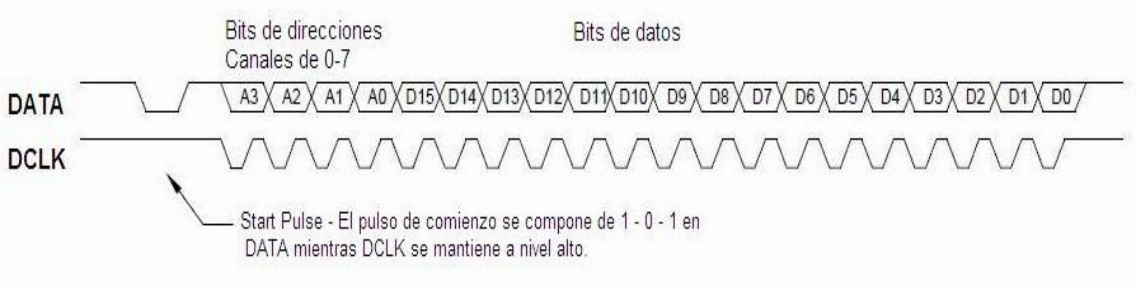


Figura 19. Diagrama de secuencia de transmisión de datos del sensor

De esta forma, se permite trabajar con señales *DCLK* de hasta 4 MHz, pero actualmente se aconseja trabajar con un máximo de 2 MHz, ya que la mayoría de sensores transmiten 7 paquetes a 2 MHz (canales de 0-6) y un octavo a 1 MHz (canal 7). En la figura 20 se observa la señal DCLK emitida por el sensor, que para este modelo concreto posee una frecuencia de 2,2 MHz, cumpliendo así las especificaciones dadas por el fabricante.

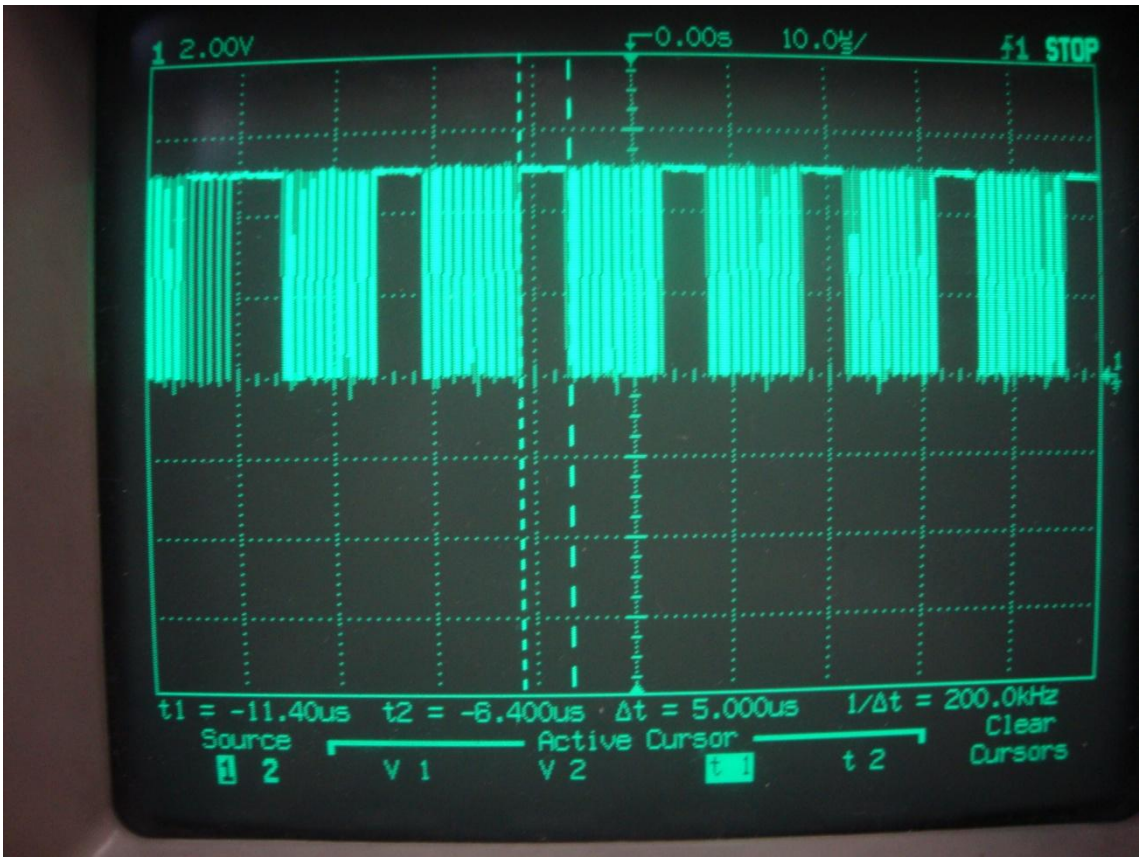


Figura 20. Señal digital de reloj del sensor, DCLK

Contenido del paquete de datos

Como ya se ha comentado, actualmente, la mayoría de estos sensores transmiten 8 canales (0-7), aunque también pueden ser transmitidos 16 canales (0-15). El canal viene especificado por los 4 bits de direcciones del paquete.

El canal 0 contiene la información relativa a la alimentación propia del sensor. Esto permite que este voltaje sea ajustado para adaptar varias longitudes de cable y niveles de voltaje automáticamente. El voltaje de realimentación debe mantenerse al 75% del valor máximo de fondo de escala del ADC (aproximadamente 7,5 V).

Los canales del 1-6 son Fx, Fy, Fz, Mx, My, y Mz respectivamente. Si el paquete de datos se transmite a 64 KHz y 8 paquetes son transmitidos, se puede decir que cada canal posee una frecuencia de transmisión en torno a los 8 KHz. El canal 7 contiene la información de calibración. La mayoría de sensores transmiten este canal a 1 MHz. La información de calibración es guardada en la memoria EEPROM del sensor y transmitida como un paquete formado por valor/dirección donde la dirección es el byte más significativo y el valor es el byte menos significativo.

Los canales del 8-15 están actualmente reservados y solo son transmitidos en dispositivos concretos. Si el paquete de datos se transmite a 64 KHz también en este caso, y 16 paquetes son transmitidos, se puede decir que cada canal posee una frecuencia de trasmisión en torno a los 4 KHz.

En la tabla 21 se observa un cuadro resumen con los bits de direcciones asociados a los canales de correspondientes, para el modelo 50M31A sobre el que se va a trabajar en este diseño y que posee 7 canales de información.

Tabla 21. Bits de direcciones asociados al canal correspondiente

Bits de direcciones	Canal
0000	Alimentación del sensor
0001	Fx
0010	Fy
0011	Fz
0100	Mx
0101	My
0110	Mz
0111	Calibración del sensor

Comunicación externa

La comunicación externa se realiza a través de un cable que en el caso del modelo 50M31A, como la mayoría de sensores digitales de JR3®, es de tipo modular plano con un conector RJ-11(6 pines) en su extremo para una fácil portabilidad a cualquier tipo de diseño.

Estos sensores digitales utilizan un cable modular estándar (standard), en lugar de cable opuesto (reverse). Los términos "standard" y "reverse" son fuente de gran confusión en este tipo de conexionados, por este motivo la figura 21 muestra la asignación de cada cable para las dos terminaciones del cable modular de 6 conexiones.

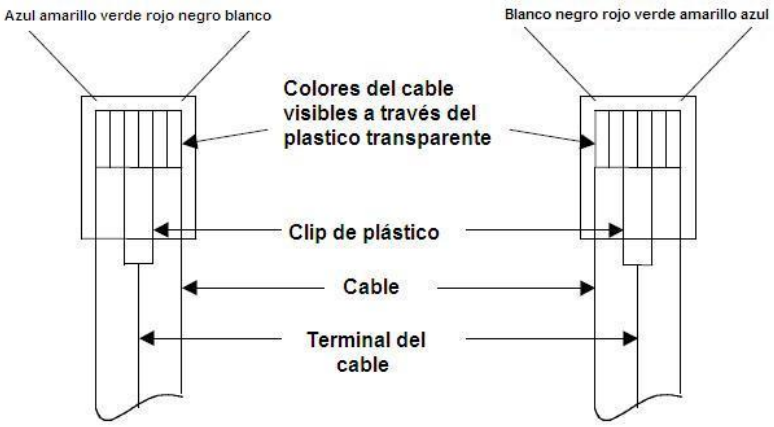
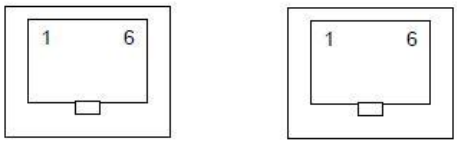


Figura 21. Cable del sensor de 6 pines

En la figura 22 se muestra el patillaje correspondiente al conector del sensor y del módulo receptor.



Conector RJ-11 de 6 pines del sensor

Conector RJ-11 de 6 pines del receptor

PIN	SIGNAL
1	DCLK +
2	DCLK -
3	+ ~ 8 V
4	PWR COM
5	DATA +
6	DATA -

PIN	SIGNAL
1	DATA -
2	DATA +
3	PWR COM
4	+ ~ 8 V
5	DCLK -
6	DCLK +

Figura 22. Patillaje del conector del sensor y del módulo receptor

Matriz de calibración

Todos los sensores multi-eje poseen algún grado de acoplamiento, fenómeno que provoca que la fuerza o par de un determinado eje produzca un cambio sobre la fuerza o par de otros ejes. Cada sensor de JR3® está individualmente calibrado, con determinadas cargas aplicadas a cada eje. Los datos de calibración generan una matriz de calibración y desacoplo, la cual es usada para convertir voltajes de salida en datos de fuerza y par de carga.

En los sensores equipados con salida digital de datos, como es el caso del modelo utilizado, la matriz de calibración se guarda en una memoria no volátil. Cuando el sensor es conectado a la tarjeta receptora esta información es automáticamente descargada en los primeros instantes de operación. El receptor aplica entonces, la matriz al flujo de datos del sensor sin la intervención del usuario.

3.1.1.2 Método de medición del sensor de fuerza/par JR3®

El sistema de medición de fuerza está formado por:

1. Dispositivo de detección (sensor). Como se puede apreciar en la figura 23, éste se compone de anillos elásticos en el perímetro externo de las placas de montaje. El uso de anillos elásticos proporciona una gran rigidez al dispositivo, lo que deriva en una desviación mínima bajo carga y un mejor rendimiento en frecuencias altas. Los anillos y sus medidores de deformación se colocan de manera que las medidas de tensión puedan ser utilizadas para deducir las fuerzas y momentos en todas las direcciones cartesianas (X, Y, Z), que pasan por el sensor. La cavidad interna entre las placas de montaje contiene la electrónica interna donde las señales son amplificadas, digitalizadas y transmitidas a la placa receptora.

Figura 23.Vista superior de un sensor de fuerza/par JR3®

2. Tarjeta de adquisición de datos. Para tratar de forma adecuada los datos que provienen del sensor, el fabricante proporciona una tarjeta receptora con conexión PCI. Esta tarjeta dispone de un dispositivo programable del tipo CPLD (*Programmable Logic Device*) el cual se encarga de separar el paquete de 20 bits en 4 bits de direcciones y 16 bits de datos para que estos sean tratados por un mínimo de dos DSP (*Digital Signal Processors*) a los que el usuario pueda acceder a través del controlador de la tarjeta para obtener la información deseada. En la figura 24 se puede ver una vista esquemática de este proceso.

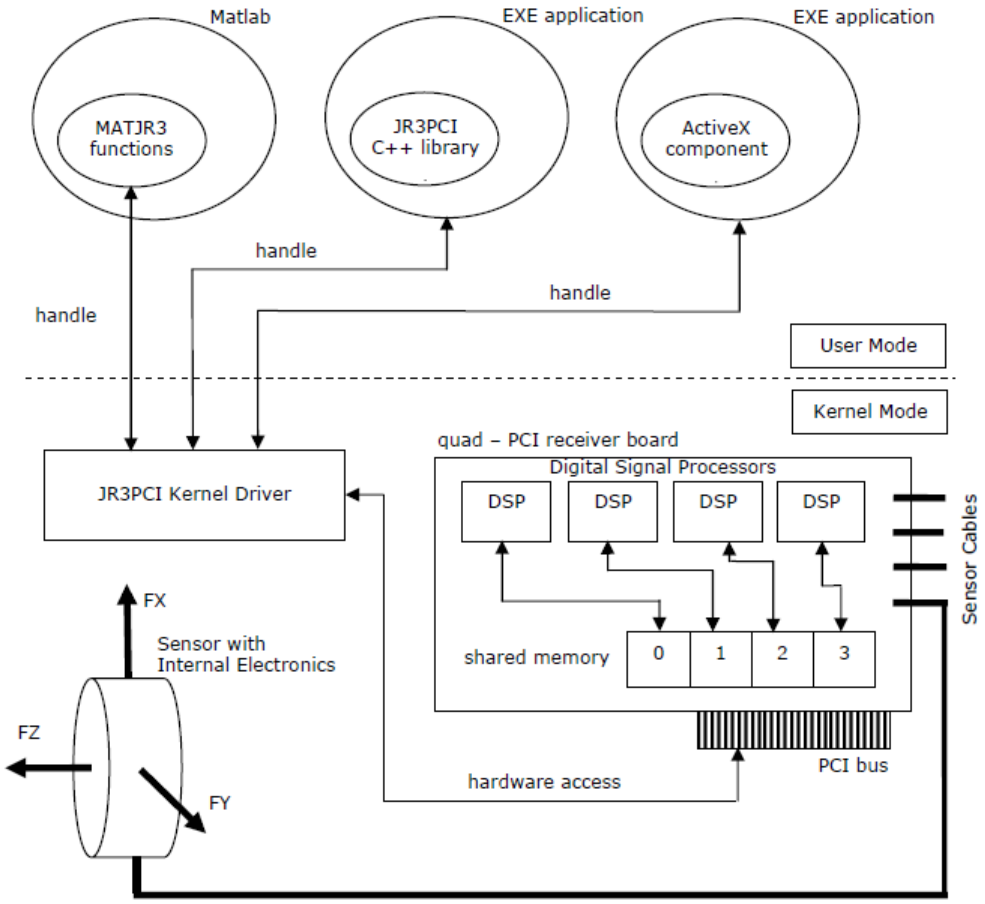


Figura 24. Vista esquemática del sensor conectado a una tarjeta receptora PCI

El fabricante de estas tarjetas proporciona un software específico para el tratamiento de los datos incluyendo una interfaz gráfica, además de distintas funciones y librerías para trabajar con entornos de desarrollo como MATLAB® y LABVIEW®. Para este diseño, trabajar con este tipo de tarjetas es un claro inconveniente, ya que de esta manera no se cumpliría ninguno de los requisitos de diseño referentes a la portabilidad, flexibilidad y protocolo de comunicaciones CANopen expuestos hasta ahora. En la figura 25 se puede ver una imagen real de la tarjeta receptora.

Figura 25. Tarjeta receptora de JR3® con conexión PCI

3.1.2 Transceiver MAX485 de Maxim®

Para realizar la comunicación entre el sensor y el microcontrolador y sabiendo que el sensor trabaja bajo la norma RS-485 [12], se va a utilizar un circuito integrado especialmente diseñado para ello, como es el **MAX485**.

RS-485 es una norma de comunicación serie que utiliza dos líneas (A y B) de manera que la tensión diferencial (sin tierra absoluta) entre ambas marca el nivel lógico que se está enviando. La transferencia es *half-dúplex* (al trabajar con 2 hilos solamente) ya que sólo es posible que un equipo envíe información gobernando las líneas de datos (A y B) y otro u otros equipos reciban. Está pensada para una comunicación multipunto. El medio físico es un par de hilos trenzados entre sí para reducir el posible ruido electromagnético inducido. El estándar RS-485 permite la interconexión de hasta 32 dispositivos sobre un único par de hilos, con velocidades de hasta 10Mbits por segundo y una distancia máxima de 1200 metros. Ambas magnitudes, velocidad y distancia están ligadas entre sí, de manera que si se aumenta una, se reduce la otra. En la tabla 22 se pueden observar estas características.

Tabla 22. Características del dispositivo MAX485

HALF/FULL DUPLEX	TASA DE DATOS (Mbps)	APAGADO DE BAJA ENERGÍA	ENABLE ENVIAR/ RECIBIR	CORRIENTE INACTIVA(µA)	NÚMERO DE TRANSMISORES EN EL BUS	LIMITACIÓN DE SLEW-RATE	NÚMERO DE PINES
HALF	2.5	NO	SI	300	32	NO	8

Al tratarse de un estándar relativamente abierto permite muchas y muy diferentes configuraciones y utilidades. Esta norma sólo especifica características eléctricas de una unidad, pero no especifica o recomienda ningún protocolo de datos como tampoco algún conector especifico.

Los circuitos integrados que manejan esta norma además pueden soportar "colisiones", es decir que más de un circuito transmisor esté emitiendo. Al transmitir en modo diferencial, si el terminal A está a una tensión superior a B (con un valor diferencial superior a 0,2V) se estará recibiendo un "1" y en caso contrario (tensión de B superior en más de 0,2V a A) se interpreta un "0". En la figura 26 se muestra el patillaje del integrado con encapsulado DIP (*Dual In-line Package*) y la correspondiente descripción de los pines en la tabla 23.

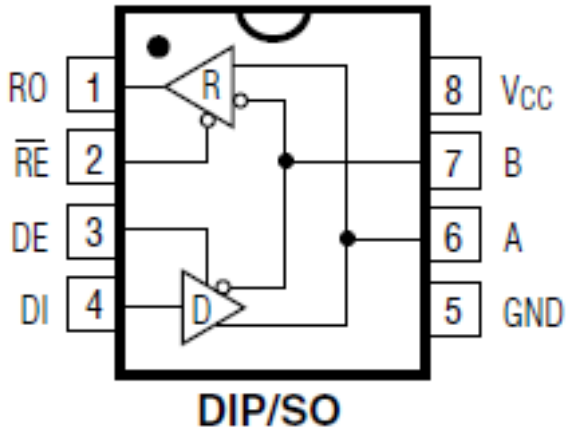


Figura 26. Patillaje del transceiver MAX485

Tabla 23.Descripción de pines del transceiver MAX485

Nº DE PIN	NOMBRE	DESCRIPCIÓN
1	RO	Salida de recepción
2	RE	Enable para habilitar recepción. Activo a nivel bajo.
3	DE	Enable para habilitar transmisión. Activo a nivel alto.
4	DI	Entrada de transmisión.
5	GND	Conexión a masa.
6	A	Entrada recepción/salida transmisión no invertida.
7	B	Entrada recepción/salida transmisión invertida.
8	VCC	Conexión de alimentación.(4.75V≤Vcc≤5.25V)

Para nuestro diseño, se usarán dos integrados configurados en modo receptor. El primero de ellos transportará la señal de reloj del sensor (DCLK), hasta el microcontrolador para así sincronizar de manera adecuada la comunicación serie entre ambos (pin 14). El segundo integrado será utilizado para transportar el paquete de datos con la información de las medidas de fuerza/par del sensor (DATA) hasta el pin habilitado para la recepción serie de datos del micro (pin 15).

Aunque para este diseño solo se va a utilizar la configuración en modo de recepción, a continuación también se especifica las conexiones a realizar para utilizar este tipo de transceiver en modo de transmisión, por si sirviera de ayuda en otras posibles implementaciones. Para realizar la transmisión se necesita

controlar el terminal DE (habilita envío) y RE (habilita recepción). Como presentan una lógica opuesta se pueden unir para ser controlados por una línea solamente. Al transmitir información hay que colocar el pin 2 y el 3 en nivel alto (+5V), de esta forma se habilita el integrado para enviar información por el pin 4 y se bloquea para recibir información por el pin 1, en caso contrario, para recibir información hay que colocar el pin 2 y el 3 en nivel bajo, en este caso se bloquea el envío de información por el pin 4 y se activa la recepción de información por el pin 1. En las tablas 24 y 25 se muestran la configuración de los pines para estos modos de funcionamiento descritos, y en la figura 37 (apartado 3.2.2) se muestran las conexiones realizadas para este diseño.

