UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN ENCODER ABSOLUTO CANopen

AUTOR: JULIAN FORTES MONTEIRO

TUTOR: ALBERTO JARDÓN HUETE

JULIO 2009

Título: DISEÑO E IMPLEMENTACIÓN ENCODER ABSOLUTO CANopen.

Autor: JULIAN FORTES MONTEIRO.

Tutor: ALBERTO JARDÓN HUETE.

Rama: I.T.I. ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL.

La defensa del presente Proyecto Fin de Carrera se realizó el día 23 de 07 de 2009; siendo calificada por el siguiente tribunal:

Presidente: Santiago Martínez de la Casa Díaz.

Secretario: Paolo Pierro.

Vocal: Agapito Ledezma Espino.

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente Secretario Vocal

RESUMEN.

A pesar de que en los últimos años los dispositivos CANopen (y en particular los encoders absolutos) han tenido una gran evolución en el mercado, estos han sido diseñados para trabajar en entornos industriales en los que prima la robustez. Lo que se traduce en la introducción de carcasas protectoras que aumentan considerablemente el tamaño y peso de los dispositivos.

Esta circunstancia ha dado lugar al diseño e implementación de un encoder absoluto que cumpla con todas las especificaciones del protocolo CANopen sin descuidar las características anteriormente mencionadas, es decir, se ha diseñado un dispositivo de tamaño reducido y muy ligero.

Para el desarrollo del proyecto se ha partido de un encoder absoluto comercial que facilita la lectura de posición a través de un puerto SPI a un microcontrolador y posteriormente estos datos son tratados y transmitidos a través de un Bus CAN, para lo cual se ha diseñado e implementado toda la electrónica necesaria.

El diseño software se ha realizado partiendo de la implementación del protocolo CANopen de Microchip® que proporciona un soporte en forma de librerías de programación que facilitan el desarrollo de estos sistemas. El proyecto incluye una guía práctica de estas librerías para el uso y desarrollo de sistemas CANopen.

El encoder CANopen realizado se ha puesto en marcha y validado el funcionamiento en dos plataformas robóticas del departamento de Ingeniería en Sistemas y Automática de la Universidad Carlos III de Madrid.

ABSTRACT.

Even though in the last few years the CANopen mechanisms (in particular the absolute encoders) have advanced greatly in the market, they have been designed for use in industrial environments where robustness is priority. This is translated in the introduction of protective casings that increase considerably the size and weight of the mechanisms.

This circumstance has given place to the designing and implementation of an absolute encoder that meets all the specifications of the CANopen protocol without disregarding the previously mentioned characteristics, i.e. a smaller and very light mechanism has been designed.

For the project development, a commercial absolute encoder has been designed that makes the reading position through a SPI port and microcontroller easier, subsequently this data is treated and transmitted through a Bus CAN, for which all the necessary electronics have been designed and implemented.

The design of the software has been executed from the implementation of the CANopen Microchip® protocol that provides a library shape support of programming that makes the development of these systems easier. The project includes a practical guide of these libraries for the use and development of CANopen systems.

The successful CANopen encoder has been put into operation and validated on two robotic platforms in the Systems and Automatic Engineering department in the Carlos III University in Madrid.

A Gabriel,

¡Te quiere papa!

AGRADECIMIENTOS.

He de expresar mi profundo agradecimiento a todas aquellas personas que me han apoyado durante estos últimos años y cuya influencia ha sido de vital importancia para poder llegar al punto en el que me encuentro, finalizando una carrera y el proyecto culmen de la misma.

No puedo nombrar a todos, pero si quiero reconocer específicamente el valor a algunos de ellos:

En primer lugar y como no, a mis padres que siempre me han animado y respaldado para que este gran día llegara. A todos mis hermanos y en especial a Isa por estos años de convivencia.

A mi novia Vanesa, por su paciencia, por estar ahí siempre que la he necesitado y por ese precioso hijo que me ha dado.

A Kate por su eficaz y profesional ayuda con el ingles.

A todos los compañeros de la universidad que se subieron al carro conmigo y que han estado a mi lado tanto en los buenos, como en los malos momentos… María Jesús por esos interminables días en la biblioteca. A la peña de Tenerife por todos esos grandes momentos que hemos compartido y sin los que el paso por la universidad no hubiera sido lo mismo. Y como iba a olvidarme de Javi y Vero, a los que quiero recordar que la unión hace la fuerza.

A todo el laboratorio 1.3.C.13 por su ayuda y en especial a Javier Quijano y a Conrrado por hacer del laboratorio un lugar agradable.

Por último quiero hacer una mención especial a mi tutor Alberto Jardón por darme la oportunidad de realizar el proyecto y formar parte del departamento de Ingeniería en Sistemas y Automática.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1

INTRODUCCIÓN.

1. INTRODUCCIÓN.

Actualmente las líneas de investigación en el campo de la robótica se están orientando al área de servicios tales como asistencia personal, educación, tareas sociales, etc. Los robots móviles actuales no están adaptados para ser utilizados en entornos domésticos, debido a su volumen y a su falta de maniobrabilidad en un entorno tan complejo como son estos escenarios por lo que se está trabajando en nuevas líneas de diseño. Por supuesto, hay ciertas excepciones, como es el caso del exitoso robot aspiradora Roomba de pequeño volumen, lo que le facilita su movilidad [17].

En este campo han surgido muchos proyectos internacionales de desarrollo de robots humanoides adaptados a entornos domésticos o de oficinas que trabajen como robots de servicio, puesto que pueden llegar a tener mejor maniobrabilidad y facilidad para moverse en entornos con superficies irregulares.

El problema de los robots humanoides es que tienen un alto número de grados de libertad, lo que implica en la mayoría de los casos un peso elevado introducido en gran parte por los motores, actuadores y reductores necesarios por cada grado de libertad, además de suponer un coste elevado. Para solventar estos problemas se pretende dar un salto cualitativo, desarrollando robots bípedos ligeros, conservando los grados de libertad de un robot humanoide pero reduciendo el número de actuadores de tal forma que estos sean inferior al número de articulaciones que posea basándose en técnicas de control de dinámica pasiva que permitan desarrollar un paso adecuado.

1.1 Motivaciones.

Basándose en estas nuevas líneas de investigación, la Universidad Calos III de Madrid, y en concreto los **Departamentos de Ingeniería de Sistemas y Automática** e **Ingeniería Mecánica** han decidido crear PASIBOT, un nuevo robot que servirá como plataforma de investigación y ayuda a otros proyectos de investigación que están siendo desarrollados, como son los robots humanoides y robots de asistencia a discapacitados. Este proyecto es pionero en España y trata

de abrir nuevas líneas de investigación en consonancia con estas técnicas de diseño. En la siguiente figura se puede observar el diseño de PASIBOT y el esbozo geométrico de una pierna [6].

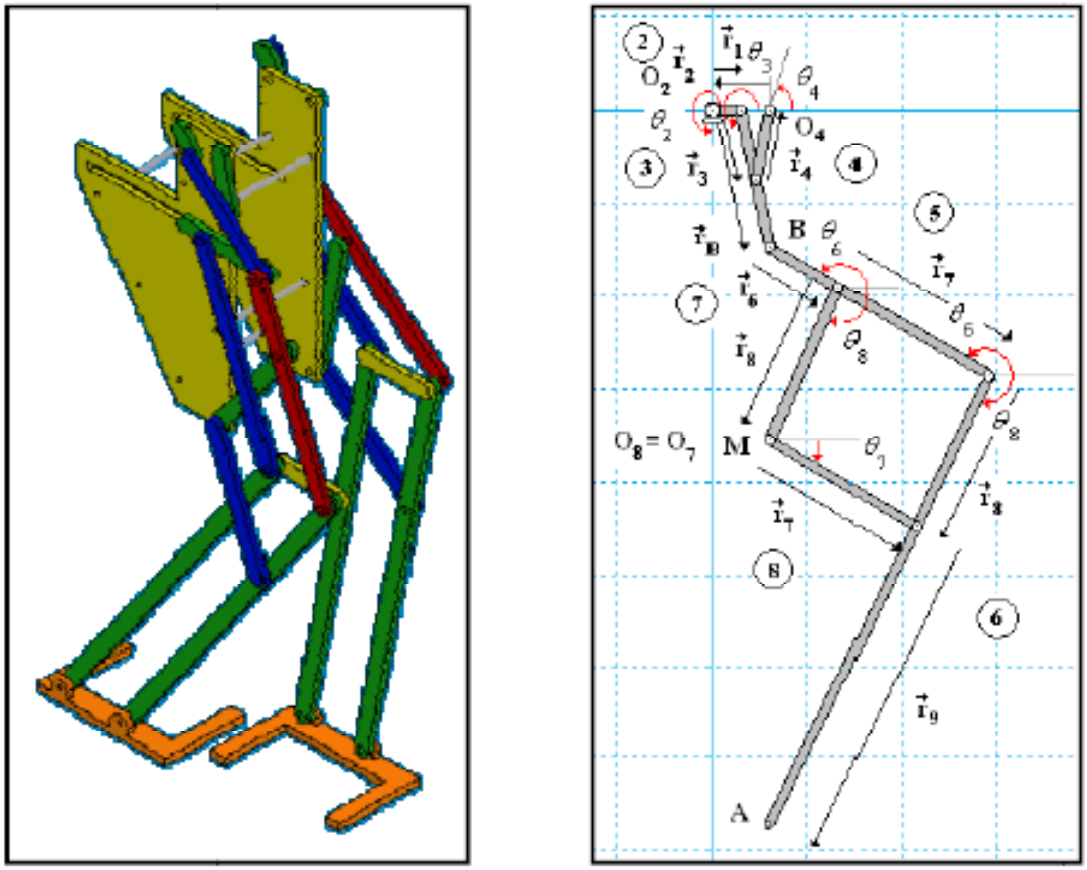


Figura 1: Diseño de PASIBOT y Esbozo Geométrico de una Pierna.

PASIBOT es un prototipo intermedio entre un robot pasivo y un bípedo al que se pretende hacer caminar con técnicas de caminar pasivo [6]. Este concepto implica que el número de actuadores a controlar es bajo. En concreto, se pretende hacer caminar a PASIBOT con un solo actuador, controlado por un encoder absoluto, un encoder relativo y un inclinómetro.

Uno de los primeros problemas que surgen en cualquier diseño actual, es que vivimos en un entorno que evoluciona constantemente y esto se acentúa más cuando hablamos del mundo de la tecnología. Por lo que no es suficiente dar solución a los problemas que se nos presenta en cada momento, sino que tenemos que pensar en futuras modificaciones o ampliaciones de los sistemas que pretendemos desarrollar. Esta filosofía permite adaptar fácilmente un sistema

obsoleto a nuevas necesidades.

Por lo expuesto anteriormente, surge la necesidad de plantearse cual es el medio más adecuado para interconectar los distintos componentes del sistema que se desea desarrollar teniendo en cuenta posibles ampliaciones del sistema o incluso la sustitución de alguno de los componentes ya existentes. Por lo que se ha pensado en un sistema estándar, flexible y robusto, en cuanto a configuración, sustitución y ampliaciones, como son los sistemas basados en CAN-Bus.