Tabla 24.Tabla de funcionamiento del MAX485 en modo de transmisión

ENTRADAS			SALIDAS	
RE	DE	DI	B	A
X	1	1	0	1
X	1	0	1	0
0	0	X	ALTA IMP.	ALTA IMP.
1	0	X	ALTA IMP.*	ALTA IMP.*

Tabla 25. Tabla de funcionamiento del MAX485 en modo de recepción

ENTRADAS			SALIDA
RE	DE	A-B	RO
X	1	≥+0.2V	1
X	1	≤-0.2	0
0	0	Entrada abierta	1
1	0	X	ALTA IMP.*

Por último se colocará una resistencia de terminal de 120 Ω (R_t) a través de las líneas A y B en cada extremo de la línea, para proveer a los dispositivos de una adaptación de impedancia al medio de transmisión.

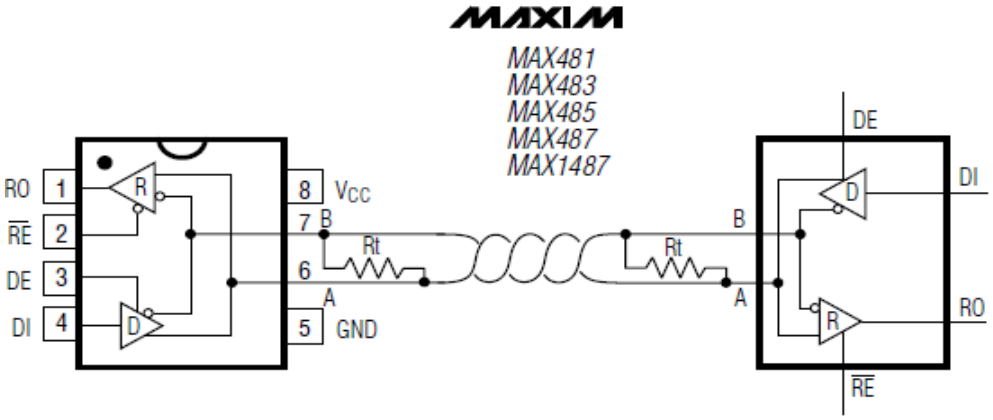


Figura 27. Circuito típico de operación del MAX485

3.1.3 Microcontrolador PIC18F2580

Las principales características del microcontrolador PIC18F2580 están resumidas en la tabla 26 en el que cabe destacar el módulo **ECAN** y el puerto de comunicaciones serie **MSSP** que a continuación se describirán más detalladamente. Además este microcontrolador incluye 4 *TIMER*, de los cuáles uno será utilizado para controlar ciertas características temporales de los Nodos CANopen, y otro que se utilizará para tareas relacionadas con la adquisición de datos. Posee una memoria de programa de 32K, que permite hasta un máximo de 16384 instrucciones simples de programa.

Tabla 26.Características del microcontrolador P18F2580

CARACTERÍSTICAS	PIC18F2580
Frecuencia de operación	DC – 40 MHz
Memoria de programa (Bytes)	32768
Memoria de programa (Instrucciones)	16384
Memoria de datos (Bytes)	1536
Memoria de datos EEPROM (Bytes)	256
Fuentes de interrupción	19
Puertos de E/S	Puertos A, B, C, (E)
Timers	4
Módulos Captura/Compara/PWM	1
Módulos Captura/Compara/PWM Mejorados	0
Modulo ECAN	1
Comunicaciones serie	MSSP, EUSART

Comunicaciones paralelo(PSP)	No
Convertidor ADC de 10 bits	8 Canales de entrada
Comparadores	0
Resets y retardos	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (opcional), WDT
Detección de programación a nivel alto/bajo	Si
Reset de Brown out programable	Si
Conjunto de instrucciones	75 instrucciones; 83 con el Set de instrucción extendido habilitado
Encapsulados	28-pin SPDIP, 28-pin SOIC ,28-pin QFN
Máxima corriente que circula por los pines (mA)	25

3.1.3.1 Módulo ECAN

Las características que posee el **módulo ECAN** del microcontrolador son las siguientes:

- Permite una tasa de transferencia de hasta 1Mbps.
- Cumple la normativa 2.0B según las especificaciones dadas por Bosch.
- Tres modos de operación (*Legacy* (0), *Enhanced Legacy* (1), *FIFO* (2)).
- Tres buffers dedicados a transmisión.
- Dos buffers dedicados a recepción.
- Seis buffers programables para transmisión ó recepción.
- Tres máscaras de recepción de 29 bits.
- Dieciséis filtros de recepción de 29 bits asociables dinámicamente.
- Tratamiento automático de Tramas Remotas.
- Características de tratamiento de error avanzado.
- Longitud de datos de 0-8 bytes.

- Tasa de bits programable hasta 1Mbit/seg.
- Fuente de reloj programable
- Modo de bajo consumo (*Sleep Mode*).

3.1.3.2 Módulo de comunicaciones serie MSSP

El MSSP (Puerto serie síncrono maestro - *Master Synchronous Serial Port*) es un módulo muy útil, y a la vez uno de los circuitos más complejos dentro del microcontrolador. Este módulo permite la comunicación de alta velocidad entre un microcontrolador y otros periféricos u otros microcontroladores al utilizar varias líneas de E/S (como máximo dos o tres líneas). Por eso, se utiliza con frecuencia para conectar el microcontrolador a los visualizadores LCD, los convertidores A/D, las memorias EEPROM seriales, los registros de desplazamiento etc. La característica principal de este tipo de comunicación es que es síncrona y adecuada para ser utilizada en sistemas con un sólo maestro y uno o más esclavos. Un dispositivo maestro contiene un circuito para generación de baudios y además, suministra señales de reloj a todos los dispositivos del sistema. Los dispositivos esclavos no disponen de un circuito interno para generación de señales de reloj. El módulo MSSP puede funcionar en uno de dos modos:

- Modo SPI (Interfaz periférica serie - *Serial Peripheral Interface*).
- Modo I^2C (Circuito inter-integrado - *Inter-Integrated Circuit*).

El modo seleccionado para este diseño es el SPI-esclavo, del que se tratará su configuración software en el capítulo 3.3.2.

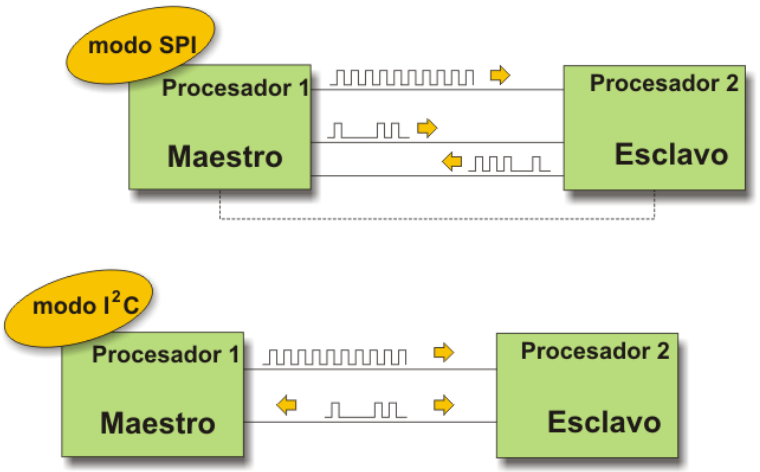


Figura 28. Modos de funcionamiento del MSSP

3.1.3.3 Módulo CCP

El módulo CCP (*Captura/Comparación/PWM*) es un periférico interno que permite medir y controlar diferentes eventos.

El **modo de captura** proporciona el acceso al estado actual de un registro que cambia su valor constantemente. Para el modelo 18F2580, son los registros de los temporizadores Timer1 y Timer3.

El **modo de comparación** compara constantemente valores de dos registros. Uno de ellos es el registro del temporizador Timer1. Este circuito también le permite al usuario activar un evento externo después de que haya expirado una cantidad de tiempo predeterminada.

PWM (*Pulse Width Modulation* - modulación por ancho de pulsos) puede generar señales de frecuencia y de ciclo de trabajo variados por uno o más pines de salida.

El microcontrolador PIC18f2580 dispone de un módulo CCP (CCP1), que para este diseño se utilizará en modo de captura. Tanto la configuración de este puerto como el tratamiento que se le ha dado para este proyecto se detallarán en el capítulo 3.3.3.

3.1.4 Transceiver MCP2551

Este componente tiene como principales características soportar la tasa de transferencia máxima de 1 Mbps que se especifica en el protocolo CANopen, implementar los requerimientos de la capa física ISO-11898, es capaz de detectar fallos en la tierra y permite conectar un máximo de 112 Nodos al Bus. En las figuras 29 y 30 se define el patillaje y funcionalidad de los mismos.

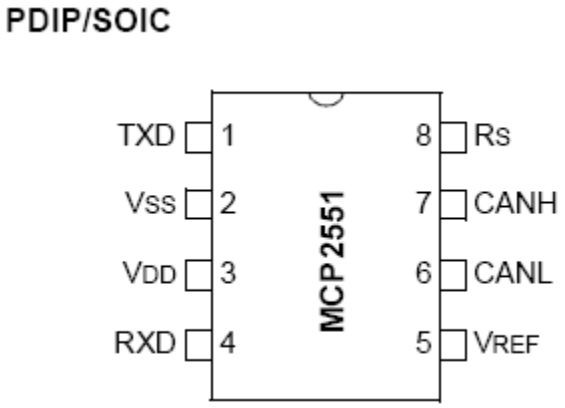


Figura 29. Patillaje del MCP2551

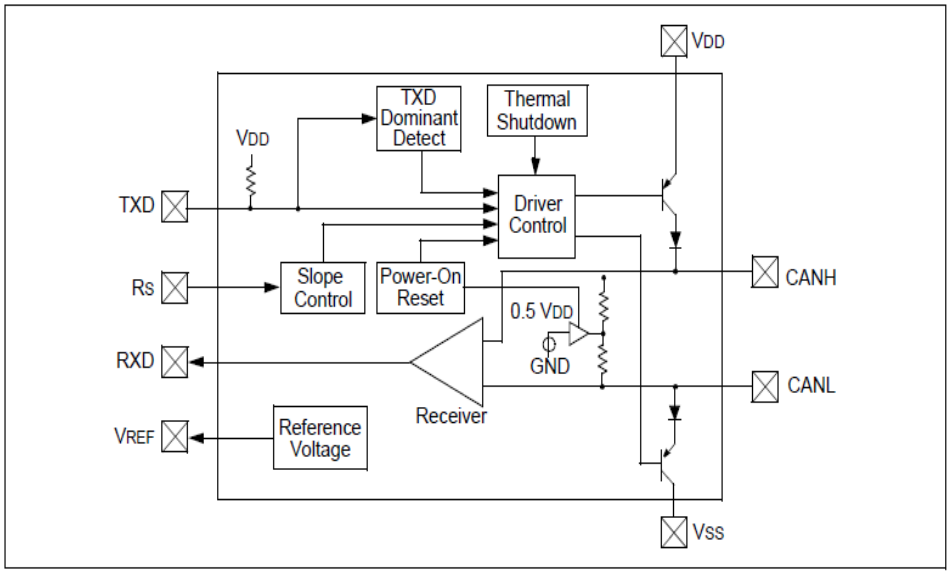


Figura 30.Diagrama de bloques del MCP2551

En la tabla 27 se muestra la descripción de cada pin.

Tabla 27. Descripción de los pines del MCP2551

Nº de pin	Nombre del pin	Función del pin
1	TxD	Entrada de transmisión de datos.
2	V_{SS}	Tensión de masa.
3	V_{DD}	Tensión de alimentación.
4	RxD	Salida de recepción de datos.
5	V_{ref}	Voltaje de referencia.
6	CANL	Nivel bajo de la señal CAN.
7	CANH	Nivel alto de la señal CAN.
8	R_s	Resistencia de control de modo.

La descripción detallada de cada pin se muestra a continuación:

- **Pin 1: TxD:** Entrada digital que recibe los datos que van a ser enviados por el Bus.
- **Pin 2: V_{SS}**: Referencia de tierra.
- **Pin 3: V_{DD}**: Tensión de alimentación (+5V).
- **Pin 4: RxD**: Señal digital que transmite los datos recibidos por el bus.

- **Pin 5: Vref**: Tensión de referencia a la salida definida como $V_{dd}/2$.
- **Pin 6: CANL**: Parte baja de la señal diferencial del Bus.
- **Pin 7: CANH**: Parte alta de la señal diferencial del Bus.
- **Pin 8: R_S**: Este pin permite seleccionar distintos modos de operación (*High Speed*, *Slope Control*, *Standby*) mediante una resistencia externa.

El transceiver es capaz de operar en tres modos, dependiendo de la tasa de transferencia deseada y el nivel de emisiones **EMI** (Emisiones ElectroMagnéticas) permitidas en el sistema. La tasa máxima de transferencia es controlada por medio de una resistencia externa (R_S) que permite modificar los tiempos de la pendiente de transición (***slew rate***) de las señales CANH y CANL. Mientras que las emisiones EMI son controladas por dicha pendiente y mediante un regulador interno de las señales CANH y CANL que proporciona un control simétrico entre ambas señales.

Los modos de operación son los siguientes:

- **High Speed**: Las pendientes de transición son lo más rápidas posibles para conseguir tasas de transferencias muy altas. Este modo se selecciona conectando directamente el pin R_S a masa.
- **Slope Control**: Reduce aun más las emisiones EMI, controlando el *slew rate* de las señales CANH y CANL, a costa de limitar la velocidad máxima de transferencia. La pendiente de la señal será proporcional a la intensidad que circule por el pin R_S. Este modo se selecciona conectando una resistencia (R_S) entre el pin R_S y masa. En la figura 31 se ilustran los valores típicos de *slew rate* en función del valor de la resistencia externa.

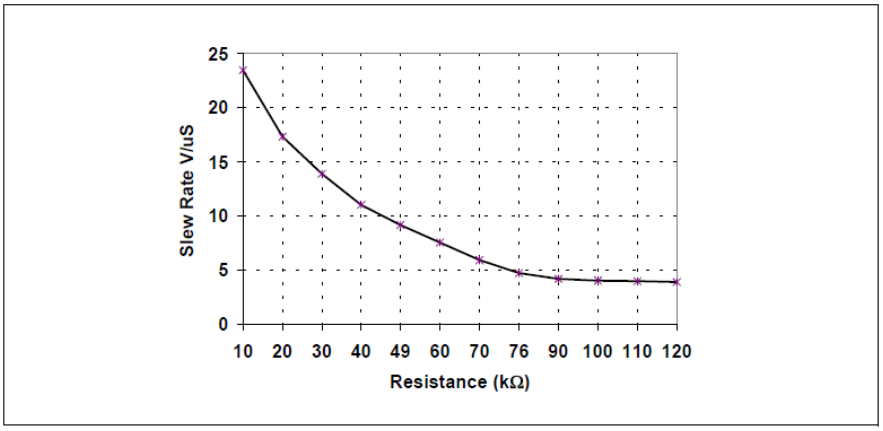


Figura 31. Valores de slew-rate en función de Rs

- **Standby**: Permite introducir al dispositivo en modo *standby* o *sleep* cuando el valor de R_S es muy elevado. En este modo se desactiva la transmisión,

mientras que la recepción sigue operativa consumiendo muy poca intensidad. El microcontrolador puede seguir monitorizando el bus y cambiar al modo de operación normal cuando detecte actividad. El primer mensaje puede perderse cuando la tasa de transferencia es elevada.

3.2 Diseño hardware

El diseño del módulo a nivel hardware consiste en la creación de una placa PCB (*Printed Circuit Board,* placa de circuito impreso).

Para el diseño del *Layout* de la placa se han tenido en cuenta la configuración de los distintos dispositivos para que se cumplan las especificaciones del diseño, y finalmente se ha obtenido una placa como la que se puede observar en la figura 32 con unas medidas de 85x65 mm.

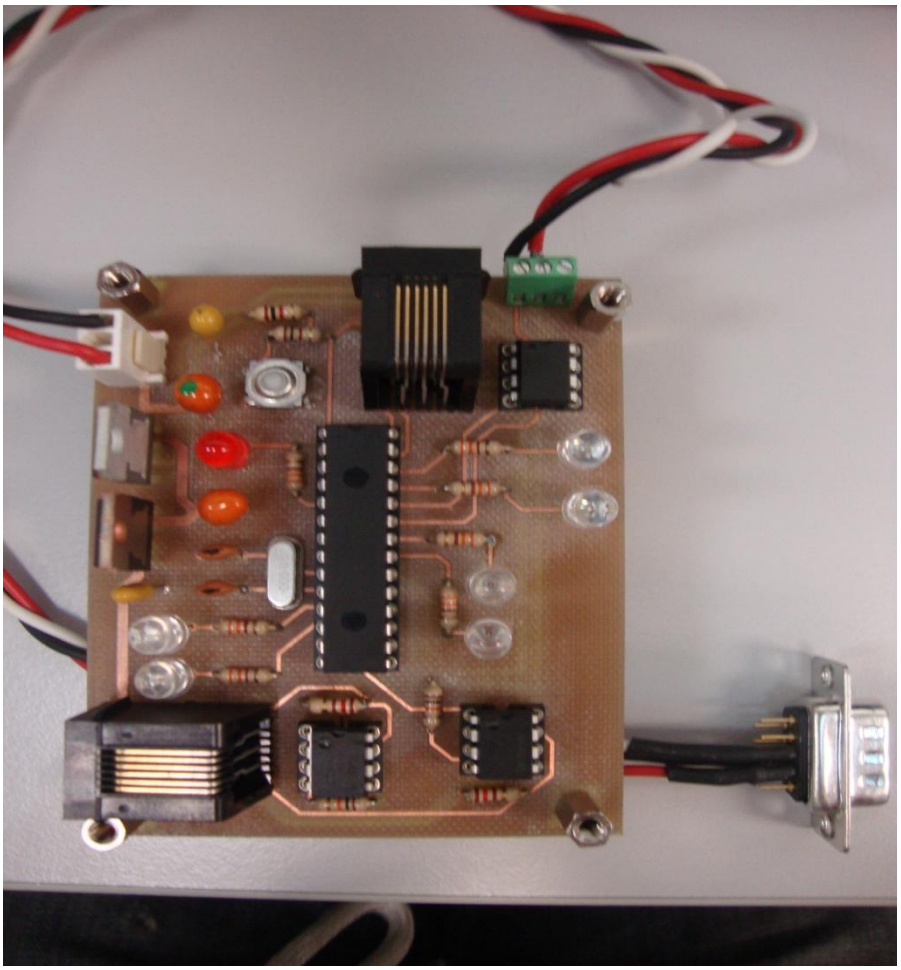


Figura 32. Vista superior de la placa

En cuanto al conexionado del sensor es el siguiente:

- **DATA-** (1) conectado a la señal B (7) del MAX485 (U2) encargado de recibir los datos del sensor.
- **DATA+** (2), conectado a la señal A (6) del MAX485 (U2) encargado de recibir los datos del sensor.
- **GND** (3), conectado a la señal de masa del circuito.
- **EXCITATION** (4), conectado a la salida del regulador de tensión que proporciona 8 V.
- **DCLK-** (5), conectado a la señal B del MAX485 (U3), encargado de recibir la señal de reloj del sensor.
- **DCLK-** (6), conectado a la señal A del MAX485 (U3), encargado de recibir la señal de reloj del sensor.

Los transceivers MAX485 encargados de las comunicaciones serie del diseño se conectarán como se detalla a continuación:

- **U2**, integrado encargado de enviar los datos de lectura del sensor al microcontrolador configurado en modo de recepción de datos:

- **A** (6) y **B** (7) como hemos detallado anteriormente en las conexiones del sensor.
- **DE** (3) y **RE** (2) conectadas a la masa del circuito para habilitar la recepción continua de datos.
- **DI** (4), es indiferente, pero la conectamos a la Vcc del circuito mediante una resistencia de pull-up de 10 KΩ para evitar ruidos o distorsiones indeseables.
- **RO** (1), conectado a la señal SDI (Serial Data Input) (15) del microcontrolador, habilitada para la recepción de datos serie síncrono.

- **U3**, integrado encargado de enviar la señal de reloj del sensor al microcontrolador configurado, también, en modo de recepción de datos:

- **A** (6) y **B** (7) como hemos detallado anteriormente en las conexiones del sensor.

- **DE** (3) y **RE** (2) conectadas a la masa del circuito para habilitar la recepción continua de datos.

- **DI** (4), es indiferente, pero la conectamos a la Vcc del circuito mediante una resistencia de pull-up de 10 KΩ para evitar ruidos o distorsiones indeseables

- **RO** (1), conectado a la patilla (14) del microcontrolador por la cual se está recibiendo la señal DCLK.