CAN-Bus es un sistema estandarizado de comunicación serie muy potente y que permite una gran flexibilidad a la hora de sustituir e introducir nuevos componentes a la red. Este sistema sólo cubre las necesidades hardware de sistema, por lo que es necesario introducir un protocolo de comunicaciones a nivel software. Existen varios protocolos estandarizados de nivel software como son el CANopen, DeviceNet, SAE J1939 o CANKingdom.

A la hora de seleccionar el protocolo adecuado se han tenido en cuenta criterios como la velocidad de transferencia, integración y disponibilidad de componentes en el mercado, llegando a la conclusión de que el más apropiado es el CANopen.

1.2 Objetivos del proyecto.

A pesar de que en los últimos años los dispositivos CANopen han tenido una gran evolución en el mercado, a la hora de seleccionar un encoder absoluto que se adapte a las necesidades de nuestro diseño nos encontramos con un problema. Este problema está originado porque la mayoría de los fabricantes orientan sus diseños hacia la industria, donde lo que se pretende es conseguir un sistema robusto y eficaz, dando menor importancia al tamaño y peso del componente que en la mayoría de los casos no suele ser un factor determinante. Esto provoca que a la hora de buscar un encoder absoluto que funcionalmente cumpla el protocolo CANopen, nos encontremos con dispositivos con un volumen y peso considerable, debido principalmente a las carcasas protectoras introducidos en estos diseños para solventar los problemas que supone trabajar en los entornos industriales.

En nuestro caso las características que priman, sin dejar de lado la robustez y

eficacia de los Nodos CAN, son un diseño que estructuralmente sea de reducido tamaño, ligero y fácilmente integrable en el diseño de PASIBOT.

Por consiguiente, el objetivo de este proyecto es el diseño de un encoder absoluto CANopen que se integrara en PASIBOT para el control de la posición de su único actuador. Dicho encoder ha de cumplir con las especificaciones descritas por CiA (CAN in Automation) teniéndose en cuenta el documento "DS-301; Application Layer and Communication Profile" [2], que describe las características comunes de cualquier dispositivo CANopen, y el "DS-406; Device profile for encoders" [4], que describe el perfil específico de un encoder CANopen. Este encoder absoluto además de cumplir con las características básicas de este componente según el perfil específico (encoder absoluto de tipo C1), ha de tener muy en cuenta el tamaño y peso del mismo.

Para el diseño del encoder absoluto CANopen se ha partido de un encoder absoluto comercial compuesto por lector y disco codificado que se integra en el diseño mecánico de PASIBOT, como se puede observar en la siguiente figura.

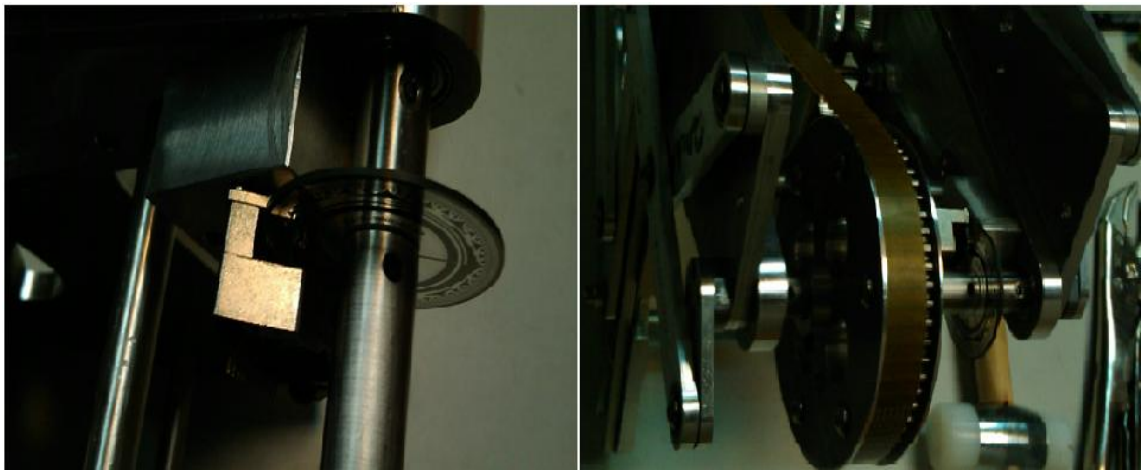


Figura 2: Imagen Encoder Absoluto en PASIBOT.

El encoder absoluto comercial seleccionado es muy ligero, de tamaño reducido y facilita la lectura de posición por un puerto SPI. Estos datos serán recogidos por un microcontrolador a través de dicho puerto y una vez tratados se enviaran a un conjunto de drivers CAN que se encargaran de acondicionar los datos para posteriormente envíalos al Bus CAN. En la siguiente figura se muestra el esquema

inicialmente planteado para el diseño del Nodo CANopen, además del concepto básico de adquisición y transmisión de datos.

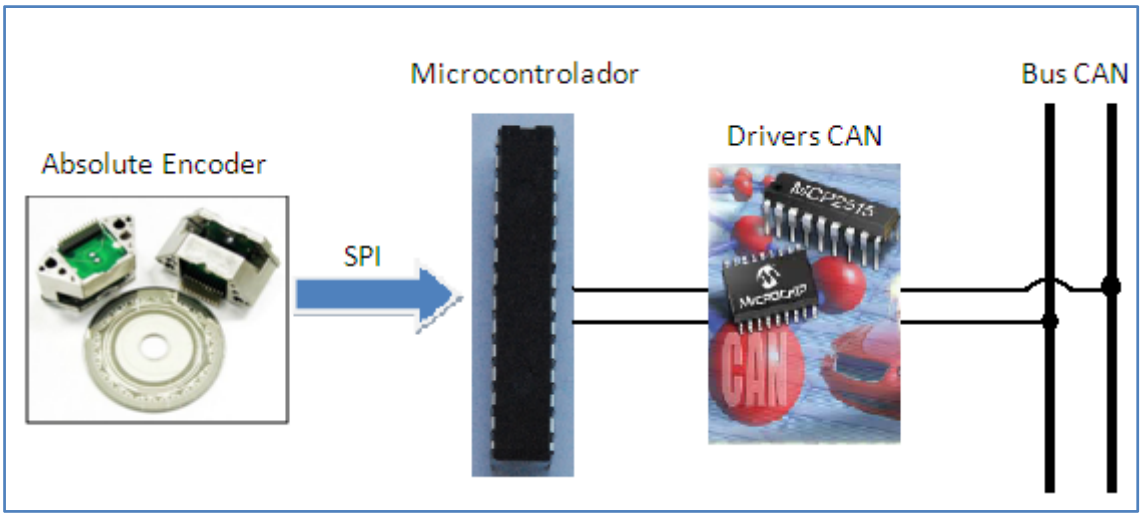


Figura 3: Esquema inicial del diseño.

Para la implementación del protocolo CANopen se usaran componentes de Microchip® por su gran variedad y disponibilidad en el mercado, además de por la existencia de herramientas de diseño muy potentes y entornos de desarrollo adecuados para diseñar sistemas CAN.

1.3 Estructura del documento.

Este documento se dividirá en ocho capítulos de los cuales, este primero se ha dedicado ha introducir el marco en el cual se desarrolla el proyecto, así como las motivaciones que dieron lugar al diseño del encoder absoluto CANopen y objetivos que inicialmente se pretenden alcanzar.

El segundo capítulo está dedicado a explicar todo lo relacionado con el protocolo CAN-Bus , incluyendo la topología y dispositivos necesarios para crear un Nodo basado en dicho Bus, ya que el protocolo CANopen aprovecha la estructura y diseño del CAN-Bus para implementar su modelo de comunicación. También se incluirá una descripción básica de las tramas enviadas al bus, que puede ser ampliada consultando el documento “BOSCH Controller Area Network protocol specification” [1] y se comentaran ciertas herramientas de diagnostico del Bus.

En el tercer capítulo se describirá el protocolo de comunicaciones CANopen, centrándose especialmente en los servicios que permiten implementar un dispositivo CANopen para posteriormente explicar, en el capítulo cuatro, como se ha desarrollado el diseño del encoder absoluto CANopen tanto a nivel *hardware*, teniendo que realizar una placa de circuito impreso, como a nivel *software*, usando la pila CANopen de Microchip® [18].

En el quinto capítulo se explicarán los trabajos experimentales llevados a cabo para la comprobación del correcto funcionamiento del dispositivo diseñado y en el capítulo sexto 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 capítulos séptimo y octavo para introducir la bibliografía del documento y los anexos respectivamente.

Capítulo 2.

PROTOCOLO CAN-Bus.

2. PROTOCOLO CAN-Bus.

CAN-Bus es un protocolo de comunicación serie para el intercambio de información entre unidades de control electrónico que soporta un eficiente control en **Tiempo Real** con un nivel de fiabilidad muy elevado. Fue desarrollado inicialmente por Bosch en 1983 para la industria de la automoción y está estandarizado desde 1993 por la norma ISO 11898-1.

La siglas CAN significa Controller Area Network (Red de controladores locales) y Bus se entiende como un elemento de comunicaciones que permite transportar una gran cantidad de información entre las unidades de control abonadas a un sistema.

El planteamiento del CAN-Bus permite aumentar las funciones presentes en un sistema de control sin modificar los nodos ya existentes, además de que estas funciones pueden estar repartidas entre las distintas unidades de control. Como puede deducirse, permite disminuir notablemente el cableado en cualquier sistema de control, puesto que si una unidad de mando dispone de una información, como por ejemplo, la posición de un encoder, esta puede ser utilizada por el resto de unidades de mando sin que sea necesario que cada una de ellas reciba la información directamente de dicho sensor.

Otra ventaja obvia, además de que las funciones pueden ser repartidas entre distintas unidades de mando, es que incrementar las funciones de las mismas no presupone un coste adicional excesivo.

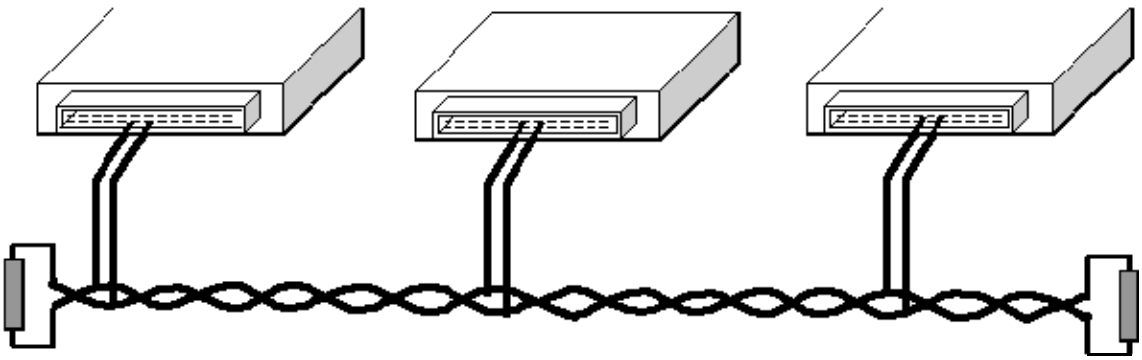


Figura 4: Sistema CAN-BUS.

2.1 Principales características del protocolo CAN-Bus.

Para garantizar un diseño transparente y una implementación flexible CAN ha sido subdividido en diferentes capas teniendo como referencia el modelo de comunicaciones ISO/OSI según la siguiente figura:

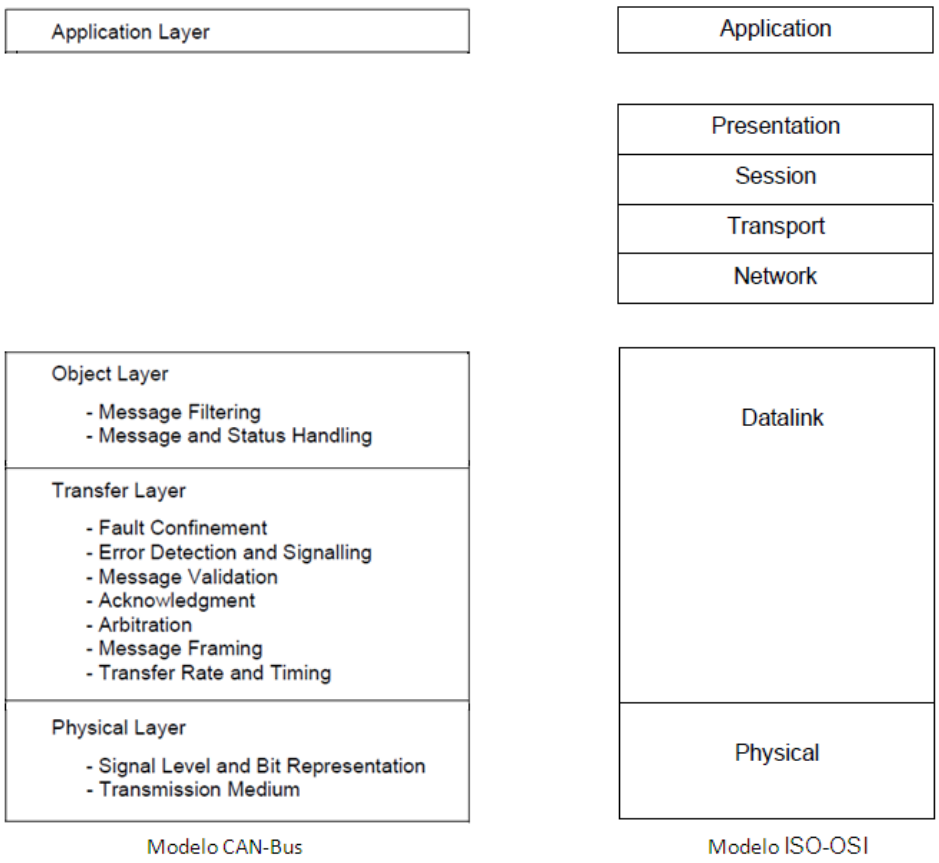


Figura 5: Comparación Entre el Modelo CAN y el Modelo ISO/OSI.

Las capas de objeto y comunicaciones cumplen con todos los servicios y funciones de la capa de lincado de datos definida en el modelo ISO/OSI. Es decir, el alcance de la capa de objetos es:

- Decidir que mensajes han de ser enviados.
- Decidir que mensajes recibidos están siendo utilizadas por el nodo.
- Proveer un interfaz a la capa de aplicación que controla el hardware.

La "Transfer layer" se encarga principalmente del protocolo de comunicaciones, por ejemplo, controlando la estructura del mensaje, chequear y señalizar errores,

etc. Dentro de esta capa se decide si el Bus esta libre para usarlo o si ha comenzado una transmisión. Esta capa es transparente para el usuario y no admite modificaciones.

La “Physical layer” es el medio físico por el cual se transmiten los datos entre los diferentes nodos del sistema. La única restricción que hay en esta capa es que todos los nodos del sistema han de utilizar el mismo medio físico (par trenzado, fibra óptica, etc.)

CAN se basa en un modelo MASTER/SLAVE orientado al mensaje, por lo que la información que se transmite se descompone en tramas a las que se asocia un identificador. Las principales características del CAN son las siguientes:

- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Configuración flexible.
- Recepción multicast con sincronización de tiempos.
- Sistema de consistencia de datos muy robusto.
- Multimaster.
- Detección y señalización de errores.
- Retransmisión automática de mensajes erróneos.
- Distinción entre fallos temporales o permanentes y capacidad autónoma de desconectar nodos defectuosos.

Todos los nodos pueden ser transmisores y receptores, y la cantidad de los mismos abonados al sistema puede ser variable (dentro de unos límites). Si la situación lo exige, un nodo puede solicitar a otro una determinada información mediante uno de los campos del mensaje (trama remota o RTR).

Cualquier nodo puede introduce una trama en el Bus con la condición de que esté libre, si otro lo intenta al mismo tiempo el conflicto se resuelve por la prioridad del mensaje indicado por el identificador del mismo. El sistema está dotado de una serie de mecanismos que aseguran que el mensaje es trasmitido y recibido correctamente. Cuando un mensaje presenta un error, es anulado y vuelto a

trasmitir de forma correcta, de la misma forma un nodo con problemas avisa a los demás mediante el propio mensaje. Si la situación es irreversible, dicho nodo queda fuera de servicio pero el sistema sigue funcionando.

2.2 Elementos que componen el sistema CAN-Bus.

Medio físico:

El medio físico más común es el par trenzado, aunque se puede usar otros sistemas como la fibra óptica. Este une todas las unidades de control que forman el sistema y es por donde circula la información entre los distintos nodos. Esta información se trasmite por diferencia de tensión entre los dos cables, de forma que un valor alto de tensión representa un "1" o valor "recesivo" y un valor bajo de tensión representa un "0" o valor "dominante". La combinación adecuada de unos y ceros conforman la trama a trasmitir.

En uno de los cable 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 ó se derive a masa, el sistema trabajará con la señal de Low con respecto a masa. Y 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.

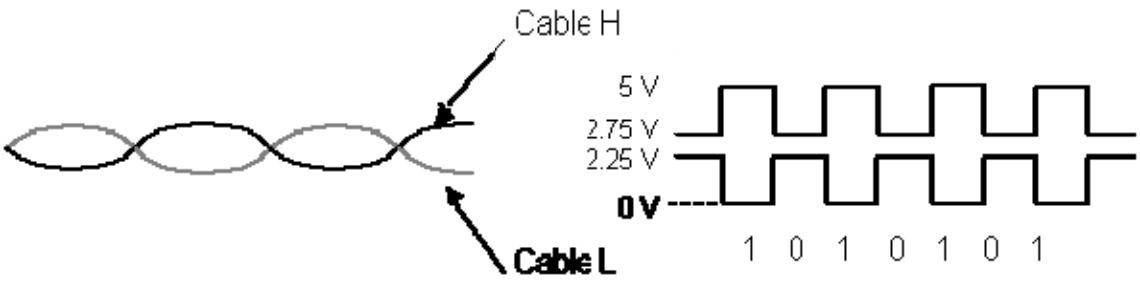


Figura 6: Niveles de tensión en el Bus.

El medio físico ha de estar dimensionado correctamente teniendo en cuenta los parámetros del cable seleccionado, ya que la resistencia de estos depende de la

sección, conductividad y longitud del mismo. Esta resistencia provoca una mayor caída de tensión en los cables a medida que aumenta, pudiendo provocar que los niveles de tensión entre los distintos nodos del sistema de control caiga por debajo del nivel mínimo que permita al nodo receptor interpretar la información correctamente.

Este parámetro es importante para un Bus de longitudes considerables, pudiendo omitir este en un Bus de reducidas dimensiones al considerar R_L despreciable.

En cuanto a la tasa de transferencia máxima de transmisión también hay que tener en cuenta el dimensionado del Bus ya que sobre este, influye la capacidad parasita que los cables introducen en el sistema y que delimitan la frecuencia máxima que es capaz de soportar el sistema de comunicaciones. La figura siguiente representa el circuito equivalente utilizado para dimensionar el Bus, donde:

R_L: Resistencia equivalente de cable.
C_L: Capacidad parasita introducida por el cable.
V_{out}: Tensión de salida del nodo emisor.
V_{int}: Tensión mínima de entrada en el nodo receptor.
Z_i: Impedancia de entrada.

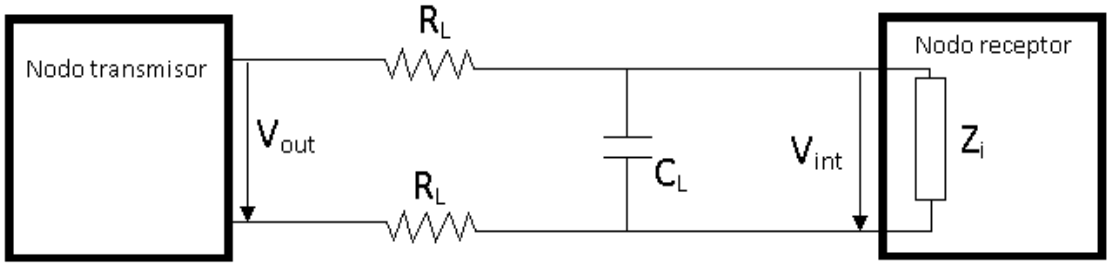


Figura 7: Circuito equivalente del Bus.

Otro de los aspectos a tener en cuenta en este apartado, son las interferencias electromagnéticas que se puedan producir en el entorno del sistema a controlar. Por lo que, como ya se ha comentado, el medio físico más utilizado es el par trenzado que minimiza este efecto sin tener que asumir el coste extra que supone la fibra óptica.

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 nodos abonados al sistema, ya que impiden fenómenos de reflexión que pueden perturbar el mensaje.

Microcontrolador:

Se encarga de gestionar y controlar todos los procesos de dispositivo y establece comunicación con el Bus a través del controlador CAN asociado al dispositivo.

Controlador:

Es el elemento encargado de la comunicación entre el microcontrolador del Nodo y el transmisor-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 los distintos Nodos para la correcta emisión y recepción de los mensajes

Transmisor / Receptor (Transceiver):

El transmisor-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 0.1 A y en ningún caso interviene modificando el contenido del mensaje. Tiene la misión de recibir y 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 para suminístrasela al controlador del dispositivo CAN.

Por otra parte, también proporciona protección frente a cortocircuitos y niveles de tensión peligrosos en el Bus. Funcionalmente está situado entre los cables que forman la línea CAN-Bus y el controlador CAN según se muestra en la siguiente figura.

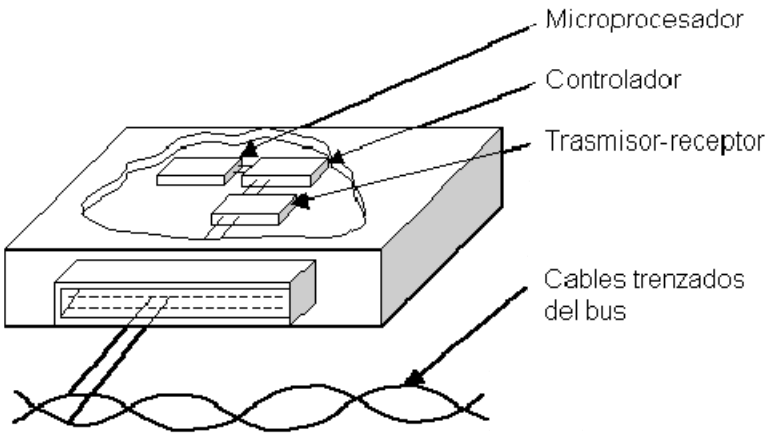


Figura 8: Componentes de un nodo.