Al microcontrolador, además de conectarse lo definido anteriormente, habrá que añadir:

- El circuito de reset, formado por una resistencia de 100 Ω, un pulsador referenciado a la masa del circuito y una resistencia de *pull-up* de 1 KΩ.

- Un cristal de cuarzo de 20MHz con sus respectivos condensadores de desacoplo de 22pF, valores seleccionados conforme a las recomendaciones del fabricante del microcontrolador.

- Siete diodos LED para indicar el estado de lectura de fuerza y par en los tres ejes de coordenadas (x, y, z), y el estado del nodo de comunicaciones CAN.

- Un conector RJ-11 de 6 pines, del que se utilizaran 5 para implementar el circuito de programación del microcontrolador que se describirá en el capitulo 3.2.1. Las señales del microcontrolador necesarias para la programación de éste serán **PGC** (27) y **PGD** (28).

- Las señales **CANRX** (24) y **CANTX** (23) se conectaran a las patillas **RxD** y **TxD** del transceiver MCP2551 respectivamente.

Las conexiones realizadas en el transceiver MCP2551 serán las siguientes:

- Las señales **RXD** (4) y **TXD** (1) según se ha definido anteriormente.

- El pin $\mathbf{R_S}$ (8) se conectara a masa para seleccionar el modo **High Speed**.

- Y las señales **CANL** (6) y **CANH** (7) se conectaran al conector DB9 que permitirá la conexión con otros dispositivos CANopen. En la tabla 28 se muestra el conexionado del conector para la comunicación CAN.

Tabla 28. Conexionado del conector CAN

DB-9	SEÑAL
2	CAN_L
3	GND
7	CAN_H

3.2.1 Circuito de programación

Con el fin de poder programar el microcontrolador PIC sin necesidad de retirarlo (o desoldarlo de la PCB) y utilizar otros circuitos auxiliares se ha decidido introducir un conector RJ-11 de 6 pines que permite tener acceso a las patillas de programación del microcontrolador (PGD y PGC). A este modo de programación se le conoce como programación ICSP (*InCircuit Serial Programming*).

Programar los microcontroladores sin tener que desmontarlos de los circuitos donde están ubicados es una particularidad que tienen casi todos los dispositivos, pero si, además, su memoria de programa es del tipo Flash les confiere aún mayor flexibilidad, pues se pueden realizar cambios en los programas y volver a reprogramar el microcontrolador tantas veces como sea necesario para depurar la aplicación. A la hora de llevar a cabo este tipo de programación/depuración del microcontrolador se usará la herramienta que proporciona el entorno de desarrollo de Microchip, **MPLAB ICD 2** [13]. En la figura 33 se muestra el patillaje que debe tener el conector del ICD 2 para su ensamblado en la placa del diseño.

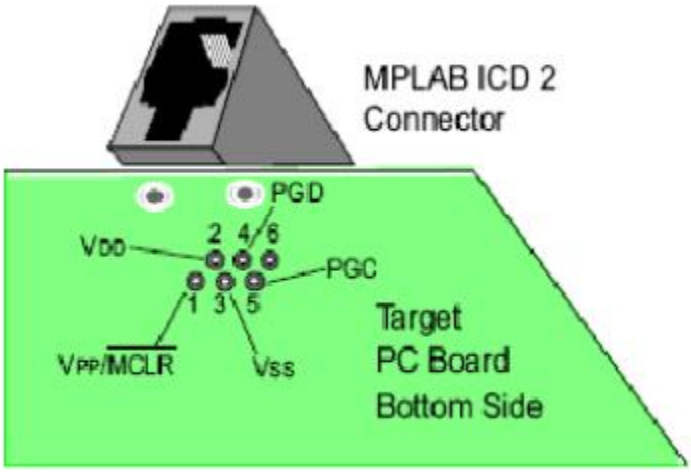


Figura 33. Patillaje del conector ICD 2

En la figura 34 se muestran las conexiones del conector ICD 2 con el dispositivo sobre el que se quiere trabajar, en este caso el microcontrolador 18F2580. Aunque se dispone de seis pines solo será necesario conectar cinco de éstos. Una resistencia de *pull-up* (1KΩ) conectada a la señal MCLR, que se utiliza como reset del PIC como ya se ha especificado en el apartado 3.2. Las señales V_{DD} y V_{SS} se conectan a V_{CC} y GND del microcontrolador respectivamente, y por último las ya comentadas señales PGD y PGC. No todos los microcontroladores poseen las señales AV_{DD} y AV_{SS} pero en caso de tenerlas, deben ser conectadas como indica MPLAB ICD 2.

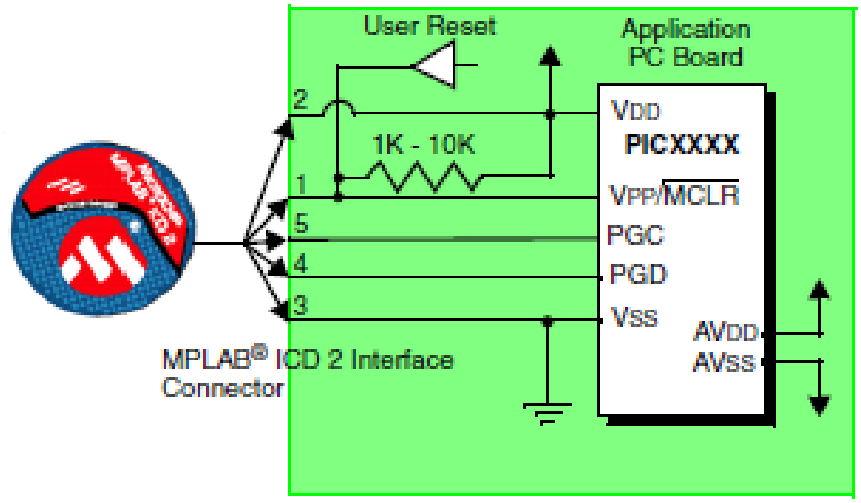


Figura 34. Esquema de conexiones entre el depurador ICD 2 y el PIC

Sin embargo existen unas directrices que proporciona MPLAB para que este modo de operación actúe correctamente. Son las enumeradas a continuación y se pueden observar gráficamente en la figura 35.

- No colocar resistencias de pull-up en PGC y PGD, ya que estas líneas poseen unas resistencias de pull-down de 4,7 KΩ dentro del depurador ICD 2.
- No colocar condensadores en PGC y PGD, ya que estos evitarían rápidas transiciones en las señales de datos y de de reloj durante las operaciones de programación y depuración.
- No colocar condensadores en MCLR, ya que estos evitarían transiciones rápidas a Vpp. Como ya se ha comentado, una resistencia es suficiente.
- No colocar diodos entre PGC y PGD, ya que provocarían que no hubiera comunicación bidireccional entre el dispositivo y el depurador.

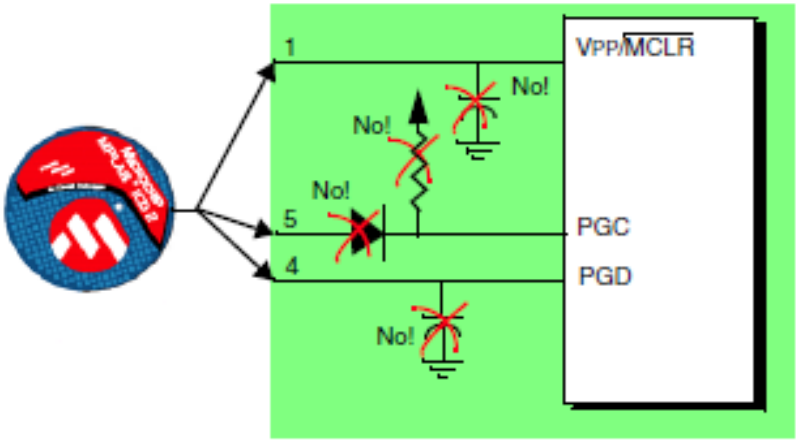


Figura 35. Vista gráfica de los elementos no deseados para el modo de operación ICD2

3.2.1.1 Modo de depuración (Debug mode)

Una vez realizadas todas las conexiones hardware correctamente, existen dos pasos para empezar a usar MPLAB ICD 2. El primero requiere que una aplicación sea programada en el microcontrolador. La segunda utiliza el hardware interno del depurador para ejecutar y probar la aplicación. Los dos pasos están directamente relacionados con operaciones de MPLAB IDE:

1. Programar el código en el dispositivo.
2. Usar la herramientas de depuración que ofrece el entorno de desarrollo para ejecutar el programa, poner *breakpoints*, ejecutar paso a paso, etc. Si el dispositivo no puede ser programado correctamente, MPLAB ICD 2 no activará el modo de depuración.

3.2.1.2 Requisitos para trabajar en Debug mode

Para depurar (poner breakpoints, ver registros, etc.) con MPLAB ICD 2 existen una serie de elementos que deben estar funcionando correctamente:

- MPLAB ICD 2 debe estar conectado al PC mediante USB o RS-232, y además debe estar alimentado a través de la placa donde se encuentra el dispositivo.
- El conexionado para la programación “on chip” debe estar perfectamente configurado como se ha especificado en los apartados anteriores.
- El microcontrolador a programar debe tener un oscilador apropiado, según las características que indique el fabricante del PIC.

- Las palabras de configuración del microcontrolador (*configuration bits*) deben estar configuradas de manera apropiada, como se mostrará en detalle en el apartado 3.3.1.

Una vez hecho esto, el dispositivo ya estará listo para entrar en el modo de depuración. La configuración del microcontrolador quedará como se muestra en la figura 36.

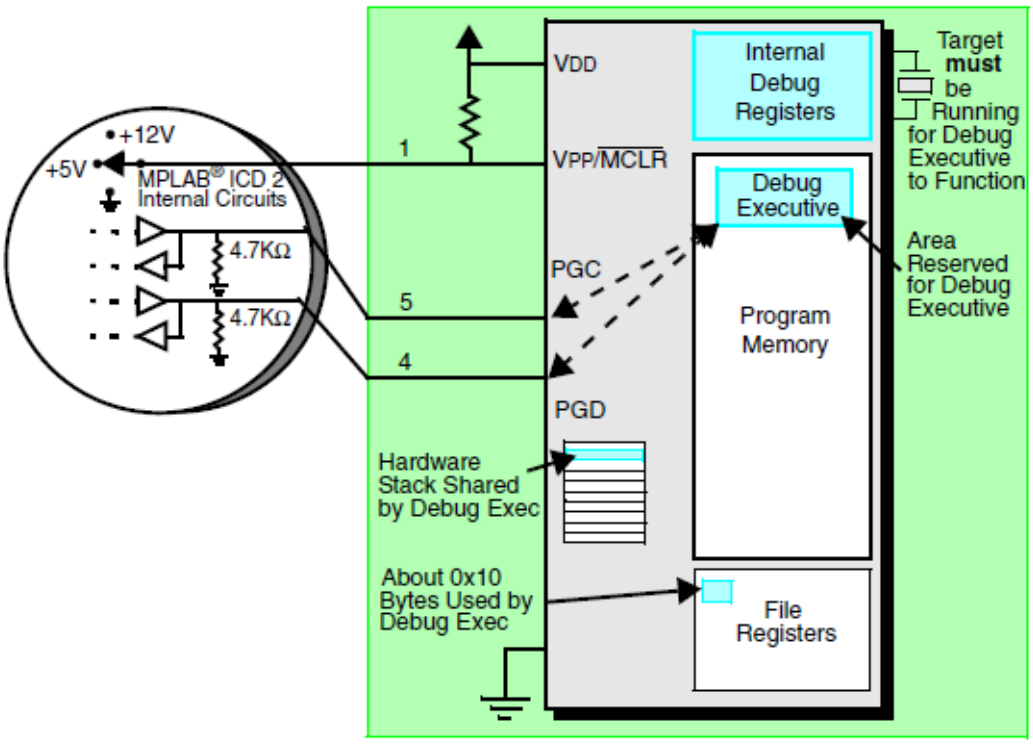


Figura 36. Dispositivo listo para trabajar en modo de depuración

3.2.2 Esquemático final

A continuación, en la figura 37, se muestra el esquemático final, que implementa todos los circuitos diseñados según lo descrito durante este capítulo. La creación de este esquema se ha llevado a cabo con la herramienta dedicada a tal fin, *OrCAD Capture CIS®*, para después diseñar la placa de circuito impreso con *OrCAD Layout Plus®*, teniendo en cuenta las especificaciones de cada componente con el fin de conseguir una placa de tamaño mínimo para que el diseño final sea lo más compacto posible.

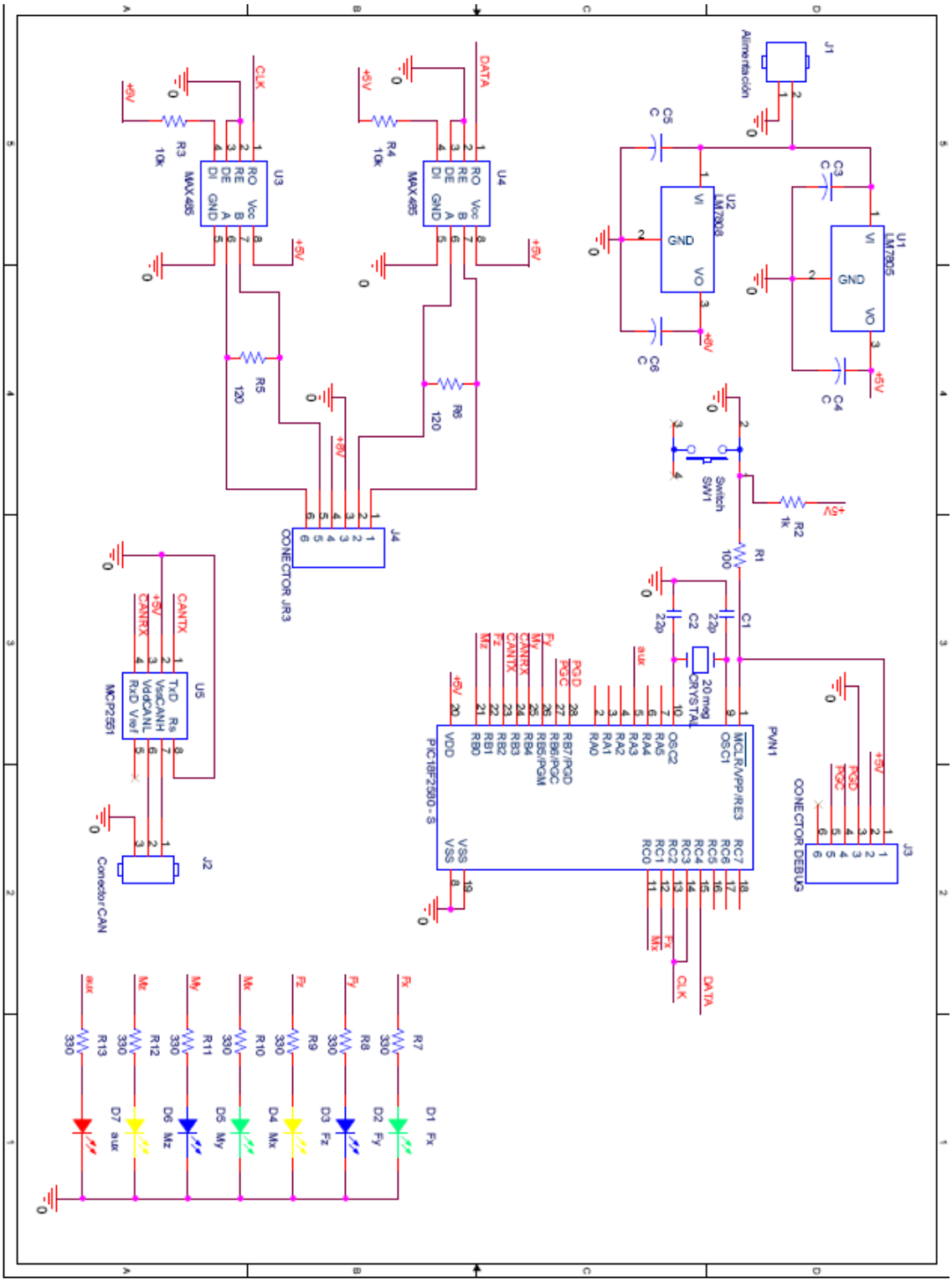


Figura 37. Esquemático final de la placa

En la figura 38 se observa una imagen correspondiente a la capa principal del diseño.

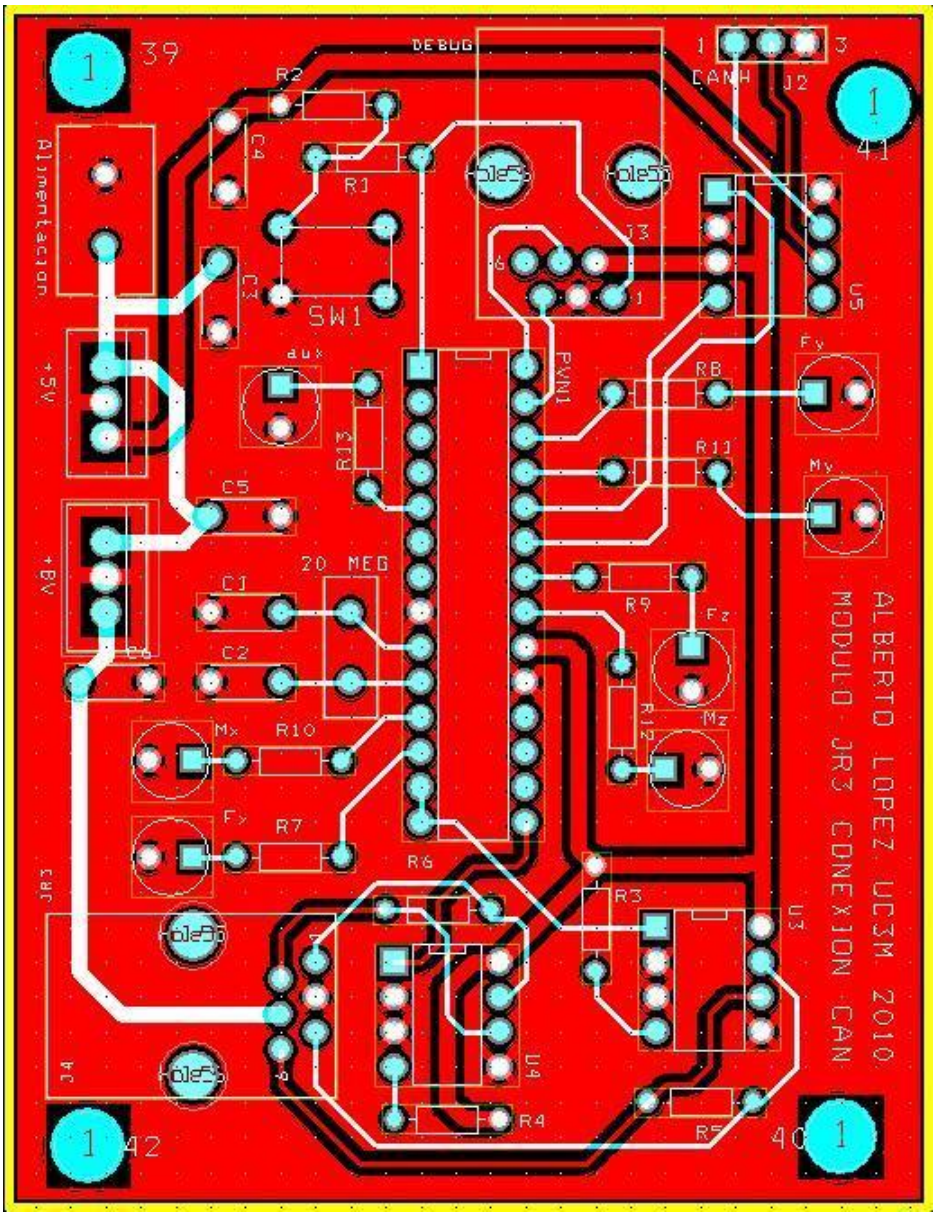


Figura 38. Capa principal del diseño

3.3 Diseño software

El desarrollo software del proyecto comprende toda la programación necesaria del microcontrolador PIC. Esta parte junto con el desarrollo hardware componen en general la estructura principal del proyecto.

El software se ha desarrollado en lenguaje C utilizando el entorno de programación MPLAB y el compilador C18, ambos de Microchip®. Se ha utilizado este lenguaje ya que es muy potente para la programación de microcontroladores y permite hacer más legible el programa, además se simplifica la tarea del programador con respecto a las instrucciones de ensamblador.