2.3 Topología del CAN-Bus.

El sistema CAN-Bus está orientado hacía el mensaje y no al destinatario. La información en la línea es trasmitida en forma de mensajes estructurados en la que una parte del mismo es un identificador que indica la clase de dato que contiene. Todas las unidades de control reciben el mensaje, lo filtran y solo lo emplean las que necesitan dicho dato. Naturalmente, la totalidad de los nodos abonados al sistema son capaces tanto de introducir como de recoger mensajes de la línea. Cuando el bus está libre cualquier unidad conectada puede empezar a trasmitir un nuevo mensaje. En el caso de que una o varias unidades pretendan introducir un mensaje al mismo tiempo, lo hará la que tenga una mayor prioridad. Esta prioridad viene indicada por el identificador. El proceso de trasmisión de datos se desarrolla siguiendo un ciclo de varias fases:

1. **Suministro de datos:** Una unidad de mando recibe información de los sensores que tiene asociados (r.p.m., velocidad, posición, etc.) y su microprocesador pasa la información al controlador donde es gestionada y acondicionada para a su vez ser pasada al transmisor-receptor (transceiver)

donde se transforma en señales eléctricas.

2. **Trasmisión de datos:** El controlador de dicha unidad transfiere los datos y su identificador junto con la petición de inicio de trasmisión, asumiendo la responsabilidad de que el mensaje sea correctamente trasmitido a todos los nodos asociados al sistema. Para trasmitir el mensaje ha tenido que encontrar el bus libre, y en caso de colisión intentando trasmitir simultáneamente, tener una prioridad mayor. A partir del momento en que esto ocurre, el resto de unidades de mando se convierten en receptoras.

3. **Recepción del mensaje:** Cuando la totalidad de las unidades de mando reciben el mensaje, verifican el identificador para determinar si el mensaje va a ser utilizado por ellos. Las unidades de mando que necesiten los datos del mensaje lo procesan, si no lo necesitan, el mensaje es ignorado.

El sistema CAN-Bus dispone de mecanismos para detectar errores en la trasmisión de mensajes, de forma que todos los receptores realizan un chequeo del mensaje analizando una parte del mismo llamado campo CRC. Otros mecanismos de control que se aplican en las unidades emisoras son la monitorización del nivel del bus, la presencia de campos de formato fijo en el mensaje (verificación de la trama), análisis estadísticos por parte de las unidades de mando de sus propios fallos, etc. Estas medidas hacen que las probabilidades de error en la emisión y recepción de mensajes sean muy bajas, por lo que es un sistema extraordinariamente seguro. Según las especificaciones del protocolo de Bosch [1], la probabilidad de error residual de mensajes erróneos no detectado ha de ser inferior a $\mathbf{4.7 \times 10^{-11}}$.

2.4 Formato de las tramas CAN.

La información que circula entre las unidades de mando a través del puerto de comunicaciones serie son paquetes de bits con una longitud limitada y con una estructura definida de campos que conforman el mensaje. Uno de los campos actúa de identificador del tipo de dato que se transporta, del nodo que lo trasmite y de la prioridad para trasmitirlo respecto a otros. El mensaje no va direccionado a

ningún nodo en concreto, sino que cada uno de los nodos reconocerá mediante este identificador si el mensaje le interesa o no.

La estructura del menaje permite llevar a cabo el proceso de comunicación entre las unidades de mando según el protocolo definido por Bosch para el CAN-Bus, donde los distintos campos que lo compone facilitan desde identificar a la unidad de mando, como indicar el principio y el final del mensaje, mostrar los datos, permitir distintos controles, etc.

Los mensajes son introducidos en la línea con una cadencia que oscila entre los 0.055 y 12.9 milisegundos dependiendo de la velocidad del Bus y de la unidad de mando que los introduce.

Estructura del mensaje estándar:

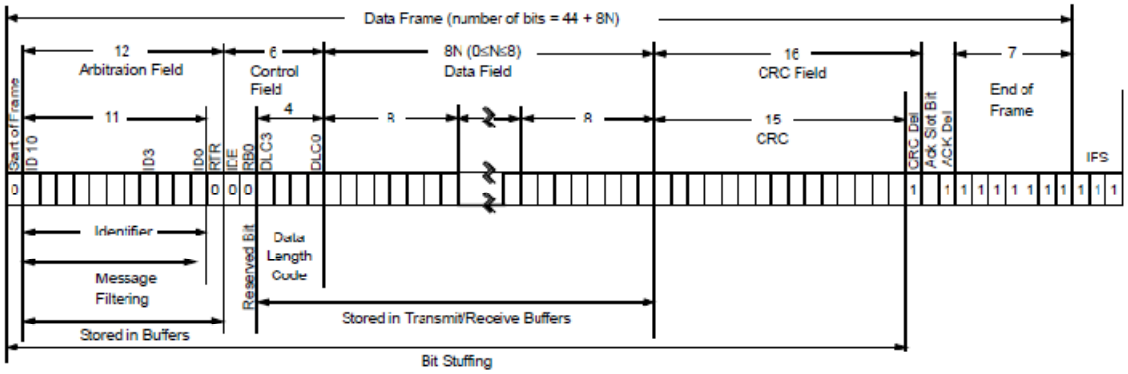


Figura 9: Estructura del mensaje estándar CAN.

- **Campo de inicio del mensaje (SOF)**: El mensaje se inicia con un bit dominante "0", cuyo flanco descendente es utilizado por las unidades de mando para sincronizarse entre sí.
- **Campo de arbitraje**: Los 12 bits de este campo se emplean como identificador y permite reconocer al Nodos que emite la trama, el tipo de mensaje y la prioridad de este. Cuanto más bajo sea el valor del identificador más alta es la prioridad, y por lo tanto determina el orden en el que van a ser introducidos los mensajes en la Bus.
- **El bit RTR:** Forma parte del campo de arbitraje é indica si el mensaje contiene datos (RTR=0) o si se trata de una trama remota sin datos

(RTR=1). Una trama de datos siempre tiene prioridad frente a las tramas remotas que se emplea para solicitar datos a otros nodos, bien porque se necesitan o para realizar un chequeo.

- **Campo de control**: Este campo informa sobre las características del campo de datos. El bit IDE indica cuando es un "0" que se trata de una trama estándar y cuando es un "1" que es una trama extendida. La diferencia entre una trama estándar y una trama extendida es que el identificador de la primera tiene 11 bits y la segunda 29 bits. Ambas tramas pueden coexistir eventualmente, siempre y cuando todos los controladores del sistema soporten el formato extendido, y la razón de su presencia es la existencia de dos versiones de CAN [1].
- **Campo DLC:** Los cuatro bit que lo componen indican el número de bytes contenido en el campo de datos.
- **Campo de datos**: En este campo aparece la información del mensaje con los datos que la unidad de mando correspondiente introduce en la línea CAN-Bus. Puede contener entre 0 y 8 bytes (de 0 a 64 bit).
- **Campo de chequeo (CRC)**: Este campo tiene un formato predefinido con una longitud de 16 bits de los cuales los 15 primeros son utilizados para la detección de errores, mientras que el último siempre es un bit recesivo "1" que delimita el campo CRC.
- **Campo de confirmación (ACK)**: El campo ACK está compuesto por dos bit que son siempre trasmitidos como recesivos "1". Todos los nodos que reciben el mismo CRC modifican el primer bit del campo ACK por uno dominante "0", de forma que la unidad de mando que está todavía trasmitiendo reconoce que al menos uno de los Nodo conectados al sistema ha recibido el mensaje correctamente. De no ser así, el Nodo transmisor interpreta que su mensaje presenta un error.
- **Campo de final de mensaje (EOF)**: Este campo indica el final del mensaje con una cadena de 7 bits recesivos.
- **Interframe space (IFS):** Está compuesto de al menos tres bits recesivos llamados ***intermission*** y su función es la de separar dos tramas consecutivas. Esto permite el procesado interno de los mensajes antes de

recibir una nueva trama. Después del IFS el bus permanece en un estado recesivo indicando que está libre para una nueva transmisión.

Puede ocurrir que en determinados mensajes se produzcan largas cadenas de ceros o unos, y que esto provoque una pérdida de sincronización entre los distintos Nodos. El protocolo CAN resuelve esta situación insertando un bit de diferente polaridad cada cinco bits iguales: cada cinco "0" se inserta un "1" y viceversa. El Nodo que utiliza el mensaje, descarta un bit posterior a cinco bits iguales. Estos bits reciben el nombre de bit ***stuffing***.

2.5 Filtrado de mensajes CAN.

El CAN-Bus está basado en un concepto de comunicación ***Broadcast***, lo que significa que cualquier Nodo perteneciente a la red pueden escuchar todos los mensajes que se transmiten por ella. Después de recibir un mensaje cada nodo ha de decidir si este es aceptado o no, y por eso es necesario implementar filtros de aceptación en cada uno de los nodos. En la figura siguiente se puede observar como el nodo 2 transmite un mensaje y los demás nodos lo reciben. El Nodo 3 desecha el mensaje debido a que el filtro no coincide con el identificador de la trama recibida, mientras que los nodos 1 y 4 si lo aceptan, por lo que han de procesar el contenido.

También se puede observar que el Nodo que transmite la trama monitoriza el Bus para chequeo de errores.

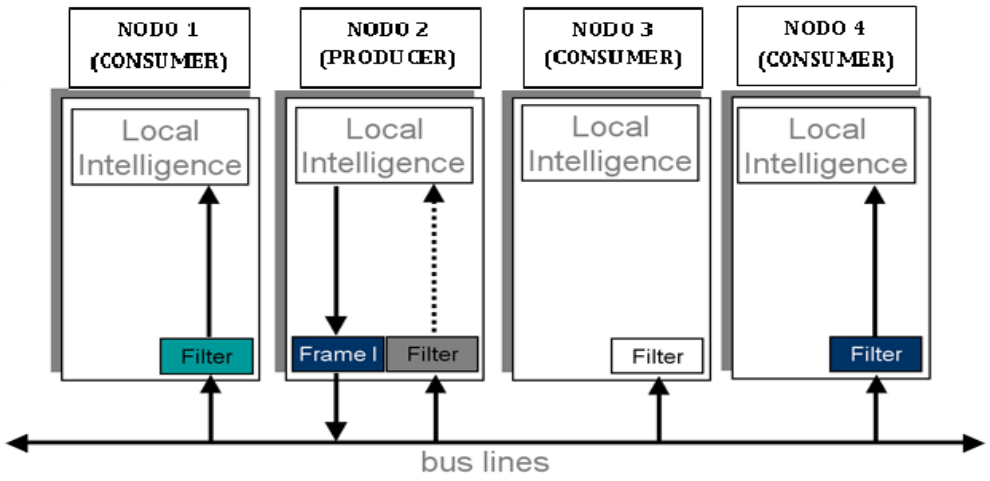


Figura 10: Estructura Broadcast.