El archivo **AppJR3.c** contiene la rutina principal del programa, a través del cual se configuran todos los parámetros iniciales y posteriormente se ejecutan todos los procesos relacionados con la aplicación y comunicación. A continuación se describe el código desarrollado para la aplicación diseñada, que se comentará en los siguientes pasos.

3.3.1 Configuración software del microcontrolador

```
#pragma config OSC =HS ,FCMEN = OFF,IESO = OFF //CONFIG1H
#pragma config PWRT = ON,BOREN = BOHW,BORV = 3 //CONFIG2L
#pragma config WDT = OFF,WDTPS = 32768 //CONFIG2H
#pragma config MCLRE = ON,LPT1OSC = OFF,PBADEN = OFF//CONFIG3H
#pragma config STVREN = ON,LVP = OFF,XINST = OFF,DEBUG = ON //CONFIG4L
#pragma config CP0 = OFF,CP1 = OFF,CP2 = OFF//CONFIG5L
#pragma config CPB = OFF,CPD = OFF//CONFIG5H
#pragma config WRT0 = OFF,WRT1 = OFF,WRT2 = OFF//CONFIG6L
#pragma config WRTB = OFF,WRTC = OFF,WRTD = OFF//CONFIG6H
#pragma config EBTR0 = OFF,EBTR1 = OFF,EBTR2 = OFF,EBTR3 = OFF//CONFIG7L
#pragma config EBTRB = OFF//CONFIG7H
#pragma config BBSIZ = 1024 // 1K words (2K bytes) Boot Block
```

A través de estas líneas se llevan a cabo las instrucciones necesarias para una correcta configuración del microcontrolador 18F2580, en las que se resalta como más importante:

- OSC = HS, mediante la cual se selecciona la velocidad del oscilador, en este caso, de alta velocidad ya que el cristal conectado es de 20 MHz.
- WDT=OFF, el timer del perro guardián (watch dog) desactivado para así poder utilizar el método de programación on-chip.
- DEBUG=ON, para activar o desactivar el modo de depuración.
- MCLRE=ON, para indicar que existe un circuito de reset externo para restablecer todos los valores iniciales del micro en caso de ser necesario.

El resto de instrucciones se utilizan para proteger código, bloques de arranque y de registros contra lectura y escritura.

Una vez configurado el microcontrolador en sí, se procede a la configuración de los distintos pines de éste que se van a utilizar en este diseño.

```
TRISAbits.TRISA3 = 0; //pin conectado al LED configurado como salida

SSPCON1bits.SSPEN=1;//pines del puerto SPI habilitados

TRISCbits.TRISC2 = 0x01; //configuro pin CCP1 como entrada

TRISAbits.TRISA3 = 0; // pin conectado al LED configurado como salida

TRISBbits.TRISB1 = 0; // pin conectado al LED configurado como salida

TRISBbits.TRISB4 = 0; // pin conectado al LED configurado como salida
```

3.3.2 Configuración software de la comunicación serie

El siguiente paso es configurar la lectura del sensor a través del puerto serie SPI, para ello, se utilizará la función void OpenSPI(unsigned char **sync_mode**, unsigned char ***bus_mode***, unsigned char ***smp_phase***) incluida en la librería **spi.h**, encargada de gestionar todas las operaciones que se efectúen con este puerto. El código relacionado es el siguiente:

```
OpenSPI(SLV_SSOFF, MODE_10, SMPEND);
```

Donde:

- SLV_SSOFF: Por un lado se activa el modo SPI esclavo (ya que el dispositivo maestro es el sensor) y por otro lado, se deshabilita el selector de esclavo ya que solo disponemos de un dispositivo (PIC) y no es necesario seleccionar su prioridad.
- MODE_10: La lectura de datos se realiza en cada flanco de bajada de la señal de reloj. El estado de reposo de la señal de reloj se define a nivel alto.
- SMPEND: Indica que el punto de muestreo de los datos respecto al ciclo de reloj es al final de éste.

3.3.3 Tratamiento de la señal de reloj, DCLK

Como se vio anteriormente en el capítulo 3.1.1.1 y gráficamente en la figura 20, la señal DCLK posee un pulso de comienzo, a partir de ahora llamado *start pulse,* de un período aproximado de 4,8 µs. Cuando este pulso termina (flanco de bajada) la señal de datos envía los bits de direcciones correspondientes de una fuerza o par concreto y a continuación los 16 bits datos en sincronía con la señal de 2,2Mhz que se repite durante 10 µs hasta la llegada de un nuevo *start pulse*.

Por tanto para capturar este *start pulse* se ha creado el código siguiente:

```
T3CONbits.T3CCP1=0x10; //se asocia el timer3 al CCP1

OpenTimer3(TIMER_INT_OFF & T3_SOURCE_INT & T3_PS_1_1 );//configuración
timer3

OpenCapture1(CAPTURE_INT_OFF & C1_EVERY_RISE_EDGE);//configuración
capture por flanco de subida

PIR1bits.CCP1IF = 0x00; //Limpiar flag modo capture
CCP1CONbits.CCP1M3=0; //configuración capture por flanco
CCP1CONbits.CCP1M2=1;// de subida
CCP1CONbits.CCP1M1=0;
CCP1CONbits.CCP1M0=1;
while(!PIR1bits.CCP1IF); //esperar a flanco de subida
TMR3L=0x00; //borrar cuenta de Timer3
PIR1bits.CCP1IF = 0x00; // limpiar flag modo capture
CCP1CONbits.CCP1M0=0;//configurar capture por flanco de    bajada
while(!PIR1bits.CCP1IF); //esperar a flanco de bajada

                    ***Rutina de adquisición datos***
```

Nótese que una vez recogido el flanco de subida de la señal a capturar el número de instrucciones es el mínimo posible para ganar rapidez y poder capturar señales con períodos muy pequeños y evitar hacer lecturas de datos de forma errónea. Para ello, por ejemplo, se obvia la llamada a funciones de librerías externas y se escribe directamente en los registros pertinentes.

3.3.4 Rutina de adquisición de datos

Una vez configurado el puerto serie correctamente y realizada la captura de la señal DCLK, se procede a crear la rutina de adquisición de datos. A continuación se enumeran los pasos seguidos para tratar de forma adecuada los datos de fuerza y par que suministra el sensor.

Lo primero será crear las variables donde se almacenarán los datos de fuerza y par correspondientes. Para ello, estas variables han sido declaradas del tipo unión de estructuras para poder tratarlas bit a bit en el caso de que fuera necesario. A continuación se muestra la variable creada para guardar la información correspondiente a la fuerza ejercida sobre el eje x. Las variables para almacenar la información del resto de los ejes se realiza de forma idéntica. El tamaño de estas

variables será de 16 bits (word), ya que como se vió en el apartado 3.1.1.1, éste era el tamaño del paquete datos.

```
 union
{
short word;
struct
{
unsigned B0:1;
unsigned B1:1;
…
unsigned B15:1;

};
}fxbits;
```

Una vez hecho esto, se comprueba si la señal capturada es un *start pulse*, y en caso afirmativo, se irán almacenando en la variable c2JR3 previamente declarada, mediante la llamada a la función ***void getsSPI (char * rdptr, unsigned char length)***. Apuntar que esta función ha sido modificada para guardar los datos en el mismo orden en que se presentan en el sensor y evitar de esta manera posibles problemas con el significado de éstos. Una vez almacenado el paquete de datos, este se transfiere a una variable de 24 bits, del tipo unión de estructuras para que, como se explicaba anteriormente, se pueda dar un trato bit a bit a los datos leídos.

```
unsigned char c2JR3[3];

…

// rutina de lectura

        if (TMR3L>20){      //si  han pasado más de 4,8 µs(start pulse)

                            getsSPI((unsigned char*)&c2JR3[2], 3); //lectura de
                            datos

                            lecturabits.tres_bytes=*(short long*)c2JR3;
```

Por último, se comprueba a que eje y de qué tipo es el dato leído, y se guardará entonces en el registro correspondiente los 16 bits de datos. El código mostrado a continuación correspondiente a la lectura de fuerza en el eje x.

```
if(lecturabits.B23==0 && lecturabits.B22==0 && lecturabits.B21==0 &&
lecturabits.B20==1){ //Si los bits de direcciones indican 0001 es Fx
```

```
fxbits.B0=lecturabits.B4; //se guarda la lectura de fuerza
fxbits.B1=lecturabits.B5;
fxbits.B2=lecturabits.B6;
fxbits.B3=lecturabits.B7;
fxbits.B4=lecturabits.B8;
fxbits.B5=lecturabits.B9;
fxbits.B6=lecturabits.B10;
fxbits.B7=lecturabits.B11;
fxbits.B8=lecturabits.B12;
fxbits.B9=lecturabits.B13;
fxbits.B10=lecturabits.B14;
fxbits.B11=lecturabits.B15;
fxbits.B12=lecturabits.B16;
fxbits.B13=lecturabits.B17;
fxbits.B14=lecturabits.B18;
fxbits.B15=lecturabits.B19;
uJR3force.word=fxbits.word;
}
```

Una vez hecho esto ya tenemos una variable en formato binario (uJR3force.word) para que pueda ser tratada y enviada a través de CANopen.

3.4 Configuración software de la comunicación CAN

Para llevar a cabo la comunicación entre el microcontrolador y el dispositivo o red CAN a la que se desee conectar, es necesario conocer los las características de la pila CANopen que proporciona Microchip® para poder realizar la implementación deseada.

Para ello primero se hará una descripción de la pila y sus características para, a continuación, desarrollar el código y las aplicaciones necesarias para integrar el módulo diseñado en una red CAN.

3.4.1 Descripción de la pila CANopen de Microchip®

La pila CANopen de Microchip® no es simplemente un enfoque genérico de una pequeña aplicación, sino que esta es una pila de comunicación genérica basada en CANopen que puede ser modificada según las necesidades del usuario. Está basada en la documentación estándar DS-301 de CAN in Automation (CiA) [14]. De hecho, la mayoría del código está limitado a implementar áreas de las especificaciones

dadas por CiA, con especial énfasis en dar a conocer al usuario como debe desarrollar una aplicación bajo el protocolo CANopen. Esta pila implementa el desarrollo de una aplicación basada en el perfil de dispositivo DS-401 [15], es decir, un módulo de I/O genérico.

Todo el código suministrado en esta *Application Note* está desarrollado para los PIC18F8680 y PIC18F4680, los cuales incluyen tecnología ECAN, y se recomienda que sean compilados con la versión V2.30 (ó superior) del compilador C18 de Microchip®. Aunque el código esta desarrollado para los microcontroladores mencionados anteriormente, esta pila es fácilmente adaptable a otros microcontroladores de la familia PIC18 que incluya tecnología CAN. Por otro lado, es muy recomendable que el usuario de estos códigos tenga un mínimo de conocimientos sobre sistemas CANopen, y en su defecto que tenga acceso a la documentación estándar CANopen ó lea previamente los capítulos dedicados a CAN-Bus y CANopen de este proyecto (secciones 2.3 y 2.4).

3.4.2 Características principales de la pila CANopen

La pila CANopen implementa las capas bajas del protocolo y permite implementar sobre esta la capa de aplicaciones. Esta pila está dividida en una serie de archivos fuentes y de cabecera más pequeños, escrita todo en C y que permite a los usuarios seleccionar los servicios necesarios y selectivamente construir un proyecto adaptado a las especificaciones de su diseño. Aun así, la aplicación y ciertos aspectos de la comunicación deben ser desarrollados por el usuario. En la tabla 29 se muestra una lista completa de los archivos fuente.

Tabla 29. Ficheros incluidos en la pila CANopen

File Name	Descripción
CO_CANDRV.c CO_CANDRV.h	Driver de configuración del módulo ECAN. Pueden ser sustituidos por los de otros dispositivos si es necesario.
CO_COMM.c CO_COMM.h	Servicios de gestión de las comunicaciones. Requerido para todas las aplicaciones.
CO_DEV.c CO_DEV.h	Archivos específicos de cada dispositivo. Deben ser editados por el Usuario
CO_DICT.c CO_DICT.h CO_DICT.def	Estructura del Diccionario de Objetos. Requeridos para todas las aplicaciones.
CO_MAIN.c CO_MAIN.h	Servicios principales de CANopen. Requeridos para todas las aplicaciones
CO_MEMIO.c CO_MEMIO.h	Funciones de copiado de memoria usadas por el Diccionario. Requeridos para todas las aplicaciones
CO_NMT.c CO_NMT.h	Gestión de los endpoint de comunicaciones del Bus.
CO_NMTE.c CO_NMTE.h	Endpoint de comunicación para los servicios Node Guard, Heartbeat y Boot-up.
CO_PDO.c CO_PDO.h	Servicios PDO generales.
CO_PDO1.c CO_PDO1.h CO_PDO2.c CO_PDO2.h CO_PDO3.c CO_PDO3.h CO_PDO4.c CO_PDO4.h	Endpoint de tratamiento de Objetos PDO. Proporciona una plantilla que requiere ser desarrollada por el usuario para cada aplicación en concreto. Deben ser utilizados con los archivos de servicios PDO generales.
CO_SDO1.c CO_SDO1.h	Endpoint de comunicaciones SDO que debe implementarse por defecto.
CO_SYNC.c CO_SYNC.h	Endpoint de comunicaciones para los consumidores del Objeto SYNC.
CO_TOOLS.c CO_TOOLS.h	Herramientas de conversión de identificadores entre el formato Microchip y CANopen. Todos los COB-ID están guardados en formato Microchip y son Convertidos a formato CANopen cuando es necesario.
CO_ABERR.h	Definiciones de errores comunes. Requerido para todas las aplicaciones.

Las características más destacables en este diseño incluyen:

- Máquina de estados encargada de manejar todas las comunicaciones entre los Nodos y Objetos.
- Un Service Data Object (SDO).
- Hasta 4 transmit y 4 receive Process Data Objects (TPDOs y RPDOs).

- Soporte para trasferencia de mensajes acelerados y segmentados.
- Soporte de mapeado de PDOs estático.
- Estructura para el Diccionario de Objetos PDO y SDO.
- Node Guard/Life Guard.
- Consumidor de Objetos SYNC.
- Productor de Heartbeat.
- Soporte para la configuración del módulo ECAN.

Como se puede observar en esta lista, la pila está desarrollada para aplicaciones con un claro perfil de "esclavo".

3.4.3 Uso de la pila CANopen

En este apartado se hará una descripción más detallada de los archivos de la pila CANopen que han sido modificados para adaptarse al diseño de sistemas CANopen basados en microcontroladores PIC18 de Microchip®.

El archivo **main.c** contiene la rutina del programa principal, a través del cual se configuran todos los parámetros iniciales y posteriormente se ejecutan todos los procesos relacionados con la aplicación y comunicación.

En cuanto a la configuración, sigue los siguientes pasos:

1. Configura, llamando a la función *void TimerInit(void)*, el **timer_0** para generar un evento cada 1 ms que será utilizado para controlar todos los procesos temporales del sistema.

2. Almacena el COB-ID de sincronismo, en formato CANopen, mediante la llamada a la función *mSYNC_SetCOBID(SYNC_COB)*, donde *SYNC_COB* representa el identificador de SYNC. Posteriormente el COB-ID de sincronismo es transformado a formato Microchip mediante la llamada *mTOOLS_CO2MCHP(mSYNC_GetCOBID())* y almacenado en memoria tras invocar de nuevo a la función *mSYNC_SetCOBID(mTOOLS_GetCOBID())*. La función *mTOOLS_GetCOBID()* devuelve el valor del COB-ID de sincronismo en formato Microchip. Para más información acerca de los identificadores de mensaje es conveniente revisar el apartado 2.4.3 de este documento.

3. Asigna el Node-ID del dispositivo a través del la función *mCO_SetNodeID(NodeID)*.

4. Selecciona la tasa de transferencia del Bus por medio de la función *mCO_SetBaud(bitrate)*, donde bitrate es un valor perteneciente al rango [0…8] y que estará asociado a los parámetros de configuración de la tasa de transferencia.

5. Configura el **Node Guard ó Heartbeat** según lo expuesto en el apartado 2.4.6.4 de este documento, por medio de las funciones mNMTE_SetHeartBeat(HeartBeat), mNMTE_SetGuardTime(GuardTime), mNMTE_SetLifeFactor(LifeFactor), donde HeartBeat y GuardTime representan tiempos dados en milisegundos y LifeFactor es un factor utilizado en el **Node Guard**.

6. Llamada a la función de inicialización de todos los parámetros relacionados directamente con la aplicación que se desea desarrollar, en este caso la función *void JR3Init (void)*.

7. Inicializar el Nodo. Esta inicialización se realiza a través de la llamada *mCO_InitAll()*, que realiza otra sub-llamada a la función *void _CO_COMMResetEventManager(void)* donde se resetea el estado del Nodo y se producen sucesivas llamadas a otras funciones para configurar todo el hardware relacionado con la tasa de transferencia (*mCANReset(CANBitRate)*) y con la implementación de los ***endpoint*** tales como el NMT (*void _CO_COMM_NMT_Open(void)*), el *SYNC consumer* (*void _CO_COMM_SYNC_Open(void)*), el SDO por defecto (*void _CO_COMM_SDO1_Open(void)*) y el *Node Guard* ó *Heartbeat* (*void _CO_COMM_NMTE_Open(void)*). Una vez creados todos los endpoint se asigna los valores de estos a los filtros de los buffers de entrada (*void _CANEventManager(void)*), se activa el modo Normal de operación del módulo CAN (*mCANOpenComm()*) y por último se activa el modo PREOPERATIONAL después de envíar el mensaje de Boot-up.

La ejecución de los procesos asociados a la aplicación y comunicaciones se realizan a través del código incrustado dentro de un bucle infinito *while* de la función **main**.

En este apartado existen tres instrucciones claramente diferenciadas, que son las siguientes:

1. Procesado de todos los eventos CANopen relacionados con las comunicaciones a través de la función *mCO_ProcessAllEvents()*, tales como PDOs, SDOs, SYNC consumer, etc.

2. Procesado de las aplicaciones realizadas por el Nodo. Esta función ha de ser desarrollada por el usuario y ha de realizar todas las funciones específicas de la

aplicación que se desea desarrollar. En este caso la función se llama *void JR3ProcessEvents (void)* y se encuentra en el archivo AppJR3.c.

3. Tratamiento de todos los procesos relacionados con eventos temporales tales como Node Guard ó Heartbeat, control de tiempo de caducidad de transferencia SDO segmentada, tiempos de inhibición de los PDOs, etc. Este apartado se encuentra implementado en la función *mCO_ProcessAllTimeEvents()* y se ejecuta cada vez que el **timer_0** se desborda. Es decir, cada vez que la función *unsigned charTimerIsOverflowEvent (void)* devuelve el valor TRUE.

3.4.3.1 Funciones implementadas

void JR3Init (void)

JR3Init es una función desarrollada por el usuario y que configura todos los parámetros del microcontrolador relacionados con la lectura del sensor como el puerto SPI y el módulo CCP, como se puede ver detallado en los capítulos 3.3.2 y 3.3.3. Esta función se encuentra localizada en el archivo **AppJR3.c**.

Tras configurar los parámetros relacionados con la adquisición de datos, el siguiente paso que realiza la función *JR3Init* es configurar los modos de sincronismo de los TPDOs según el perfil de dispositivo de un sensor de fuerza. Se han de habilitar dos PDOs, uno en modo asíncrono (TPDO_1) y el otro en modo síncrono (TPDO_2). Es decir, el PDO_1 transmitirá un mensaje cada vez que detecte un cambio de fuerza o par y el PDO_2 enviara el mensaje con los datos de fuerza/par de forma sincronizada con el SYNC Object.

Esta configuración se logra a través de las variables *uJR3SyncSet_1*, *uJR3SyncCount_2* y *uJR3SyncSet_2*. *uJR3SyncSet_1* configurara el TPDO_1 y las otras dos variables el TPDO_2 adoptando los siguientes valores. *uJR3SyncSet_1 = 254*, configura el TPDO_1 para transmisión asíncrona. *uJR3SyncCount_2 = uJR3SyncSet_2 = 1*, configura el TPDO_2 para transmisión acíclica síncrona.

El siguiente paso es inicializar los buffers de transmisión de ambos TPDOs (uLocalTPDO2Buffer y uLocalTPDO1Buffer). En estos buffers se almacenaran los datos de fuerza/par que serán enviados según el modo de sincronización de cada TPDO.

Por último, en esta función se han de crear los COB-ID de los PDOs, asociarlos al buffer de transmisión e indicar el tamaño de dicho buffer. A continuación se muestra el proceso para el PDO_1 ya que para los siguientes PDOs el código no varía.

```
//Crear el identificador y convertirlo en formato Microchip
mTOOLS_CO2MCHP(mCOMM_GetNodeID().byte + 0xC0000180L);
// Almacenar el COB-ID del TPDO1
mTPDOSetCOB(1, mTOOLS_GetCOBID());
// Asociar buffer del PDO1
mTPDOSetTxPtr(1, (unsigned char *)(&uLocalTPDO1Buffer[0]));
```

void JR3ProcessEvents (void)

Esta es la función principal de la lectura del sensor y a través de la cual se gestiona todo el proceso.
La primera parte de esta función consta de las declaraciones de variables necesarias para desarrollar y detectar un cambio en la lectura del sensor. Estas variables son:

- *UNSIGNED16 uJR3force*, almacena la lectura del sensor una vez que es convertido a código binario.

- *static UNSIGNED16 uJR3forceOld*, guarda la lectura anterior del sensor para detectar cambios con respecto a la actual (*uJR3force*).

El siguiente paso es realizar la lectura de fuerza como se ha detallado en el capítulo 3.3.4. Tras realizarla se chequea si se han producido cambios con respecto a la lectura anterior. En caso afirmativo esto es indicado mediante un bit de estado, el nuevo valor es almacenado en los buffers de transmisión y la nueva posición es guardada en *uJR3forceOld*. El código implementado es el siguiente:

```
// Indicar si se produce un cambio.
if (uJR3forceOld.word != uJR3force.word)
{
// Bits de estado.
uJR3State.bits.b1 = 1; // TPDO_1
uJR3State.bits.b4 = 1; // TPDO_2
// Almacenar posición en buffers de TPDO.
*(UNSIGNED16 *)uLocalTPDO2Buffer = *(UNSIGNED16
*)uLocalTPDO1Buffer = *((UNSIGNED16 *)(&uJR3force.word));
// Guardar nuevo valor de fuerza.
uJR3forceOld.word = uJR3force.word;
}
```

La variable *uJR3State* está declarada de forma global e indica el estado de los TPDOs según se indica a continuación:

- b0, indica que el TPDO_1 ha de ser enviado.
- b1, indica al TPDO_1 que la lectura del sensor ha variado.
- b2, cuando la variable *u JR3SyncSet_1 = 0* indica al TPDO_1 que la lectura está lista para ser enviada cuando se reciba un SYNC Object.
- b3, indica que el TPDO_2 ha de ser enviado.
- b4, indica al TPDO_2 que la lectura del sensor ha variado.
- b5, cuando la variable *u JR3SyncSet_2 = 0* indica al TPDO_2 que la lectura esta lista para ser enviada cuando se reciba un SYNC Object.
- b6 y b7, no se utilizan.

Una vez que se ha detectado un cambio de posición en la lectura del sensor, el siguiente paso es chequear el modo de sincronismo y en función de este enviar la lectura. Si estamos en modo asíncrono (*uJR3SyncSet* igual a 254 ó 255) envía la lectura inmediatamente y en el modo síncrono (*uJR3SyncSet* en el rango [1…253]) espera la recepción de tantos mensajes de sincronismo como indique el valor de *uJR3SyncSet*.

Existe la posibilidad de que se le asigne a la variable *uJR3SyncSet* el valor '0' en cuyo caso cumple la misma funcionalidad que cuando se le asigna el valor '1' pero sólo transmitirá el mensaje si se ha producido un cambio de posición antes de recibir el SYNC Object.

```
if (uJR3State.bits.b1)
{
        switch (uJR3SyncSet_1)
        {
        case 0: // transmisión acíclica síncrona.
        // Activar flag de transmisión síncrona.
        uJR3State.bits.b2 = 1;
        break;
        case 254: // Transmisión asíncrona.
        case 255:
        // Activar flag de transmisión asíncrona.
        uJR3State.bits.b0 = 1;
        break;
        }
}
```

La última tarea que realiza esta función es la activación de un flag para que el dato sea enviado por el NMT y el reset de los flag indicadores de envío de datos pendientes.

```
// Esta el mensaje listo para envío?
if (mTPDOIsPutRdy(1) && uJR3State.bits.b0)
{
// Informar al NMT que el mensaje está listo para ser enviado.
mTPDOWritten(1);
// Reset de los flag indicadores de envío pendiente.
uJR3State.bits.b0 = 0;
uJR3State.bits.b1 = 0;
}
```

void CO_COMMSyncEvent(void)

Es la función encargada del tratamiento de sincronismos de la comunicación. En este caso se editará, para el caso del TPDO_1, de la forma que muestra el código siguiente:

```
{
// TPDO1
// Procesar sólo en modo síncrono
if ((uJR3SyncSet_1 == 0) && (uJR3State.bits.b2))
{
        // Activar flag para envío de mensaje.
        uJR3State.bits.b2 = 0;
        u JR3State.bits.b0 = 1;
}
else
{
        if ((u JR3SyncSet_1 >= 1) && (uJR3SyncSet_1 <= 240))
        {
        // Cuenta de mensajes de sincronismo para el modo síncrono.
        uJR3SyncCount_1--;
        // Habilitar envío de mensaje.
                if (uJR3SyncCount_1 == 0)
                {
                // Reset del contador de mensajes de sincronismo.
                uJR3SyncCount_1 = uJR3SyncSet_1;
                // Activar flag de transmisión.
                uJR3State.bits.b0 = 1;
                }
        }
```

```
}
```

void AppLED(void)

Esta función controlara el LED con el fin de indicar el estado del Nodo. Cuando el dispositivo se encuentre en estado STOPPED el LED permanecerá apagado, parpadeara en PRE-OPERATIONAL y se encenderá en OPERATIONAL. El cambio de un estado a otro se puede ver en la figura 22 de este documento.

```
if (COMM_STATE_STOP)
{
LATCbits.LATC1 = 0;
}else if (COMM_STATE_OPER)
        {
        LATCbits.LATC1 = 1;
        }else
        {
                if ( ContLED <= 0)
                {
                ContLED += LED_PERIOD_MS;
                LATCbits.LATC1 = ~LATCbits.LATC1;
                }else
                        ContLED -= CO_TICK_PERIOD;
        }
```

Donde:

- LED_PERIOD_MS*:* Indica la frecuencia con la que el LED parpadea.
- CO_TICK_PERIOD*:* Indica con qué frecuencia en milisegundos es llamada la función.

3.4.3.2 Tasa de transferencia

El último paso para implementar la comunicación CAN es el cálculo de la tasa de transferencia nominal, que es el número de bits por segundo que será capaz de transmitirse a través del bus. El cálculo de la tasa de transferencia es muy importante a la hora de diseñar cualquier dispositivo CAN, ya que si este no es calculado de forma adecuada el nodo no podrá conectarse al bus de control ó los datos trasmitidos y recibidos no serán interpretados de forma adecuada.
Para ello, el protocolo CANopen proporciona una serie de tasas de transferencia recomendadas que quedan recogidas en la tabla 30.

Tabla 30. Tasas de transferencia recomendadas por el protocolo CANopen

Bit rate Bus length (1)	Nominal bit time t_b	Number of time quanta per bit	Length of time quantum t_q	Location of sample point
1 Mbit/s 25 m	1 µs	8	125 ns	6 t_q (750 ns)
800 kbit/s 50 m	1,25 µs	10	125 ns	8 t_q (1 µs)
500 kbit/s 100 m	2 µs	16	125 ns	14 t_q (1,75 µs)
250 kbit/s 250 m (2)	4 µs	16	250 ns	14 t_q (3,5 µs)
125 kbit/s 500 m (2)	8 µs	16	500 ns	14 t_q (7 µs)
50 kbit/s 1000 m (3)	20 µs	16	1,25 µs	14 t_q (17,5 µs)
20 kbit/s 2500 m (3)	50 µs	16	3,125 µs	14 t_q (43,75 µs)
10 kbit/s 5000 m (3)	100 µs	16	6,25 µs	14 t_q (87,5 µs)

Para realizar el cálculo se ha de tener en cuenta la partición del tiempo de bit nominal representado en la figura 39.

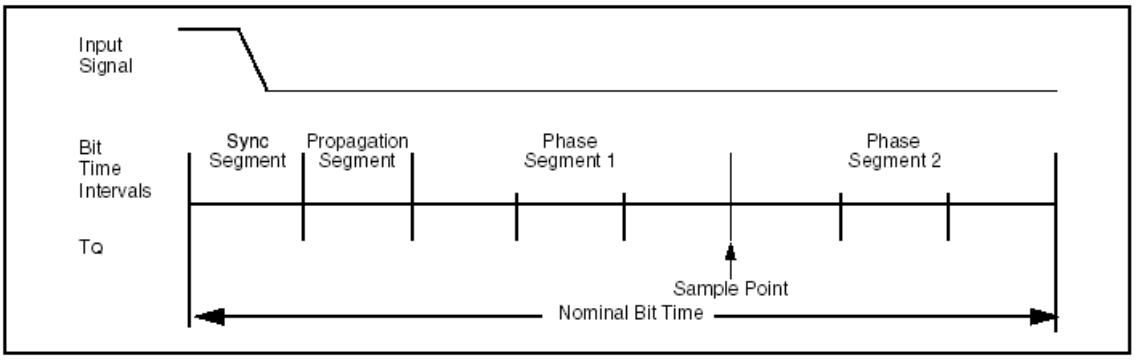


Figura 39. Partición del tiempo de bit nominal

El tiempo de bit nominal está dividido en 4 segmentos de tiempo no solapados, y que a su vez, están subdivididos en unidades de tiempos denominados Time Quanta (TQ). Por definición, el tiempo de bit nominal ha de ser programado con un mínimo de 8 TQ y un máximo de 25. Los segmentos son los siguientes:

- **Synchronization Segment (Sync_Seg)**, usado para sincronizar los distintos nodos del bus aprovechando los flancos de bajada de la trama transmitida. Su duración es de 1 TQ.

- **Propagation Time Segment (Prop_Seg)**, este segmento es utilizado para compensar los retardos de propagación de la señal en el Bus y su duración es de 1 a 8 TQ.

- **Phase Buffer Segment 1 (Phase_Seg1)**, determina el punto de muestreo de la señal del Bus y su duración es de 1 a 8 TQ.

- **Phase Buffer Segment 2 (Phase_Seg2)**, proporciona un retraso antes de la transmisión de siguiente bit y aunque puede ser programado con una longitud de 1 a 8 TQ, su longitud mínima debe ser de 2 TQ debido a tiempos de procesado de los datos.

Todos los cálculos realizados para obtener la tasa de transferencia se realizan desde la premisa de que el oscilador del sistema es ideal y debido a que esto no es real el modulo CAN dispone de un modo de resincronización que aumenta el Phase_Seg1 cuando el flanco se produce después del Sync_Seg ó disminuye el Phase_Seg2 cuando el flanco se produce antes del Sync_Seg. La cantidad de TQ que estos segmentos aumenta o disminuyen está definido en un segmento denomina **Synchronization Jump Width (SJW)** y que puede ser programada con una longitud de 1 a 4 TQ.

Con ayuda de las ecuaciones 3, 4 y 5 se calcularán los datos necesarios que se asignarán a los registros BRGCON1, BRGCON2 y BRGCON3. La frecuencia del reloj externo es dividida por el factor denominado BRP.

$$T_Q = \frac{2 * (BRP + 1)}{F_{OSC}}$$

Ecuación 3

$$\text{Nominal Bit Time} = T_Q * (\text{Sync\_Seg} + \text{Prop\_Seg} + \text{Phase\_Seg1} + \text{Phase\_Seg2})$$

Ecuación 4

$$T_{BIT} = \frac{1}{\text{Nominal Bit Time}}$$

Ecuación 5

Teniendo en cuenta que la F_{OSC} es de 20 MHz, que el valor de BRP seleccionado es igual a 0 y el valor de SJW igual a 2 T_Q, se han obtenido los valores que se muestran en la tabla 31, para una tasa de transferencia de **500 MHz**, que es la más adecuada para los sistemas CAN utilizados en el departamento.

Tabla 31. Parámetros calculados en la tasa de transferencia

TQ[µs]	0,1
Sync_Seg	1 T_Q
Prop_Seg	8 T_Q
Phase_Seg1	7 T_Q
Phase_Seg2	4 T_Q
Nominal Bit Time[µs]	2
T_{BIT} [Mhz]	500
BRGCON1	40h
BRGCON2	B7h
BRGCON3	83h

Capítulo 4

Ensayos y resultados

En este capítulo se explicarán los diferentes ensayos y pruebas que se han llevado a cabo hasta alcanzar la versión más óptima posible de la aplicación, que es la que se ha detallado a lo largo del capítulo 3. Consta de pruebas o versiones de código que, por no conseguir el objetivo buscado o bien por no saturar más el código final no se han incluido en la versión definitiva pero que sí se han tenido presentes a la hora de realizar la aplicación. Finalmente se incluirán las pruebas realizadas tras realizar la integración del dispositivo creado en un sistema real CAN, que permitirá comprobar si este cumple con todas las expectativas y especificaciones requeridas.

4.1 Captura de pulso de comienzo

Uno de los mayores problemas a la hora realizar la aplicación deseada viene relacionada con la captura de la señal de *pulse start*, explicada en detalle en el apartado 3.1.1.1. A continuación se explican las pruebas realizadas antes de llegar a la versión final.

Una vez estudiados los recursos del microcontrolador y las técnicas disponibles para capturar una señal, quedan dos opciones disponibles:

- Medir el ancho de pulso de la señal con la interrupción externa (INTEXT).
- Capturar el pulso con el módulo CCP.

Después de realizar las pruebas pertinentes, la primera opción quedó descartada ante la incapacidad de medir el tiempo deseado (4,8 µs). Esto se debe a que velocidad de la señal DCLK es demasiado elevada para ser procesada por la interrupción externa.

Por el contrario, este problema no ocurre al usar el módulo CCP que éste está dotado de funciones específicas para poder medir señales de este tipo. Aun así y tal y como se muestra en el apartado 3.3.3 se han reducido el número de instrucciones a ejecutar para evitar problemas de velocidad. Para comprobar que se estaba

capturando el *pulse start* correctamente se creó una matriz de datos que se rellenara con el valor de las veinte últimas señales capturadas. El pseudocódigo para esta prueba se muestra a continuación:

```
***configuración módulo CCP***
while(!PIR1bits.CCP1IF); //esperar a flanco de subida
TMR3L=0x00; //borrar cuenta de Timer3
PIR1bits.CCP1IF = 0x00; // limpiar flag modo capture
CCP1CONbits.CCP1M0=0;//configurar capture por flanco de    bajada
while(!PIR1bits.CCP1IF); //esperar a flanco de bajada
matriz[y]=TMR3L; //se guarda valor de la señal capturada en la matriz
y=(y+1)%20;//  se guardan las últimas 20 capturas
***lectura de puerto serie****
```

Con ayuda del depurador del compilador (*ICD2*) utilizado se creó una tabla con los valores almacenados por esta matriz. El resultado se muestra en la figura 40 y demuestra que la aplicación es capaz de capturar la señal deseada con una frecuencia aceptable y de manera correcta.

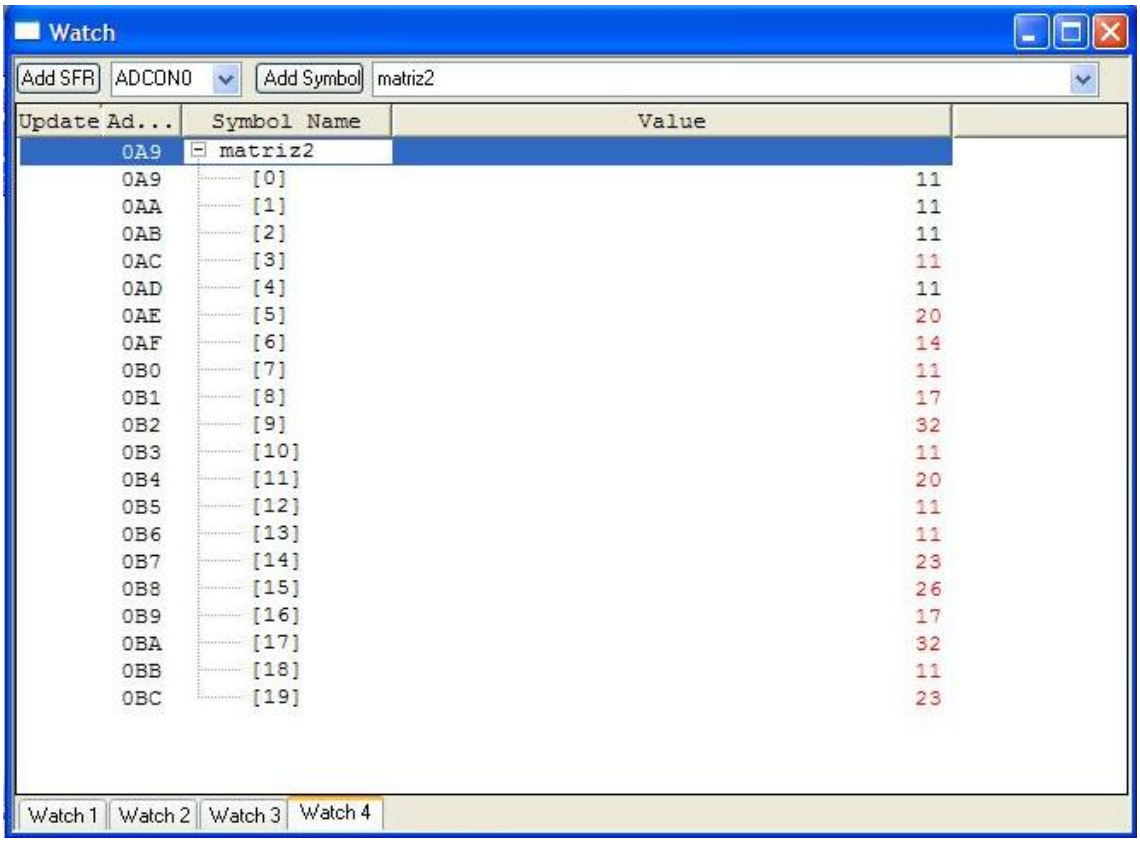


Figura 40. Matriz con valores de pulse start recogidos

4.2 Adquisición de datos por puerto serie

El siguiente paso es comprobar el correcto funcionamiento de la comunicación serie, encargada de recoger los datos de fuerza y par. Antes de elegir el puerto SPI como la opción más adecuada, se analizaron el resto de posibilidades la hora de realizar una comunicación serie con un dispositivo de este tipo. A continuación se presentan los dos módulos de comunicación serie de los que dispone el microcontrolador y los motivos por los que fueron descartados para este diseño en cuestión. Las opciones posibles son:

- Módulo EUSART:
    - Modo asíncrono. Fue descartado ya que era necesario mantener una comunicación síncrona con la señal DCLK que dispone el sensor.
    - Modo síncrono. La segunda opción más viable. El problema surgió con la velocidad de transmisión de datos (*baud rate*), ya que el sensor transmite a una velocidad de 64000 baudios y esta comunicación posee una velocidad máxima de 57600 baudios. Aun así se implementó y las pruebas en tiempo de ejecución demostraron que existía un problema de sobrecarga de datos (*overrun*), por lo que los datos llegaban más rápido al puerto serie de lo que éste era capaz de procesarlos por lo que el sistema se saturaba.
- Módulo MSSP:
    - Modo I^2C. A pesar de ser síncrono, no era una solución para este diseño ya que para conseguir usar este modo el flujo serie de datos del dispositivo conectado (en este caso el sensor) debe regirse bajo protocolo I^2C, ya que éste posee unas características específicas como una frecuencia de reloj máxima limitada a 3,4 MHz.
    - Modo SPI. La opción elegida, por ser la que a nivel teórico cumple con los requisitos de la comunicación del sistema estudiado, ya que permite una tasa de datos de hasta 10 Mbps. Tras las pruebas experimentales realizadas resultaba la opción más fiable.

A nivel hardware y para conseguir cambiar de manera sencilla de un modo de funcionamiento a otro, se soldaron conectores a cada pin correspondiente del microcontrolador para intercambiar el conexionado de las diferentes señales. En la figura 41 se puede ver la placa experimental sobre las que se realizaron estas pruebas, con dichos cambios.