2.6 Diagnosticar el CAN-Bus.

Los sistemas de seguridad que incorpora el CAN-Bus permiten que las probabilidades de fallo en el proceso de comunicación sean muy bajas, pero sigue siendo posible que cables, contactos y los propios dispositivos presenten alguna disfunción. Para el análisis de una avería, se debe tener presente que un Nodo averiado que este abonado al Bus, en muchos caso no impiden que el sistema siga trabajando con normalidad. Lógicamente no será posible llevar a cabo las funciones que implican el uso de información que proporciona la unidad averiada, pero sí todas las demás.

Es posible localizar fallos en el CAN-Bus utilizando sistemas de auto diagnosis, donde se podrá averiguar desde el estado de funcionamiento del sistema hasta los Nodos asociadas al mismo, pero necesariamente se ha de disponer del equipo de chequeo apropiado.

Otra alternativa es emplear programas informático como el CANking, MiniMon, CANAlyzer, etc y su correspondiente adaptador CANtoUSB. Estos programas permiten visualizar el tráfico de datos en el Bus, indicando el contenido de los mensajes.

Capítulo 3.

PROTOCOLO CANopen.

3. PROTOCOLO CANopen.

3.1 Modelado.

CAN está basado en los siguientes modelos de referencia, componente y comunicación.

3.1.1 Modelo de referencia.

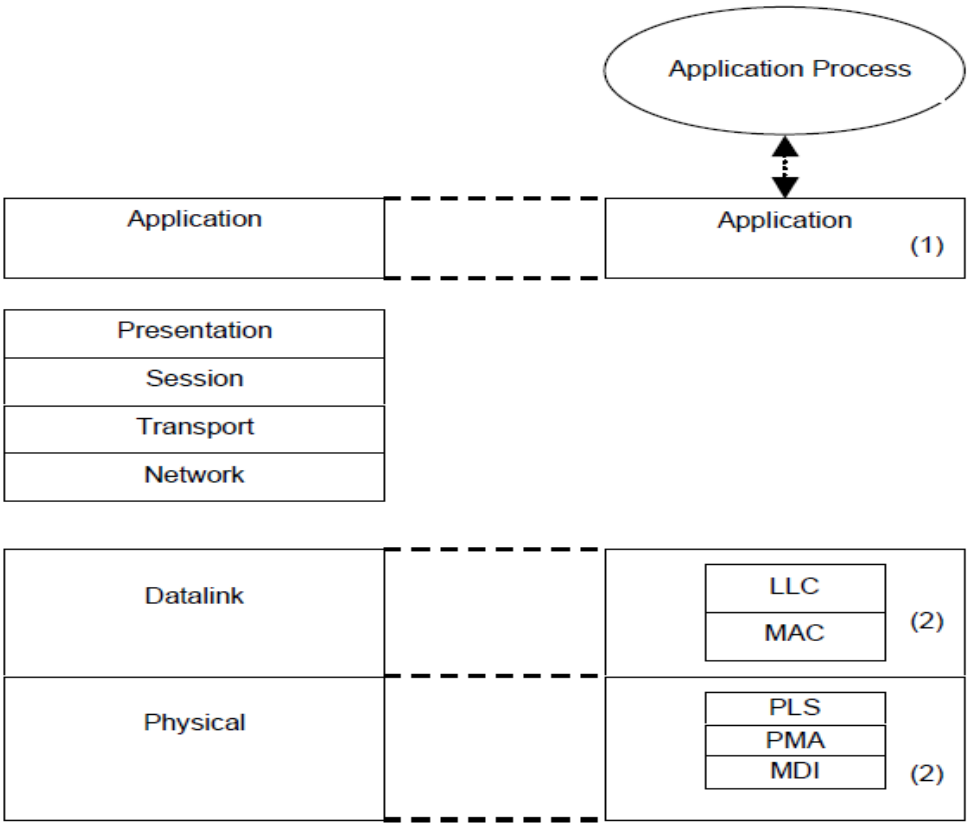


Figura 11: Modelo de referencia.

El concepto de comunicación es similar al descrito por el modelo ISO/OSI (parte izquierda de la figura). La capa física y la capa de lincado están cubiertas por el protocolo CAN-Bus de BOSCH, mientras que la capa de aplicación se describe a continuación.

Application layer:

La capa de aplicación está comprometida con un concepto de configuración,

transmisión de datos en ***Tiempo Real*** y el mecanismo de sincronización entre componentes. La funcionalidad que ofrece a la aplicación está dividida en diferentes Objetos de servicio que proporcionan una característica específica y todo lo relacionado con el servicio.

Las aplicaciones interactúan entre sí, invocando los servicios de un objeto de servicios en la capa de aplicaciones. Para realzar estos servicios, el Objeto intercambia datos por el Bus con uno o varios Objetos de servicio siguiendo un protocolo. Este protocolo se describe en el protocolo específico de cada Objeto de servicio.

Servicios primitivos:

Existen cuatro tipos de servicios primitivos que describen la forma de interactuar entre las aplicaciones y la capa de aplicación.

- **Request**: es emitido por una aplicación para solicitar un servicio a la capa de aplicación.
- **Indication**: es emitido desde la capa de aplicación para indicarle a una aplicación que se ha detectado un evento interno en la capa de aplicación o que un servicio es requerido.
- **Response**: es emitido por una aplicación para responder a una indicación previa de la capa de aplicación.
- **Confirmation**: es emitido por la capa de aplicación para informar a la aplicación del resultado de un request emitido previamente.

Tipos de servicios de la capa de aplicación:

Un tipo de servicio define los servicios primitivos que son intercambiados entre la capa aplicación y las aplicaciones para un servicio particular de objeto de servicio. Existen cuatro tipos de servicios:

- **Local service:** es un servicio que afecta sólo al objeto de servicio local. La aplicación emite un request a su objeto de servicio local que ejecuta el servicio requerido sin necesidad de establecer comunicación por el Bus.

- **Unconfirmed service:** este servicio afecta a uno o más servicios de objetos. La aplicación emite un request a su servicio local de objetos y este es transferido como una indicación a todos los servicios de objetos que se vean afectados por este requerimiento. El resultado de la indicación no es confirmado al emisor del request.
- **Confirmed service:** sólo puede afectar a un par de servicios de objetos que están interconectados entre sí vía el Bus, es decir comunicación punto a punto. La aplicación emite un request a su servicio local de objetos y este es transferido, al objeto de servicio asociado a este, como una indicación. La aplicación asociada reacciona ante esta indicación emitiendo una respuesta que es transferida al servicio de objeto que inicio el proceso. Esta respuesta es considerada por el Nodo receptor como la confirmación del servicio.
- **Provide initiated service:** afecta sólo al servicio local de objetos produciéndose cuando se detecta un evento que no ha sido solicitado mediante un request. Este evento es indicado a la aplicación.

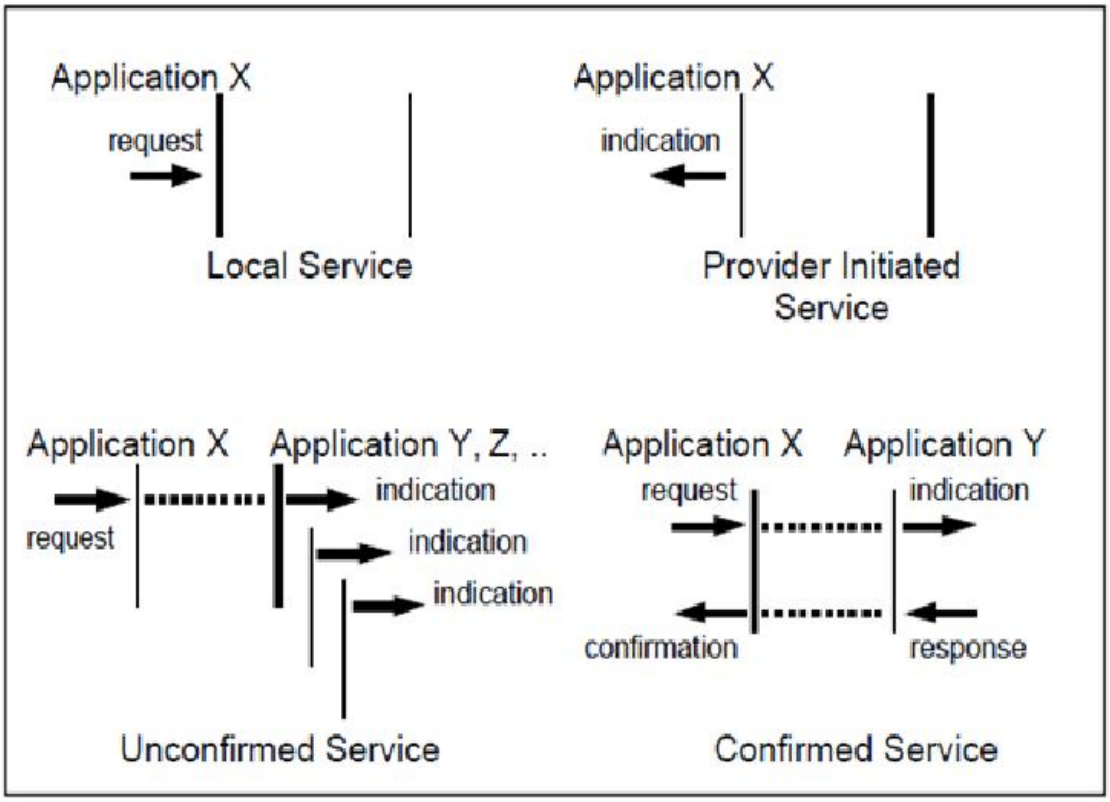


Figura 12: Tipos de Servicios.

Los servicios Unconfirmed y Confirmed se denominan servicios remotos debido a que aceden al bus para prestar el servicio.

3.1.2 Modelo del Nodo CANopen.

La figura siguiente muestra la estructura de un Nodo CANopen.

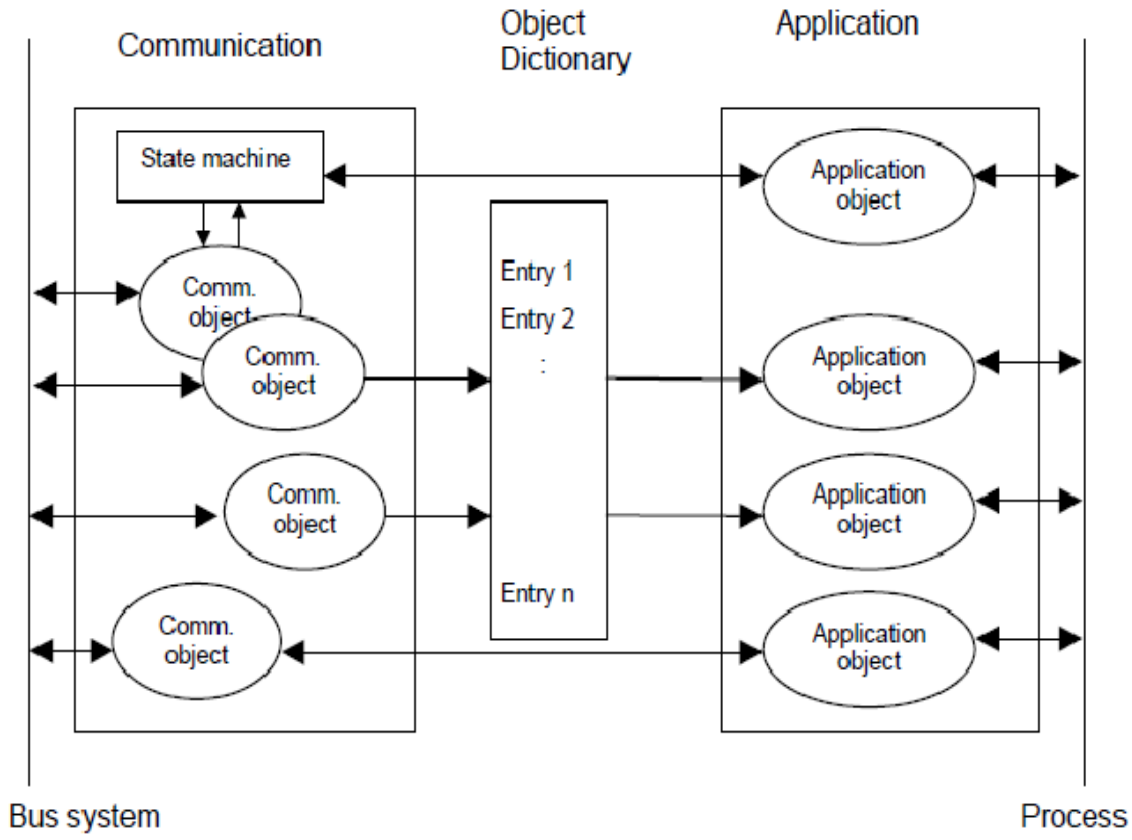


Figura 13: Modelo del Nodo CANopen.

- **Comunicación:** la función de esta unidad es suministrar los objetos de comunicación y la apropiada funcionalidad para transportar los campos de datos por el Bus.
- **Object Dictionary:** es una colección de todos los campos de datos que influyen en el desarrollo de las aplicaciones de objetos, los objetos de comunicación y la máquina de estados usada en el Nodo.
- **Aplicación:** la aplicación controla la funcionalidad del Nodo con respecto a la interacción con el entorno de procesos.

El diccionario de objetos cumple con la función de interfaz entre la comunicación y

la aplicación. La descripción completa de las aplicaciones de un dispositivo con respecto a las entradas en el diccionario del objeto se conoce como perfil del dispositivo.

El diccionario de objetos es una de las partes más importantes del perfil del componente por lo que se tratara en profundidad más adelante.

3.1.3 Modelo de comunicación.

El modelo de comunicación define los diferentes objetos de comunicación, servicios y modos de transmisión de los mensajes. Este modelo soporta tanto transmisiones síncronas como asíncronas.

Por medio de la transmisión síncrona, la adquisición de datos en la red puede ser completamente coordinada. Este tipo de comunicación esta predefinida para los objetos de comunicación (Sync message, time stamp message). Los mensajes síncronos son transmitidos con respecto a un mensaje de sincronización predefinido, mientras que los mensajes asíncronos pueden ser transmitidos en cualquier momento.

Debido al carácter eventual del mecanismo de comunicación es posible definir tiempos de inhibición para la comunicación. Estos tiempos consisten en establecer el tiempo mínimo que tiene que transcurrir entre dos servicios consecutivos del mismo objeto de datos, para garantizar que lo mensajes de baja prioridad acedan a la red durante el tiempo de inhibición. Los tiempos de inhibición pueden ser asignados por la aplicación.

Con respecto a la funcionalidad, se pueden distinguir tres tipos de comunicación:

- Comunicación Master/Slave.
- Comunicación Client/Server.
- Comunicación Producer/Consumer.

3.1.4 Comunicación Master/Slave.

Solamente puede haber un maestro en la red para una funcionalidad específica y en cualquier tiempo dado. Todos los demás nodos del Bus están considerados

como esclavos. En este tipo de comunicación se definen dos modelos (con confirmación y sin confirmación) en los que el maestro emite una petición y los esclavos responden a esta petición si el protocolo lo requiere.

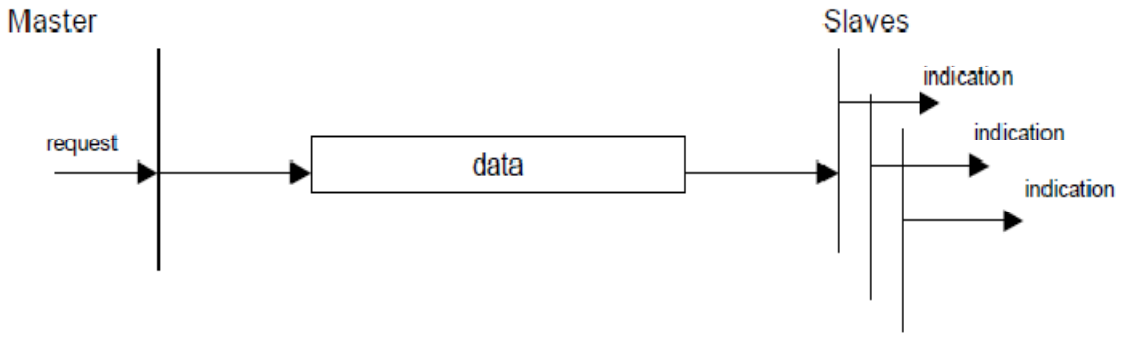


Figura 14: Comunicación Master/Slave Sin Confirmación.

En el modelo sin confirmación el maestro emite un mensaje que es recibido por todos los Nodos esclavos conectados al Bus y procesado solamente por los dispositivos cuya configuración de los filtros reconozca el identificador del mensaje entre los que deba tratar. El esclavo no emitirá una nueva trama como consecuencia de la recepción del mensaje emitido por el maestro.

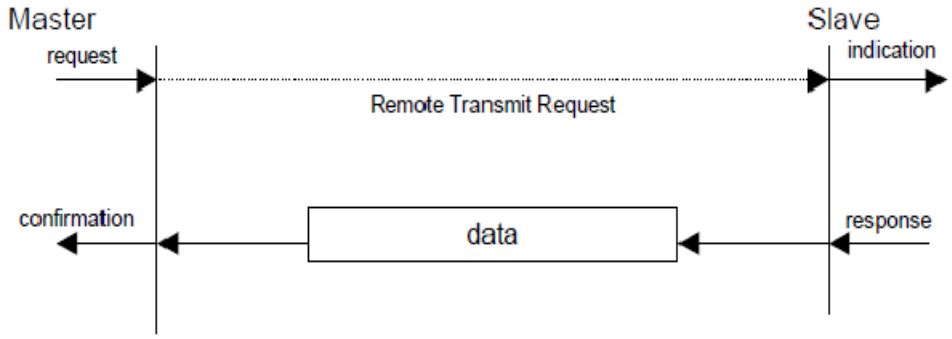


Figura 15: Comunicación Master/Slave Confirmado.

Sin embargo el modelo confirmado consiste en una petición de datos por parte del maestro al esclavo, por lo que si existe confirmación. La trama emitida por el maestro es del tipo RTR y por tanto no contiene datos. Este modo de comunicación no tiene carácter mandatorio, es decir, no es obligatoria su impleentacion.

3.1.5 Comunicación Client/Server.

Este tipo de comunicación se produce entre dos nodos donde uno de ellos cumple con la funcionalidad de cliente y el otro la de servidor. El cliente emite una petición de carga o descarga de datos pidiéndole al servidor que realice una cierta tarea. Después de finalizar la tarea el servidor emite una respuesta a la petición.

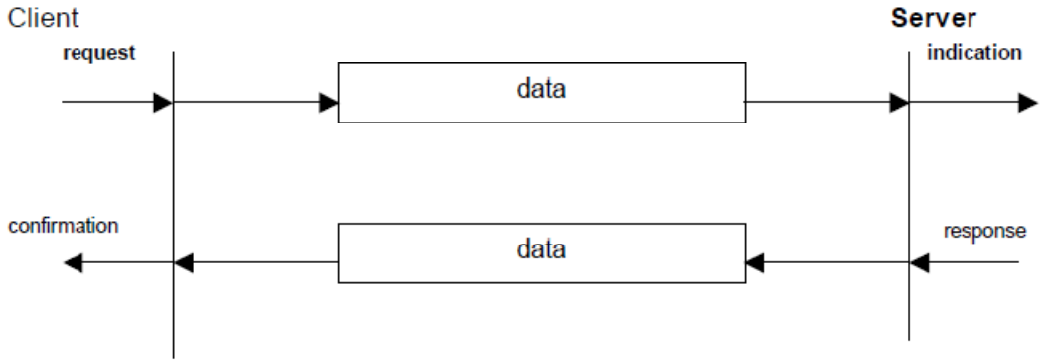


Figura 16: Comunicación Client/Server.

En el modelo cliente/servidor se establece una comunicación **punto a punto** ya que únicamente intervienen dos de los Nodos conectados al Bus.

3.1.6 Comunicación Producer/Consumer.

En este tipo de comunicación siempre se ve involucrado un Nodo como productor del mensaje y cero, uno o más consumidores de dicho mensaje. Se pueden distinguir dos variantes de este tipo de comunicación, el Push model que es un servicio sin confirmar y el Pull model que es un servicio confirmado.

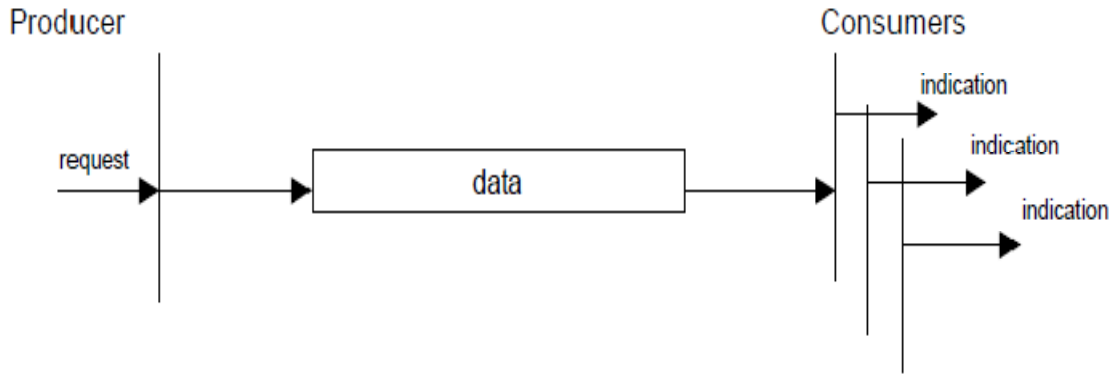


Figura 17: Push model.

El modelo Push es utilizado por los dispositivos productores de mensajes para transferir datos en **Tiempo Real** por el Bus. Este modelo es muy similar al modelo maestro/esclavo, con la diferencia de que el modelo maestro/esclavo se utiliza para controlar el estado de los Nodos en la red.