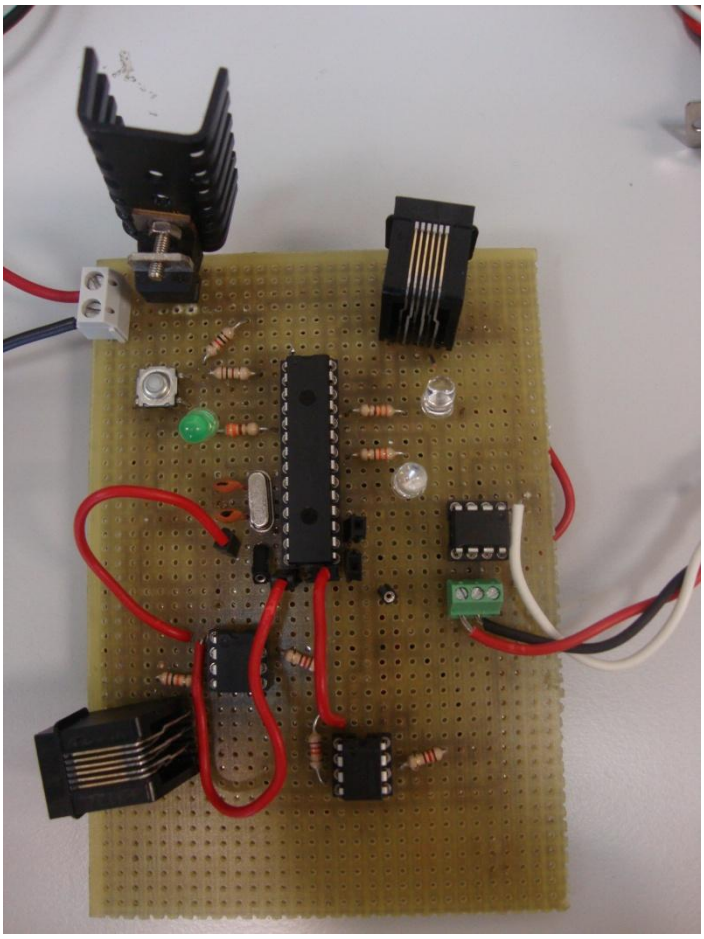


Figura 41. Placa experimental

4.3 Estudio y tratamiento de los datos adquiridos

Para comprobar la fiabilidad del sistema y de los datos obtenidos se realizaron una serie de pruebas que se expondrán a continuación.

4.3.1 Cálculo de tasa de errores

La hipótesis inicial tomada fue que se debía estar recibiendo información de los 8 canales mencionados en la sección 3.1.1.1. Esto es, como primera aproximación se supuso que el modelo de sensor elegido sólo transmitía combinaciones de bits de dirección entre 0 y 7. Combinaciones de bits de dirección entre 8 y 15 se tomarían como falsos positivos y nos ayudarían por tanto a hallar nuestra tasa de errores. Para ello se añadió al código dos variables de tipo contador. Uno de ellos (*contador_ok*) se incrementaría cada vez que se hubieran capturado paquetes de datos cuyos bits de direcciones correspondieran con los canales de 0 a 7, y el otro (*contador_error*) cada vez que los bits de direcciones correspondieran a los canales del 8 al 15. El pseudocódigo implementado para esta prueba, está basado en la rutina de adquisición de datos (3.3.4) y se muestra a continuación.

```
int contador_ok=0;
int  contador_error=0;
…rutina adquisición de datos…
if(lecturabits.B23==0 && lecturabits.B22==0 && lecturabits.B21==0 && lecturabits.B20==0){//Si los bits de direcciones van del 0000 al 0111 (canales 0-7)
contador_ok++;
…se guarda la lectura de fuerza…
}
else { // sino son del 8-15
contador_error++;
}
```

Una vez visualizados los valores de los dos contadores en diez pruebas realizadas de 1 minuto de duración cada una, se observó que la tasa de combinaciones de bits de dirección entre 0 y 7 era igual a la de entre 8 y 15 (50% frente a 50% por norma). Así se dedujo empíricamente que el modelo de sensor utilizado transmite sobre 16 canales y no sobre 8 como la primera hipótesis.

4.3.2 Visualización LED

La siguiente prueba sirve para observar el comportamiento del dispositivo en tiempo real ante un ejercicio de tracción o compresión sobre el sensor. Para ello se decidió añadir unos LED cuya salida variase en función de cada medida de fuerza y par, encendiéndose o apagándose en función de unos umbrales de fuerza previamente establecidos mediante un sencillo algoritmo condicional, mostrado a continuación:

```
if (fzbits.word>0){ // Si la fuerza ejercía sobre el eje z es mayor que 0…
PORTABits.RA3=1; //…led verde encendido
}else{
PORTABits.RA3=0; //…sino led verde apagado
}
```

Para llevar a cabo estas pruebas se dotó al sensor de una estructura plástica fácilmente manejable para poder así ejercer diferentes tipos de fuerza sobre cada eje. En la figura 42 se muestra el sensor y la estructura referida.

Figura 42. Estructura plástica del sensor

Tras realizar varios ensayos estableciendo varios márgenes de funcionamiento en cada eje, y a pesar de alguna variación indeseada en el estado de los leds, se puede decir que se alcanzan resultados de visualización bastante fiables.

4.4 Pruebas en sistemas de control CAN

El último paso para comprobar la funcionalidad del dispositivo diseñado es realizar las pruebas pertinentes sobre un sistema real con comunicación CAN. Para ello se realizaron ensayos sobre dos sistemas diferentes, que a nivel hardware están formados por los mismos elementos, un micro PC y una tarjeta de red CAN para comunicar el micro PC con los distintos dispositivos CAN, pero que a nivel software poseen diferencias. El primero de ellos tiene el sistema de control implementado en lenguaje C bajo Linux® y el segundo con un sistema implementado en MATLAB/SIMULINK®. En las figuras 43 y 44 pueden observarse estos sistemas de control mencionados.

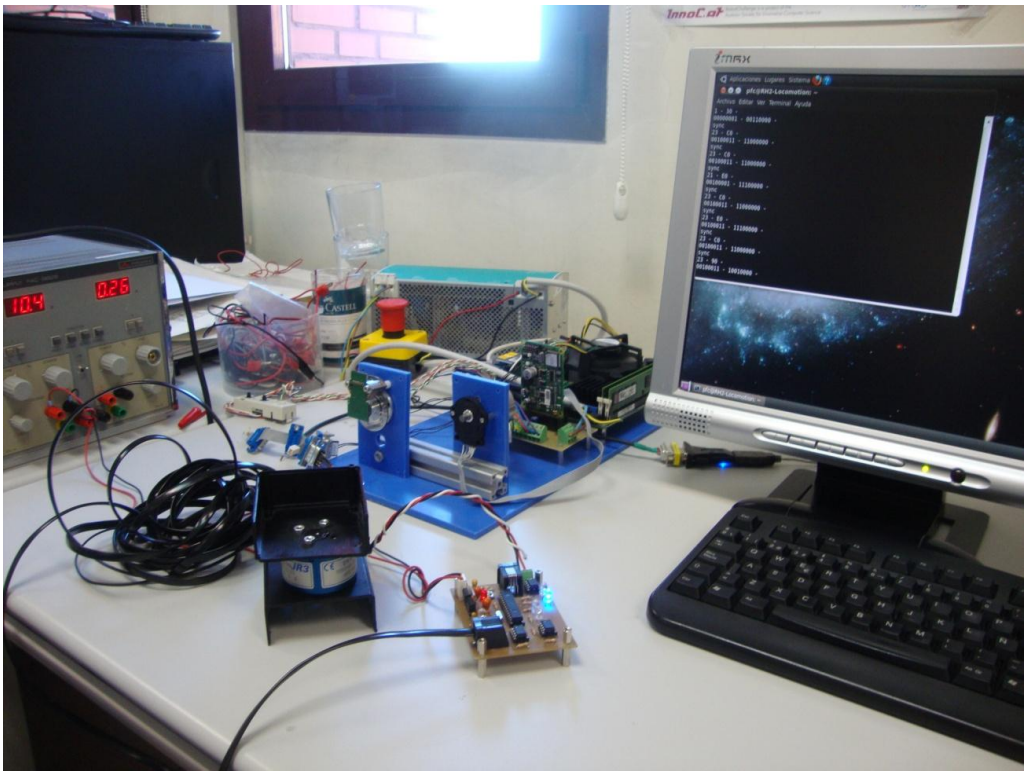


Figura 43. Sistema de control trabajando bajo Linux®

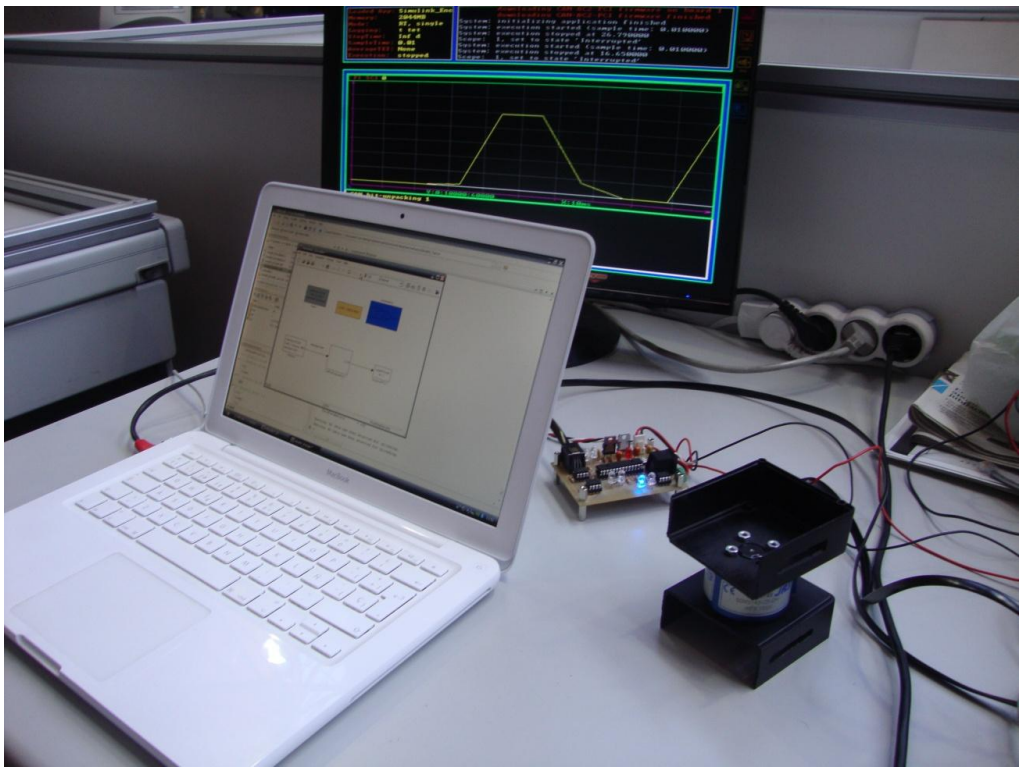


Figura 44. Sistema de control implementado con Matlab/Simulink®

4.4.1 Puesta en funcionamiento

Cualquier dispositivo CANopen requiere de una configuración mínima para comenzar a funcionar en un sistema de control. Esta configuración mínima consiste en enviar la trama ***Start Remote Node*** que permite al NMT Maestro activar el estado OPERATIONAL del NMT Esclavo seleccionado.

La trama está formada por el identificador '0' y dos bytes de datos. El primero indica la petición de activación del estado OPERATIONAL del dispositivo y el segundo el NodeID del dispositivo seleccionado. En la tabla 32 se puede observar la trama de un mensaje *start node*.

Tabla 32. Mensaje *start node*

COB-ID	COMANDO	DATO
0x00	0x01	NodeID

Además de activar el nodo para que pueda desempeñar sus funciones, en muchos casos es necesaria la configuración de parámetros y activación de ciertas características para que el nodo pueda realizar todas las tareas requeridas por el sistema. Esta configuración puede ser realizada tanto en el estado PREOPERATIONAL como en el OPERATIONAL, aunque si el parámetro a modificar afecta al desarrollo del proceso en tiempo de ejecución sólo podrá realizarse en el estado PRE-OPERATIONAL, es decir, antes de enviar la petición *Start Remote Node*.

Estas configuraciones se realizan a través de SDOs y en este caso consiste en configurar el modo de sincronismo si fuera necesario, abrir los TPDOs que van a ser utilizados y cerrarlos en el caso de que ya no sean necesarios.

4.4.1.1 Cambiar modo de sincronismo

El modo de sincronismo depende directamente de la variable *uJR3SyncSet* y esta puede ser modificada a través del sub-index 0x02 del Objeto de Diccionario 1800h para el TPDO_1 y 1801h para el TPDO_2. El valor de esta variable indica:

- '**0**', modo acíclico síncrono. Si se ha producido un evento antes del SYNC Object se transmite el PDO.
- [**1… 253**], modo síncrono. Transmisión de PDO después de recibir tantos mensajes de sincronismo como indique el valor de *uJR3SyncSet*.
- '**254**' y '**255'**, modo asíncrono. Envía un TPDO cada vez que se detecta un cambio de posición.

En las tablas 33 y 34 se muestra la estructura de la trama necesaria para realizar el cambio del modo de sincronismo.

Tabla 33. Cambiar modo SYNC TPDO_1

COB-ID	COMANDO	L_INDEX	H_INDEX	SUB-INDEX	D1	D2	D3	D4
0x600+NodeID	0x2F	0x00	0x18	0x02	MODO_SYNC	0x00	0x00	0x00

Tabla 34. Cambiar modo SYNC TPDO_2

COB-ID	COMANDO	L_INDEX	H_INDEX	SUB-INDEX	D1	D2	D3	D4
0x600+NodeID	0x2F	0x01	0x18	0x02	MODO_SYNC	0x00	0x00	0x00

Donde:

- **COB-ID**, es igual a 600h más NodeID.
- **COMANDO**, indica un download request con 1 bytes de datos.
- **L_INDEX**, es la parte baja del index.
- **H_INDEX**, es la parte alta del index.
- **SUB-INDEX**, es el sub-index del Objeto.
- **MODE_SYNC**, valor comprendido entre 0x00 y 0xFF y que será asignado a *uJR3SyncSet*.
- **D2, D3, D4**, siempre 0x00.

4.4.1.2 Abrir TPDO

En las tablas 35 y 36 están representados los datos para abrir los TPDOs.

Tabla 35. Abrir TPDO_1

COB-ID	COMANDO	L_INDEX	H_INDEX	SUB-INDEX	D1	D2	D3	D4
0x600+NodeID	0x23	0x00	0x18	0x01	0x80+NodeID	0x01	0x00	0x40

Tabla 36. Abrir TPDO_2

COB-ID	COMANDO	L_INDEX	H_INDEX	SUB-INDEX	D1	D2	D3	D4
0x600+NodeID	0x23	0x01	0x18	0x01	0x80+NodeID	0x02	0x00	0x40

Donde:

- **COB-ID**, es igual a 600h más NodeID.
- **COMANDO**, indica un *download request* con 4 bytes de datos.
- **L_INDEX**, es la parte baja del índex.
- **H_INDEX**, es la parte alta del índex.
- **SUB-INDEX**, es el sub-índex del Objeto.
- **D1**, representa la parte baja del COB-ID del TPDO.
- **D2**, representa la parte alta del COB-ID del TPDO.
- **D3**, siempre 0x00.
- **D4**, comando abrir TPDO.

La herramienta para cerrar un TPDO es idéntica a la de abrirlo con la única diferencia del comando en D4 que sería 0xC0.

4.4.1.3 Estructura de los datos transmitidos

Los datos de fuerza o par se transmiten a través de los TPDO con una estructura fija en la que el COB-ID es el "180h + NodeID" ó "280h + NodeID2 en función del TPDO que lo transmita.
En el campo de dato se transmiten dos bytes, siendo el primero de ellos el byte menos significativo y el segundo el más significativo como se puede comprobar en la tabla 37.

Tabla 37. Estructura de datos del TPDO

COB-ID	DATO_L	DATO_H	
0x180+NodeID	xxxx	xxxx	TPDO_1
0x280+NodeID	xxxx	xxxx	TPDO_2

En la figura 45 se puede ver una captura de pantalla realizada en el sistema CAN que trabaja bajo Ubuntu (una distribución de GNU/Linux), donde se muestran los datos trasmitidos por el TPDO.

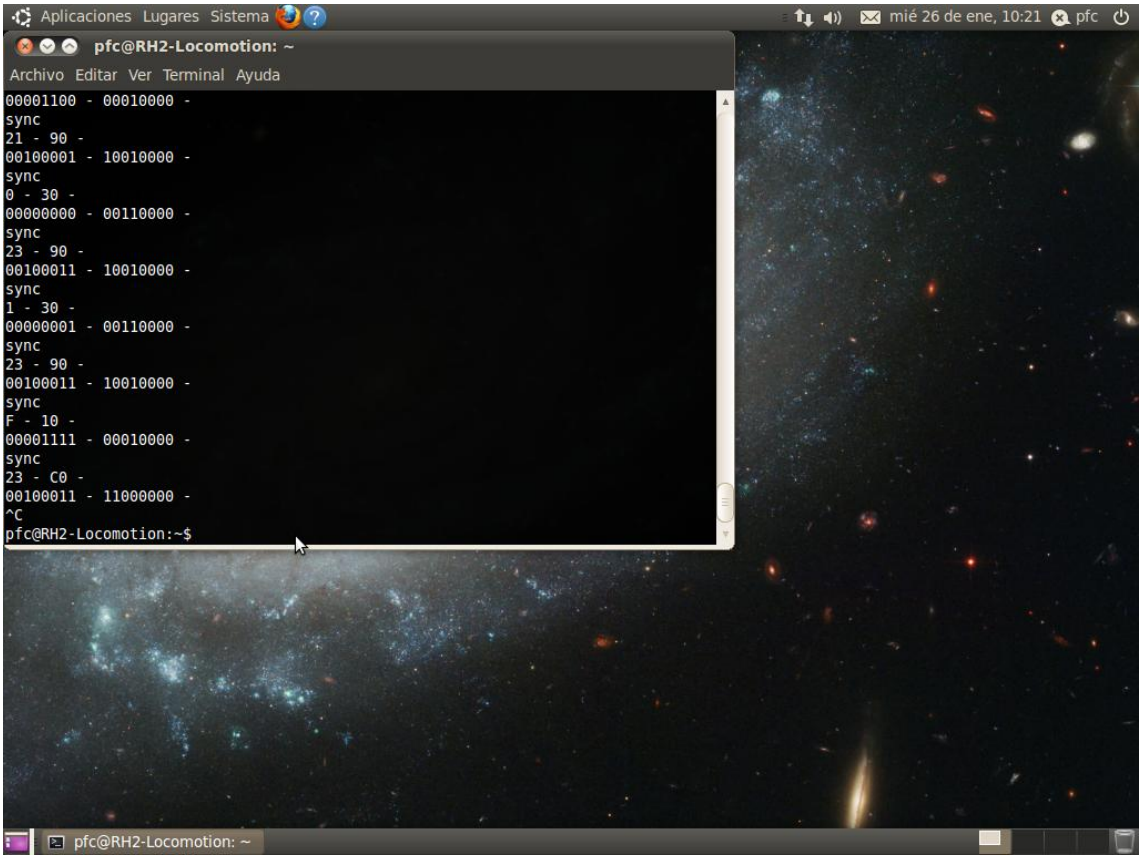


Figura 45. Datos enviados por el TPDO

4.4.2 Resultados

Tras realizar las primeras pruebas, el primer contratiempo que surgió fue que el nodo no cambiaba de estado, de PREOPERATIONAL a OPERATIONAL, según el diagrama de estados de la figura 22, y por lo tanto no existía comunicación. Este problema se solucionó colocando la resistencia que actúa como elemento de cierre, al que se hace referencia en el capítulo 2.3.4.2 de este documento, entre las señales de CAN_H y CAN_L.

Otro contratiempo que surgió fue que, después de enviar el mensaje de petición de sincronismo por parte del nodo implementado, el sistema no conseguía no conseguía sincronizarse correctamente. Después de revisar los pasos a seguir para poner en marcha un PDO, el problema estaba en que al abrir el PDO no se estaba colocando correctamente el NodeID del dispositivo, ya que el ID definido para el sensor de fuerza/par es el 0x50 y se estaba abriendo el 0x42.

Una vez solucionados todos los problemas, se consiguió realizar la implementación deseada.

Capítulo 5

Conclusiones y futuros desarrollos

En este capítulo se analizarán las conclusiones obtenidas, posteriormente se presentarán una serie de mejoras y ampliaciones sobre el diseño actual, y como punto final se hará un análisis crítico de los resultados finales.