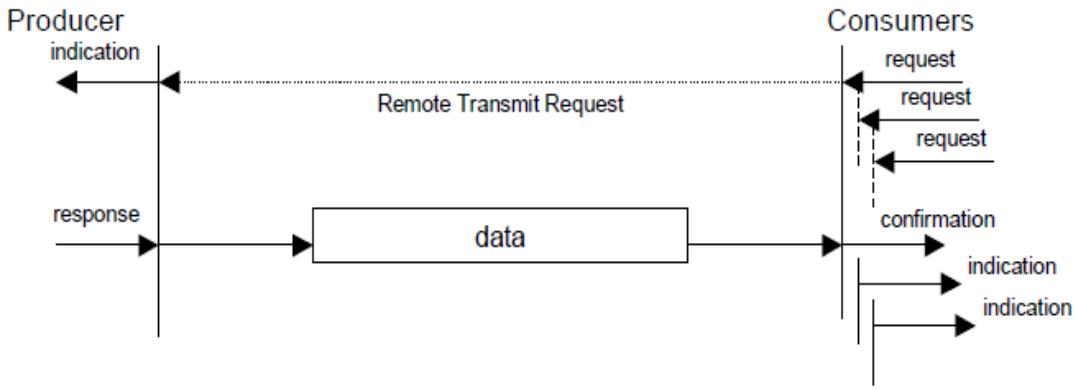


Figura 18: Pull model.

El modelo Pull, al igual que el modelo maestro/esclavo confirmado, se utiliza para solicitar datos a un Nodo productor. La diferencia radica al igual que en el caso anterior en los tipos de datos solicitados ya que el modelo Pull es usado para solicitar datos en Tiempo Real de control, mientras que el modelo maestro/esclavo confirmado se utiliza especialmente para confirmar el estado del esclavo.

3.2 Objetos de comunicación.

Los objetos de comunicación están descritos por los servicios y protocolos. Los tipos de servicios (confirmado, sin confirmar, etc.) están descritos de forma que contienen los parámetros de servicio primitivo que se definen para cada servicio en particular. Los distintos tipos de servicios que existen están definidos en el apartado 3.1.3 "*Modelo de comunicación*".

Tolos los servicios asumen que no se producirá ningún error, tanto en el lincado de datos como en la capa física del Bus. Y en el caso de que se produzcan, estos serán resueltos por la aplicación.

Todo el tráfico de datos por el Bus relacionado con un dispositivo CANopen está conceptualmente dividido en dos clases, el SDO y el PDO. El SDO incluye todo lo

relacionado con parámetros de configuración, mientras que el PDO abarca todo lo relacionado con las operaciones de **Tiempo Real** del dispositivo.

El protocolo CANopen requiere identificadores de mensajes tal que todos los PDOs reciban mayor prioridad que los SDOs, pero para garantizar que estos últimos acedan a la red se pueden establecer tiempos de inhibición para los SDOs como ya se ha comentado anteriormente. A continuación se puede observar una tabla comparativa entre ambos tipos de mensajes.

PDO (Process Data Object)	SDO (Service Data Object)
Usado para intercambio de datos en Tiempo Real.	Usado para mensajes de configuración.
Posibilidad de transmisión cíclica o acíclica.	Transmisión asíncrona normalmente.
Optimizado para alta velocidad.	Baja velocidad.
Mensaje de prioridad alta.	Mensaje de prioridad baja.
Sólo 8 bytes por mensaje.	Posibilidad de Multi-telegramas.
Campos de datos predefinidos.	Campos de datos referenciados mediante Multiplexor.
Estructura configurable mediante el canal de servicios.	Estructura fija.
Rápido.	Lento.

Tabla 1: Concepto de comunicación CANopen

Adicionalmente a los SDOs y PDOs existen otros tipos de objetos:

- **Network Management Object (NMT)** que se encarga de controlar y enviar información de la Máquina de Estados de los diferentes Nodos conectados al Bus.
- **Synchronisation Object (SYNC)** se encarga de sincronizar todas las aplicaciones de los Nodos conectados al Bus.
- **Time Stamp Object (TIME)** provee una referencia de tiempo común a todos los Nodos del Bus.
- **Emergency Object (EMCY)** envía información de errores internos en los Nodos si estos se producen y el objeto esta implementado.

3.2.1 Process Data Object (PDO).

Los Process Data Object (PDO) son usados para la transferencia de datos en **Tiempo Real**. Aunque estos no están indexados se corresponden con entradas en el Diccionario de Objetos y provee de una interfaz a los objetos de las aplicaciones. El tipo de dato y mapeado de un PDO para cada objeto de aplicación está determinada por defecto dentro de la estructura mapeada en el Diccionario de Objeto de cada dispositivo. Si el Nodo soporta el mapeado de PDO variable, el formato y contenido de los mensajes PDO pueden ser configurados entre el servidor y cliente en la fase de inicialización del Bus. Esta configuración se realiza mediante servicios SDO correspondientes a cada entrada en el Diccionario de Objetos que se desee modificar.

El número y tamaño de los PDOs de un dispositivo es específico de cada aplicación y está definido dentro del perfil específico de cada tipo de dispositivo.

Existen dos tipos de PDOs. El primero para transmitir un PDO (Transmit-PDOs ó TPDO) y el segundo para recibir un PDO (Receive-PDOs ó RPDOs). Los dispositivos que soportan la transmisión de PDOs son considerados productores de PDO y los que reciben PDOs consumidores de PDO. Los PDOs están descritos por los parámetros de comunicación PDO (en el index 20h) y por los parámetros de mapeado PDO (en el index 21h). La estructura de estos tipos de datos se explica en el apartado 3.10 "*Tipos de datos complejos predefinidos*". Los parámetros de comunicación describen la capacidad de comunicación de los PDOs (COB-ID utilizado por el PDO, tiempos de inhibición y temporización) y los parámetros de mapeado contienen información acerca del contenido de los PDOs. El índice de la correspondiente entrada en el Diccionario de Objetos se calcula aplicando las siguientes formulas:

- RPDO communication parameter index = 1400h + RPDO-number -1
- TPDO communication parameter index = 1800h + TPDO-number -1
- RPDO mapping parameter index = 1600h + RPDO-number -1
- TPDO mapping parameter index = 1A00h + TPDO-number -1

Los parámetros de comunicación y mapeado son obligatorios para todos los PDOs.

Las entradas al Diccionario de Objetos mencionada arriba están descritas en el artado 3.5 “Diccionario de Objetos”.

3.2.2 Modos de transmisión.

Se pueden distinguir dos modos de transmisión:

- Transmisión síncrona.
- Transmisión asíncrona.

Para sincronizar todos los dispositivos pertenecientes a la red CANopen se transmite periódicamente un objeto de sincronismo (SYNC Object). El mensaje de sincronismo es un objeto de comunicación predefinido que se tratara en el apartado 3.2.11 “*Synchronisation Object (SYNC)*”. En la siguiente figura se puede observar el principio de transmisión síncrono y asíncrono donde los mensajes síncronos son enviados dentro de una ventana de tiempo predefinida después de cada objeto de sincronismo.

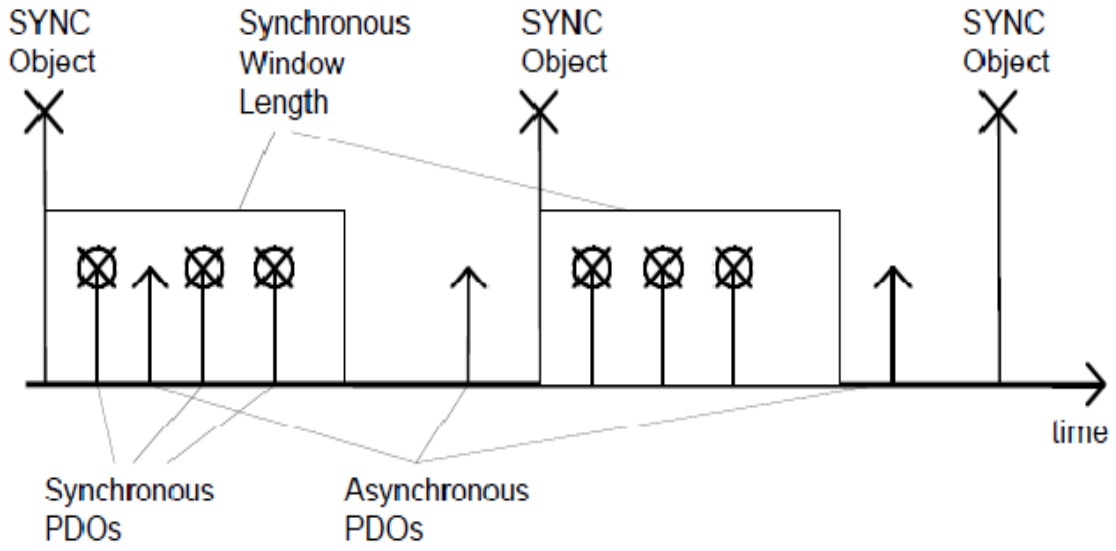


Figura 19: Transmisión de PDOs síncronos y asíncronos.

Tanto el modo de transmisión como el modo de accionamiento (evento que provoca la transmisión de un PDO) están especificados en los parámetros de tipo de transmisión de un PDO.

Para los TPDOs también se especifica la tasa de transmisión en forma de un factor

basado en el periodo de transmisión del objeto de sincronismo. Un facto '0' significa que el mensaje será transmitido después del objeto de sincronismo de forma acíclica, es decir, solamente si se produce un evento antes del objeto de sincronismo. Con el factor '1' el mensaje será transmitido después de cada objeto de sincronismo. Si el factor es n el mensaje se transmitirá después de 'n' objetos de sincronismo. En el caso de que el factor sea igual a '254' ó '255' el modo de transmisión será asíncrono.

Los TPDOs asíncronos pueden ser transmitidos en cualquier momento teniendo en cuenta sólo su prioridad con respecto a otros mensajes, pudiendo ser transmitidos dentro de la ventana de tiempo de los mensajes síncronos.

Los datos de un RPDO síncrono son pasados a la aplicación siempre que se reciban, independientemente del factor especificado por el tipo de transmisión.

Existen tres modos de accionamiento:

- **Event Driven:** La transmisión del mensaje es accionada por un evento específico del objeto. Es decir, para PDOs síncronos esto puede ser la expiración del periodo de transmisión especificado por el factor respecto del objeto de sincronismo.
- **Timer Driven:** La transmisión es accionada por un evento especificado por las características del dispositivo o si ha transcurrido un tiempo sin que se produzca el evento especificado.
- **Remotely Requested:** La transmisión de un PDO asíncrono puede ser inicializado por cualquier Nodo consumidor de PDOs por medio de una trama remota siempre y cuando este modo sea soportado.

3.2.3 Servicios PDO.

La comunicación PDO puede describirse mediante el modelo Producer/Consumer. Los datos de proceso pueden ser enviados desde un Nodo productor a uno o más Nodos consumidores (Broadcasting). Los PDOs son transmitidos en un modo no confirmado. El Nodo productor envía un Transmit-PDO con un identificador específico y que se corresponde con el identificador del Receive-PDO de uno o más

consumidores.