5.1 Conclusiones

El proyecto realizado tenía como objetivo principal desarrollar un hardware para un sensor de fuerza/par que integrara un sistema de tratamiento y adquisición de datos. Para alcanzar este propósito final se plantearon una serie de objetivos intermedios que se plantean a continuación.

- Estudio del sensor elegido e interpretación de los datos que proporciona.
- Comunicación del sensor con un PC y programación necesaria para su control.
- Elección y comunicación del microcontrolador adecuado para el diseño así como de los controladores de comunicaciones.
- Adquirir los conocimientos necesarios sobre el entorno de desarrollo MPLAB de Microchip®, y su herramienta compiladora para microcontroladores de la familia PIC18.
- Desarrollo hardware del módulo, seleccionando para ello, los elementos que componen el diseño de la placa y su distribución dentro de ella.
- Desarrollo de aplicación software encargada de adquirir y dar un trato adecuado a los datos de lectura del sensor.
- Comprobación del correcto funcionamiento del dispositivo con pruebas a nivel hardware (visualización correcta de señales en osciloscopio) y a nivel software (*breakpoints* durante ejecución de programa, visualización de variables con lectura almacenada).

Una vez finalizado el proyecto se puede decir que se ha diseñado un dispositivo que cumple con todos los requisitos inicialmente definidos, ya que éste es capaz de adquirir y almacenar la lectura de datos que proporciona un sensor comercial de fuerza/par para su posterior envío a través de un bus de comunicaciones de tipo CANopen, para que pueda ser integrado en diferentes sistemas robóticos como, por ejemplo, ASIBOT.

5.2 Trabajos futuros y ampliaciones

En este apartado se plantearán futuras mejoras que permitan obtener más prestaciones y una mayor usabilidad del dispositivo.

El trabajo futuro o ampliación más relevante a realizar siguiendo la línea de desarrollo marcada por este proyecto, sería la implementación de algoritmos de control, como los mostrados en la sección 2.1 de este documento, una vez integrado el sensor y el módulo hardware en la estructura de ASIBOT.

A nivel hardware podría llevarse a cabo un cambio de controlador principal del módulo, pudiendo ser éste alguna FPGA o CPLD de las existentes en el mercado y que proporcionen unas mayores prestaciones, sobre todo en lo que a velocidad de tratamiento y adquisición de datos se refiere. También se podría pensar en utilizar otro tipo de sensor de fuerza/par existente en el mercado que ofreciera características similares pero que pudiera ser más accesible en lo que a la lectura de datos se refiere.

El otro aspecto hardware que se podría modificar sería cambiar el diseño PCB con componentes DIP, a hacerlos con SMD (*Surface Mounted Device*, soldadura en superficie), para hacer en la medida de lo posible, el diseño aún más compacto.

En cuanto a la programación del microcontrolador, el paso más relevante sería el de la implementación de los cuatro nodos que ofrece la pila CANopen, y de los que dos se utilizarían para enviar los datos de los tres ejes de fuerza de forma síncrona y asíncrona, y otros dos para hacer esto mismo con los tres datos de par. En un principio esto sería posible ya que los buffers de transmisión tienen una capacidad máxima de 64 bits, y enviando estos tres datos sólo se ocuparían 48, con lo que se podría usar el resto para indicar, delante de cada dato, la información que viene a continuación, al igual que hace el sensor con sus bits de direcciones.

Otro aspecto que puede ser mejorado es el consumo del dispositivo, que aunque en este caso no es un factor determinante, podría serlo en otras aplicaciones en las que se decida usar el dispositivo. En este caso existe la posibilidad de trabajar con el modo *sleep* tanto del microcontrolador PIC18, como del transceiver MCP2551 ya que este último también dispone de un modo de ahorro de energía.

5.3 Análisis crítico

Los resultados obtenidos en el proyecto han sido satisfactorios ya que, además de conseguir diseñar un módulo de conexión a CANopen para un sensor de fuerza/par, se ha logrado un diseño muy compacto, económico y ligero. Estas características permiten una fácil integración en cualquier sistema robótico donde el peso y tamaño de los componentes sea un factor muy determinante. Otro de los aspectos a tener en cuenta es que el módulo creado puede ser reprogramado y depurado tantas veces como se quiera una vez integrado en cualquier sistema, gracias al circuito de programación ICSP (*InCircuit Serial Programming*) incorporado en el diseño.

Sin embargo, la lectura de datos que nos proporciona el sistema es un tanto inestable. Además, existe cierto retardo en el comportamiento del dispositivo cuando es analizado en tiempo real en un sistema de control.

En definitiva, se ha desarrollado un sistema con grandes expectativas de ampliación y mejora, y que además abre nuevas líneas de investigación para futuros proyectos como el control de fuerza en robótica o el diseño y desarrollo de módulos hardware similares para otro tipo de sensores (inclinómetros, sensores anticolisión, etc.), y de esta manera seguir mejorando las prestaciones del sistema robótico ASIBOT.

Referencias

[1] Jardón Huete A.; *Metodología de diseño de robots asistenciales. Aplicación al robot portátil ASIBOT*; Tesis Universidad Carlos III de Madrid, 2006.

[2] Fortes Monteiro, J.; *Diseño e implementación encoder absoluto CANopen*; PFC Universidad Carlos III de Madrid, 2009.

[3] Conor J. Walsh; '*Analysis of Remote Center of Compliance Devices based on Stewart Platforms*'. Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, 2010.

[4] Neville Hogan; '*Impedance control of industrials robots*'. Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, 1984.

[5] M.H.Raibert;J.J.Craig; '*Hybrid position/force control of manipulators*'. California Institute of Technology, Pasadena, California, 1981.

[6] R. Volpe e P. Khosla; '*A Theoretical and Experimental Investigation of Explicit Force Control Strategies for Manipulators*', IEEE Transactions on Automatic Control, 1993.

[7] Robert Bosch GmbH '*BOSCH Controller Area Network protocol specification*', Revision 2.0, Stuttgart, 1991.

[8] CiA (CAN in Automation). http://www.can-cia.org/ Última visita: 10-12-2010

[9] "*CANopen: high level protocol for CAN-bus*".
http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf

[10] Ross M. Fosler, '*A CANopen Stack for PIC18 ECANTM Microcontrollers*'. Microchip Technology Incorporated.

[11] '*MPLAB® C18 C Compiler User's Guide*', Microchip Technology Inc., 2005.

[12] Quick Reference for RS-485, RS-422, RS-232.
http://www.rs485.com/rs485spec.html. Última visita: 5-2-2011

[13] *'MPLAB® ICD 2 In-Circuit Debugger User's Guide*', Microchip Technology Inc., 2005.

[14] '*CANopen: a CAL based communication profile for industrial systems*', Draft Standard DS-301. Revision 4.02, CAN in Automation Group, Febrero 2002.

[15] '*CANopen device profile for digital I/O* modules', Draft Standard DS-401 Revision. 3.0, CAN in Automation Group, Junio 2008.

[16] Sánchez Díaz, R.; *Arquitectura software y hardware para la automatización de una carretilla industrial*; PFC Universidad de Salamanca, 2007.

Anexos

En este anexo se detallarán otros aspectos del actual proyecto a tener de cuenta, como son el presupuesto o el listado de componentes. También se incluirá la información más relevante de la placa diseñada y de cada componente utilizado incluyendo las hojas de características correspondientes.

A.1 Presupuesto

En la tabla 38 se muestra un resumen del presupuesto de los componentes necesarios para la implementación del módulo de conexión a CANopen diseñado. En este presupuesto no se tendrá en cuenta la mano de obra y los costes de diseño del dispositivo.

Tabla 38. Resumen de presupuesto

Resumen de presupuesto de materiales	Importe
Microcontrolador P18F2580	6,91€
Transceiver MAX485(x2)	3,4 € x 2 = 6,8 €
Transceiver MCP2551	1,79 €
Sensor de Fuerza/par de JR3® modelo 50M31A-I25-DH	950 €
Regulador de Tensión LM7805	0,49 €
Regulador de Tensión LM7808	1,57 €
Oscilador cristal de 20MHz	1,95 €
Otros componentes(resistencias, condensadores, leds, interruptor, conectores)	4,25 €
Placa PCB	125 €
Total Presupuesto	1098,76 €
IVA 18%	197,77 €
Total presupuesto con IVA	**1296,5 €**

A.2 Listado de componentes

En la tabla 39 se muestran todos los componentes necesarios para la implementación del dispositivo.

Tabla 39. Listado de componentes

Componente	Valor	Cantidad	Modelo	Footprint	Descripción
Sensor de fuerza/par	-	1	50M31A-I25-DH		Sensor de fuerza/par
Micro controlador	-	1	P18F2580, encap. SDIP,SOIC	DIP.100/28/W.300/L1.400	Micro controlador
Transceiver		2	MAX485, encap. DIP	DIP.100/8/W.300/L.400	Controlador de comunicaciones serie
Transceiver		1	MCP2551, encap. DIP	DIP.100/8/W.300/L.400	Controlador de comunicaciones CAN
Cristal	20 MHz	1	XTAL,20Mhz, HC48/4H	CRYSTAL/.400X.150/LS.200/.034	Cristal oscilador
Regulador de tensión 5V	-	1	LM7805, encap. TO-220	TO220AB	Regulador para tensiones TTL
Regulador de tensión 8V	-	1	LM7808, encap. TO-220	TO220AB	Regulador para tensión de alimentación del sensor
Pulsador	-	1	4 pines	SWITCH	Circuito de reset
Diodo led		7	LED	LED/D.225/LS.125/.031	Indicador encendido
Conectores					
Conector RJ11		2	Conector RJ11, 6 pines	TELE/TM.420FACE/6X6	Conector de programación y del sensor
Conector DB9		1	Conector DB9		Conector CANopen
Condensadores					
Condensadores	22 pF	2	Cerámico, cap 0402		Cristal
Condensadores de Tántalo	10µF, 25 V	1	Tantalum capacitor		Filtrado señal alimentación
Condensadores de Tántalo	10µF, 35 V	3	Tantalum capacitor		Filtrado señal alimentación
Resistencias					
Resistencia	1KΩ	1	Chip precisión, 1k		Circuito reset
Resistencia	100 Ω	1	Chip precisión, 100		Circuito reset
Resistencia	330 Ω	7	Chip precisión, 330		Polarización led
Resistencia	120 Ω	2	Chip precisión, 120		Resistencia de terminal, MAX485

A.3 Circuitos impresos y esquemático de la placa

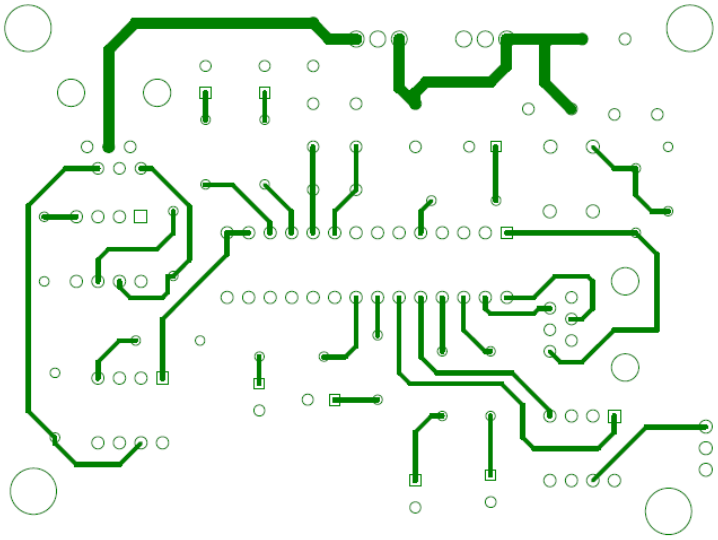


Figura 46. Capa TOP

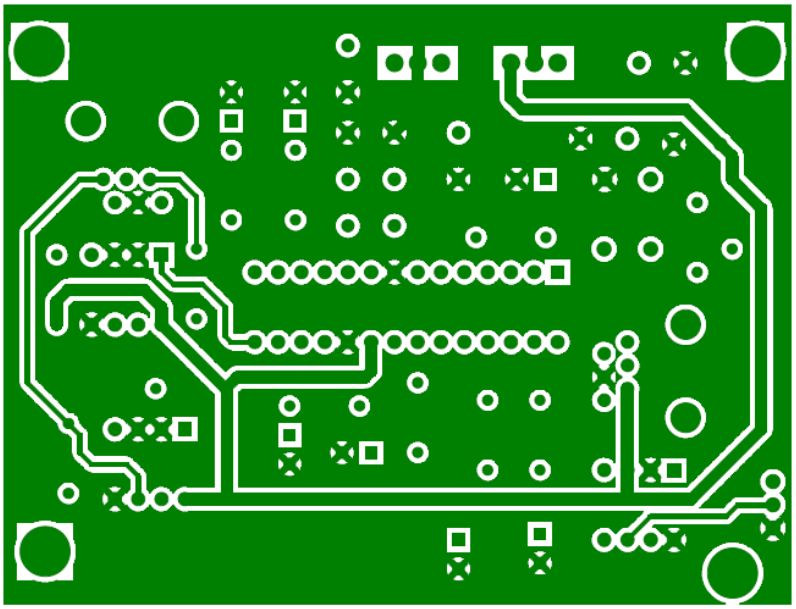


Figura 47. Capa BOTTOM

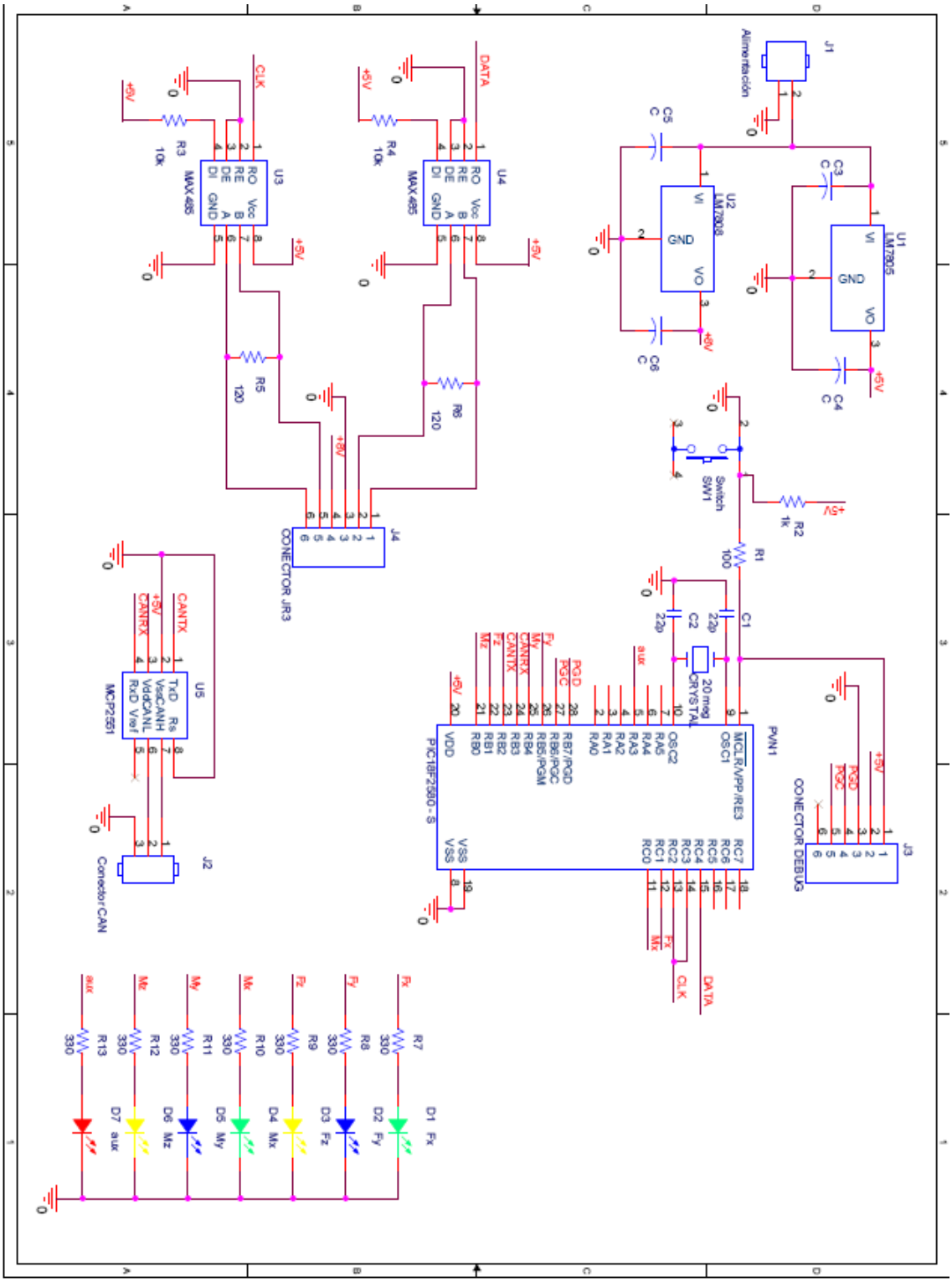


Figura 48. Esquemático de la placa

A.4 Sensor de fuerza/par JR3®

JR3, Inc. 22 Harter Ave, Woodland, CA 95776 (530) 661-3677

50M31A

SENSOR OVERVIEW

The 50M31 is a member of the ***JR3*** family of six axis force and torque transducers with internal electronics. It is 50mm in diameter and 31mm thick. The ISO25mm variant uses the mechanical interface specified by the International Standard ISO 9409-1 - A 25, other variants match specific industrial robots.

The 50M31 has signal conditioning electronics integrated into the sensor body. Included in the electronics are amplifiers, an analog to digital converter, an EEProm containing calibration data and RS-485 serial drivers. The 50M31 outputs a 2 megabit per second serial data stream which contains complete 6 axis data at 8 kHz and can be read by any of ***JR3***'s serial receivers.

The 50M31 is machined from a solid billet of aluminum. The resulting monolithic structure produces a transducer with unsurpassed hysteresis and precision properties. The 50M31 uses metal foil strain gages bonded to strain rings as the sensing element. This produces linearity, and thermal performance superior to other strain sensing technologies and geometries. The mechanical structure of the 50M31 also produces a transducer with exceptional stiffness.

As can be seen on the interface drawing, the 50M31 has captive bolts which allow direct connection to the user's tool flange. The top of the transducer contains a copy of the tool flange. This feature eliminates the need for adapter plates, which saves weight and lowers the installed height.

The 50M31 can interface to a variety of receiver electronics. ***JR3*** has DSP-based receiver electronics available for several industry standard computer busses. These receivers are able to process the force and moment data at the full 8 kHz data rate. They provide decoupling, coordinate transformation, low-pass filtering, vector calculation, threshold monitoring, peak capture and rate calculations.

JR3 Model No. 50M31A-I25

Capacities, Resolutions, Stiffnesses and Permissible Single Axis Overloads:

25 lb Model:

Fx, Fy	Fz		Mx,My	Mz	
25	50	lbs	50	50	in-lbs
100	200	N	5.0	5.0	N-m
0.01	0.02	lbs	0.02	0.02	in-lbs
29	295	klbs/in	100	25	kin-lbs/rad
110	435	lbs	180	140	in-lbs

Weight:
0.3 lb (140 g)

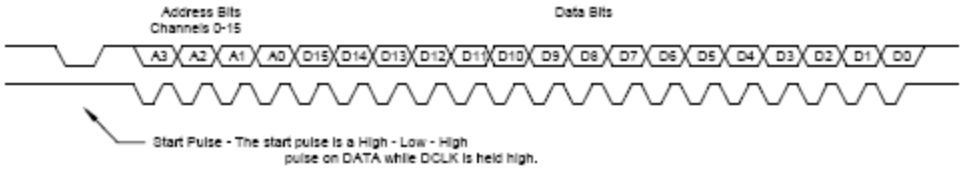
Available interfaces (bolt patterns) for 50M31:
ISO25mm, Panasonic 27mm

JR3, Inc. has been designing and manufacturing six axis loadcells since 1983. We have manufactured six axis loadcells ranging in diameter from 2 to 13 inches, and with load capacities from 2 to 25,000 lbs and 0.25 to 22,000 ft-lbs. Please feel free to call our applications engineers to discuss your particular needs.

Title:	Drawing #	Rev:	Page
Serial Communication Protocol Overview	5960	-	1 of 2

Rev	Description	Date	Approved
-	Initial Release		

Overview



The onboard electronics consists of voltage converters and regulators, amplifiers, a multiplexer, an Analog to Digital Converter (ADC), two RS-485 balanced pair transmitters, a 4k serial EEProm and a PLD. Power comes into the sensor using two or four wires in the sensor cable. The incoming power is converted and filtered to provide regulated +/- 5 volts or +/- 10 volts. Amplifiers are used to amplify and sum the strain gage signals. The summed signals are multiplexed and fed to the ADC. The ADC digitizes the signal with a resolution of 14 or 16 bits. The ADC outputs data as a serial stream. The serial stream has calibration and address information added by the PLD. The calibration information is stored in the EEProm. The serial stream is then sent out the sensor cable by the RS-485 transmitters.

DATA and DCLK are driven by RS-485 differential drivers. They need to be routed as a pair for the best noise immunity. EXCITATION, GND, and occasionally +/- 12V, are the power for the sensor. Although voltages are regulated in the sensor, it is important to regulate the voltage going to the sensor to limit power dissipation in the sensor. The receiver board powers EXCITATION at an adjusted voltage of between 7V and 9V based on channel 0 feedback. +/- 12V is not actively adjusted.

Data Packets

The data is packeted with 4 bits of address and 16 bits of data. The data is in a 2's complement bipolar form.

The baud rate on the DCLK and DATA is variable, DCLK is produced by the transmitter, and therefore can be any convenient rate. This assumes of course, that the receiver will be able to accept the data rate. Currently JR3 is designing receivers to handle DCLKs up to about 4 MHz.

The length of the data packet is approximately 24 DCLK periods with the start pulse included. This would allow about 165k packets per second. This bandwidth is well beyond what is currently needed for 6 axis load cells. Therefore, although DCLK can be 4 MHz, we are currently using a maximum DCLK of 2 MHz. Most sensors transmit 7 packets at 2 MHz (channels 0-6) and the eighth packet at 1 MHz (channel 7).

Title:	Drawing #	Rev:	Page
Serial Communication Protocol Overview	5960	-	2 of 2

Content of data packets

Currently most sensors are transmitting 8 channels (0-7), although 16 channels (0-15) can be transmitted. The channel is specified by the 4 address bits of the packet.

Channel 0 has been assigned to monitor the power voltage level at the sensor. This allows this voltage to be adjusted to accommodate various cable lengths and sensor voltage levels automatically. The feedback voltage should be maintained at 75% of the A/D full-scale (about 7.5V at the sensor).

Channels 1-6 are Fx, Fy, Fz, Mx, My & Mz respectively. With a packet rate of 64 KHz and 8 packets this sets each channel at about a 8 KHz maximum sample rate.

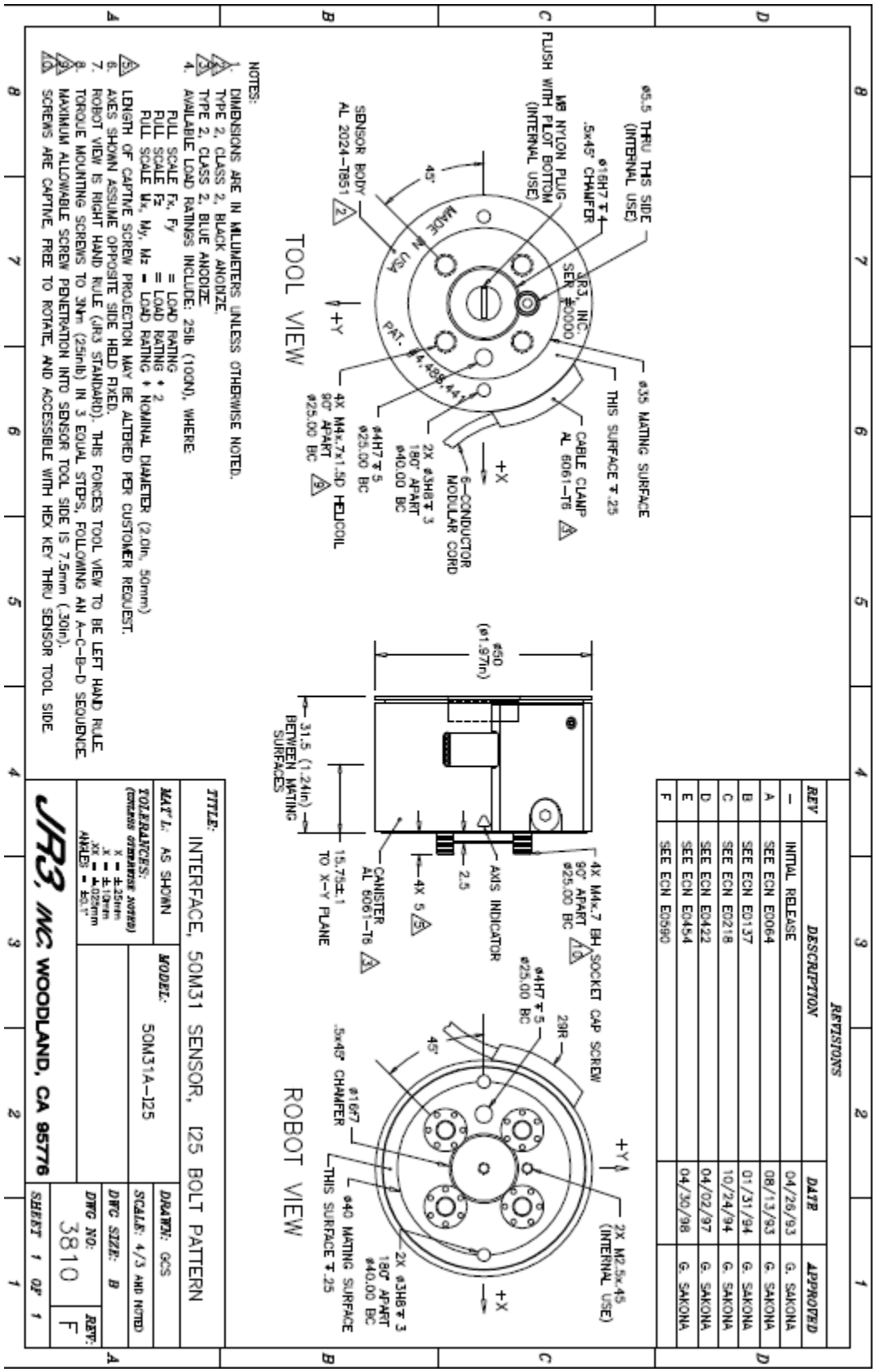
Channel 7 has been defined to carry calibration information. Most sensors transmit channel 7 at 1 MHz. Calibration information is stored in an EEProm in the sensor and transmitted one address / value pair per packet with the address as the high-order byte and the value as the low-order byte. (see JR3 Drawing #5912 for the layout of the EEProm)

Channels 8-15 are currently reserved and are only transmitted on select products. A packet rate of 64 KHz and 16 packets would set each channel at about a 4 KHz maximum sample rate.

Hardware

Our force sensor receiver board has a 10-pin modular jack which accepts either the 6-pin, 8-pin or 10-pin plug without mishap. We use 6-pin, 8-pin and 10-pin jacks on our sensors. Virginia Plastics makes jacks with contacts that do not deform when a 6-pin or 8-pin plug is inserted into a 10-pin jack.

	Receiver			Sensor		
	6-pin	8-pin	10-pin	6-pin	8-pin	10-pin
reserved	-	-	1	-	-	10
+12V	-	1	2	-	8	9
DATA-	1	2	3	6	7	8
DATA+	2	3	4	5	6	7
GND	3	4	5	4	5	6
EXCITATION	4	5	6	3	4	5
DCLK-	5	6	7	2	3	4
DCLK+	6	7	8	1	2	3
-12V	-	8	9	-	1	2
EARTH	-	-	10	-	-	1


TOOL VIEW
ROBOT VIEW
NOTES:
1. DIMENSIONS ARE IN MILLIMETERS UNLESS OTHERWISE NOTED.
2. TYPE 2, CLASS 2, BLACK ANODIZE.
3. TYPE 2, CLASS 2, BLUE ANODIZE
4. AVAILABLE LOAD RATINGS INCLUDE: 25lb (100N), WHERE:
FULL SCALE Fx, Fy = LOAD RATING
FULL SCALE Fz = LOAD RATING * 2
FULL SCALE Mx, My, Mz = LOAD RATING * NOMINAL DIAMETER (2.0in, 50mm)
5. LENGTH OF CAPTIVE SCREW PROJECTION MAY BE ALTERED PER CUSTOMER REQUEST.
AXES SHOWN ASSUME OPPOSITE SIDE HELD FIXED.
6. ROBOT VIEW IS RIGHT HAND RULE (JR3 STANDARD). THIS FORCES TOOL VIEW TO BE LEFT HAND RULE.
7. TORQUE MOUNTING SCREWS TO 3Nm (25in-lb) IN 3 EQUAL STEPS, FOLLOWING AN A-C-B-D SEQUENCE.
8. MAXIMUM ALLOWABLE SCREW PENETRATION INTO SENSOR TOOL SIDE IS 7.5mm (.30in).
9. SCREWS ARE CAPTIVE, FREE TO ROTATE, AND ACCESSIBLE WITH HEX KEY THRU SENSOR TOOL SIDE.
REVISIONS
REV | DESCRIPTION | DATE | APPROVED
– | INITIAL RELEASE | 04/26/93 | G. SAKONA
A | SEE ECN E0064 | 08/13/93 | G. SAKONA
B | SEE ECN E0137 | 01/31/94 | G. SAKONA
C | SEE ECN E0218 | 10/24/94 | G. SAKONA
D | SEE ECN E0422 | 04/02/97 | G. SAKONA
E | SEE ECN E0454 | 04/30/98 | G. SAKONA
F | SEE ECN E0590
TITLE: INTERFACE, 50M31 SENSOR, I25 BOLT PATTERN
MATL: AS SHOWN
MODEL: 50M31A-125
DRAWN: GCS
SCALE: 4/3 AND NOTED
DWG SIZE: B
DWG NO: 3810
REV: F
SHEET 1 OF 1
JR3, INC. WOODLAND, CA 95776

A.5 Microcontrolador PIC18F2580

MICROCHIP **PIC18F2480/2580/4480/4580**

28/40/44-Pin Enhanced Flash Microcontrollers with ECAN™ Technology, 10-Bit A/D and nanoWatt Technology

Power Managed Modes:

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

Flexible Oscillator Structure:

- Four Crystal modes, up to 40 MHz
- 4X Phase Lock Loop (PLL) – available for crystal and internal oscillators)
- Two External RC modes, up to 4 MHz
- Two External Clock modes, up to 40 MHz
- Internal oscillator block:
  - 8 user selectable frequencies, from 31 kHz to 8 MHz
  - Provides a complete range of clock speeds, from 31 kHz to 32 MHz when used with PLL
  - User tunable to compensate for frequency drift
- Secondary oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor
  - Allows for safe shutdown if peripheral clock stops

Special Microcontroller Features:

- C compiler optimized architecture with optional extended instruction set
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Flash/Data EEPROM Retention: > 40 years
- Self-programmable under software control
- Priority levels for interrupts
- 8 x 8 Single Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 41 ms to 131s
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Wide operating voltage range: 2.0V to 5.5V

Peripheral Highlights:

- High current sink/source 25 mA/25 mA
- Three external interrupts
- One Capture/Compare/PWM (CCP) module
- Enhanced Capture/Compare/PWM (ECCP) module (40/44-pin devices only):
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI™ (all 4 modes) and I^2C™ Master and Slave modes
- Enhanced Addressable USART module
  - Supports RS-485, RS-232 and LIN 1.3
  - RS-232 operation using internal oscillator block (no external crystal required)
  - Auto-Wake-up on Start bit
  - Auto-Baud detect
- 10-bit, up to 11-channel Analog-to-Digital Converter module (A/D), up to 100 Ksps
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual analog comparators with input multiplexing

ECAN Module Features:

- Message bit rates up to 1 Mbps
- Conforms to CAN 2.0B ACTIVE Specification
- Fully backward compatible with PIC18XXX8 CAN modules
- Three modes of operation:
  - Legacy, Enhanced Legacy, FIFO
- Three dedicated transmit buffers with prioritization
- Two dedicated receive buffers
- Six programmable receive/transmit buffers
- Three full 29-bit acceptance masks
- 16 full 29-bit acceptance filters w/ dynamic association
- DeviceNet™ data byte filter support
- Automatic remote frame handling
- Advanced error management features

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI™	Master I^2C™			
PIC18F2480	16K	8192	768	256	25	8	1/0	Y	Y	1	0	1/3
PIC18F2580	32K	16384	1536	256	36	8	1/0	Y	Y	1	0	1/3
PIC18F4480	16K	8192	768	256	25	11	1/1	Y	Y	1	2	1/3
PIC18F4580	32K	16384	1536	256	36	11	1/1	Y	Y	1	2	1/3

A.6 Transceiver MAX485

19-0122; Rev 7; 6/03

MAXIM

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

General Description

The MAX481, MAX483, MAX485, MAX487–MAX491, and MAX1487 are low-power transceivers for RS-485 and RS-422 communication. Each part contains one driver and one receiver. The MAX483, MAX487, MAX488, and MAX489 feature reduced slew-rate drivers that minimize EMI and reduce reflections caused by improperly terminated cables, thus allowing error-free data transmission up to 250kbps. The driver slew rates of the MAX481, MAX485, MAX490, MAX491, and MAX1487 are not limited, allowing them to transmit up to 2.5Mbps.

These transceivers draw between 120µA and 500µA of supply current when unloaded or fully loaded with disabled drivers. Additionally, the MAX481, MAX483, and MAX487 have a low-current shutdown mode in which they consume only 0.1µA. All parts operate from a single 5V supply.

Drivers are short-circuit current limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if the input is open circuit.

The MAX487 and MAX1487 feature quarter-unit-load receiver input impedance, allowing up to 128 MAX487/MAX1487 transceivers on the bus. Full-duplex communications are obtained using the MAX488–MAX491, while the MAX481, MAX483, MAX485, MAX487, and MAX1487 are designed for half-duplex applications.

Applications

Low-Power RS-485 Transceivers

Low-Power RS-422 Transceivers

Level Translators

Transceivers for EMI-Sensitive Applications

Industrial-Control Local Area Networks

Features

- **In µMAX Package: Smallest 8-Pin SO**
- **Slew-Rate Limited for Error-Free Data Transmission (MAX483/487/488/489)**
- **0.1µALow-Current Shutdown Mode (MAX481/483/487)**
- **Low Quiescent Current:**
  120µA (MAX483/487/488/489)
  230µA (MAX1487)
  300µA (MAX481/485/490/491)
- **-7V to +12V Common-Mode Input Voltage Range**
- **Three-State Outputs**
- **30ns Propagation Delays, 5ns Skew (MAX481/485/490/491/1487)**
- **Full-Duplex and Half-Duplex Versions Available**
- **Operate from a Single 5V Supply**
- **Allows up to 128 Transceivers on the Bus (MAX487/MAX1487)**
- **Current-Limiting and Thermal Shutdown for Driver Overload Protection**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX481CPA	0°C to +70°C	8 Plastic DIP
MAX481CSA	0°C to +70°C	8 SO
MAX481CUA	0°C to +70°C	8 µMAX
MAX481C/D	0°C to +70°C	Dice*

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

PART NUMBER	HALF/FULL DUPLEX	DATA RATE (Mbps)	SLEW-RATE LIMITED	LOW-POWER SHUTDOWN	RECEIVER/ DRIVER ENABLE	QUIESCENT CURRENT (µA)	NUMBER OF TRANSMITTERS ON BUS	PIN COUNT
MAX481	Half	2.5	No	Yes	Yes	300	32	8
MAX483	Half	0.25	Yes	Yes	Yes	120	32	8
MAX485	Half	2.5	No	No	Yes	300	32	8
MAX487	Half	0.25	Yes	Yes	Yes	120	128	8
MAX488	Full	0.25	Yes	No	No	120	32	8
MAX489	Full	0.25	Yes	No	Yes	120	32	14
MAX490	Full	2.5	No	No	No	300	32	8
MAX491	Full	2.5	No	No	Yes	300	32	14
MAX1487	Half	2.5	No	No	Yes	230	128	8

A.7 Transceiver MCP2551

MICROCHIP

MCP2551

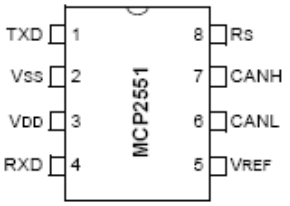
High-Speed CAN Transceiver

Features

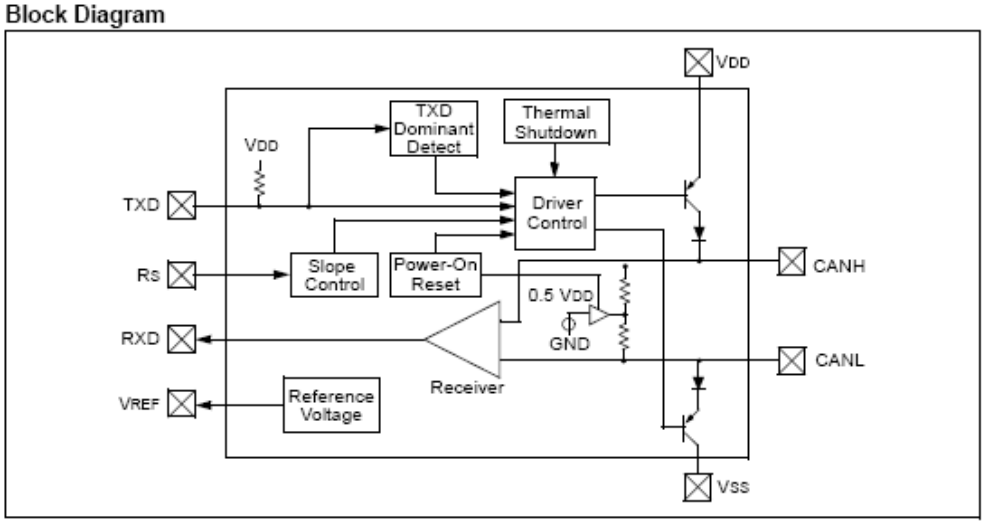
- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent Dominant) on TXD input
- Power-on Reset and voltage brown-out protection
- An unpowered node or brown-out event will not disturb the CAN bus
- Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
- Protection against high-voltage transients
- Automatic thermal shutdown protection
- Up to 112 nodes can be connected
- High-noise immunity due to differential bus implementation
- Temperature ranges:
  - Industrial (I): -40°C to +85°C
  - Extended (E): -40°C to +125°C

Package Types

PDIP/SOIC



Block Diagram

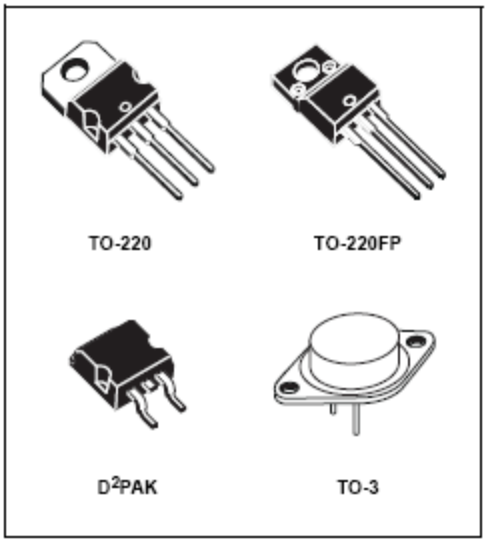


A.7 Regulador de tension L7800

L7800 SERIES

POSITIVE VOLTAGE REGULATORS

- OUTPUT CURRENT TO 1.5A
- OUTPUT VOLTAGES OF 5; 5.2; 6; 8; 8.5; 9; 12; 15; 18; 24V
- THERMAL OVERLOAD PROTECTION
- SHORT CIRCUIT PROTECTION
- OUTPUT TRANSITION SOA PROTECTION



DESCRIPTION

The L7800 series of three-terminal positive regulators is available in TO-220, TO-220FP, TO-3 and D²PAK packages and several fixed output voltages, making it useful in a wide range of applications. These regulators can provide local on-card regulation, eliminating the distribution problems associated with single point regulation. Each type employs internal current limiting, thermal shut-down and safe area protection, making it essentially indestructible. If adequate heat sinking is provided, they can deliver over 1A output current. Although designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltage and currents.

SCHEMATIC DIAGRAM