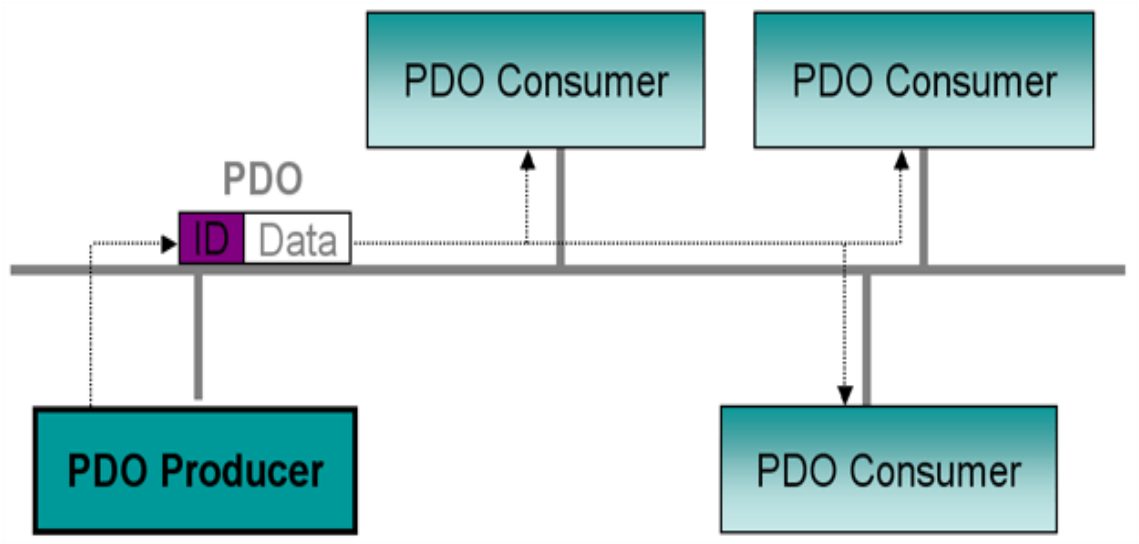


Figura 20: Comunicación PDO

Se pueden diferenciar dos tipos de servicios, el Write-PDO y el Read-PDO.

3.2.4 Protocolo del servicio Write-PDO.

Los Write-PDOs están mapeados en una sola trama de datos CAN que contiene el identificador y hasta 8 bytes de datos, siguen el modelo Push de comunicación y es producido por un Nodo pudiendo tener uno o varios consumidores.

A través de este servicio el productor del PDO manda los datos mapeados en los objetos de aplicaciones a los consumidores de PDOs.

Parameter	*Request / Indication*
Argument PDO Number Data	**Mandatory** mandatory mandatory

Tabla 2: Write-PDO

La petición de escribir un PDO es un servicio sin conformar en el que los productores de PDOs envían datos de proceso dentro de un PDO al Bus y los consumidores indican la recepción de un PDO valido como se puede observar en la

siguiente figura.

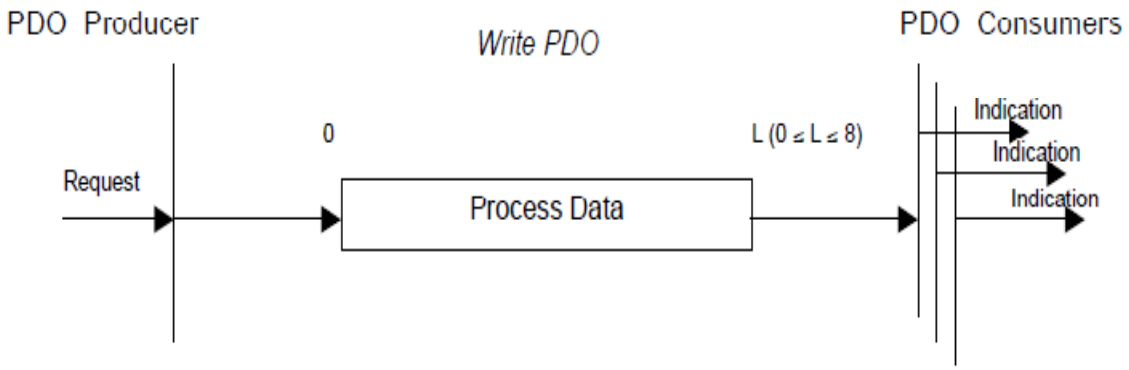


Figura 21: Write-PDO.

Process Data: se compone de hasta L bytes de datos de aplicación de acuerdo con el mapeado del PDO. Si L es mayor que el número de bytes 'n' mapeado en el PDO sólo se utilizaran los 'n' primeros bytes. En cambio si 'n' es menor, los datos recibidos en el PDO no se procesaran y se generara un mensaje de emergencia con el código de error 8210h si este es soportado.

3.2.5 Protocolo del servicio Read-PDO.

Los Read-PDO se mapean en una trama remota CAN, siguen el modelo de comunicación Pull y a diferencia de los Write-PDO existen un ó varios productores y un consumidor.

A través de este servicio cualquier Nodo consumidor de PDOs puede hacer una petición de datos mapeados en un objeto de aplicaciones al productor del PDO. Este servicio es confirmado por medio del envió del PDO con los datos mapeados en el objeto.

Parameter	*Request / Indication*	*Response / Confirm*
Argument PDO Number	**Mandatory** mandatory	
Remote Result Data		**Mandatory** mandatory

Tabla 3: Read-PDO.

Este es un servicio confirmado en el que uno ó más consumidores de PDOs transmiten una trama remota de petición de datos al Bus. El productor del PDO envía los datos solicitados como consecuencia de la recepción de esta trama remota como puede observarse en la siguiente figura. Este servicio es **opcional** y depende de las capacidades del Hardware para soportar tramas remotas.

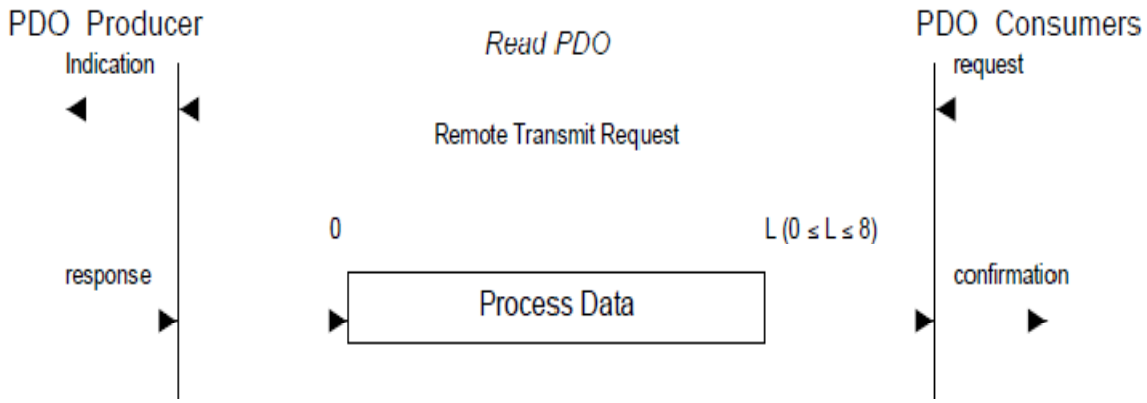


Figura 22: Read-PDO.

Process Data: se compone de hasta L bytes de datos de aplicación de acuerdo con el mapeado del PDO. Si L es mayor que el número de bytes 'n' mapeado en el PDO sólo se utilizaran los 'n' primeros bytes, en cambio si 'n' es menor los datos recibidos en el PDO no se procesaran y se generara un mensaje de emergencia con el código de error 8210h si este es soportado.

3.2.6 Service Data Object (SDO).

El Service Data Object (SDO) permite comunicar los objetos de datos entre el Maestro y los demás Nodos por medio del acceso al diccionario de objetos del dispositivo CANopen siguiendo el modelo de comunicación Client/Server, donde el Nodo propietario del Diccionario ejerce la función de servidor. Como los tipos de datos y tamaños de los mismos pueden variar desde un simple Booleano a un archivo de datos más complejo, el SDO se utiliza para la transmisión de grupos de datos múltiples desde el cliente al servidor y viceversa estableciendo un canal de comunicación punto a punto para el que se necesitan dos identificadores distintos ya que el servicio es confirmado. Aunque los dispositivos CANopen podrían soportar la implementación de más de un SDO, por defecto sólo se implementa un SDO en cada servidor.

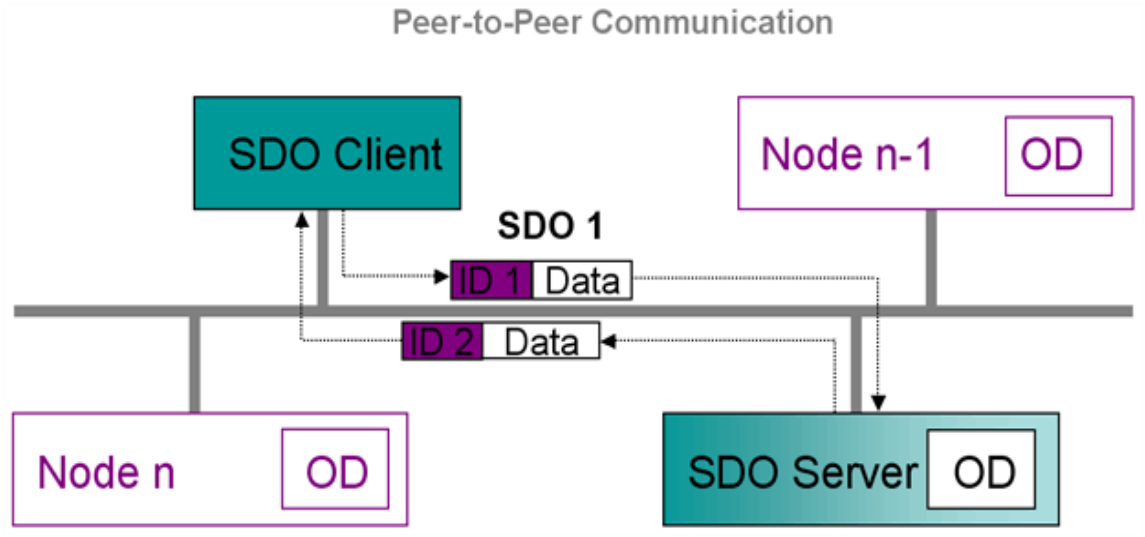


Figura 23: Comunicación punto a punto

Por medio del multiplexor (índice y subíndice del Diccionario de Objetos) el cliente puede controlar que tipo de datos están siendo transmitidos ya que el contenido de dichos datos está definido en el Diccionario de Objetos.

Los datos se transfieren por medio de una secuencia de segmentos que están precedidos de una fase de inicialización en la que el cliente y servidor se preparan para la transmisión de los segmentos. Durante la fase de inicialización es posible enviar hasta 4 bytes de datos sin necesidad de segmentación si la longitud de los datos no supera los 4 bytes, conociéndose este mecanismo como transferencia *acelerada*.

Cuando el tamaño de los datos excede los 4 bytes es necesario realizar una transferencia *segmentada* que permite enviar datos de cualquier longitud. Las tramas enviadas después de la fase de inicialización contienen un byte de control y hasta 7 bytes de datos. El final del servicio es indicado mediante un bit en la última trama.

Opcionalmente un SDO puede ser transmitido como una secuencia de *bloques* donde cada bloque está compuesto de una secuencia de hasta 127 tramas que contiene el número de secuencia y los datos a enviar. Este modo de transmisión también se inicializa con una primera trama en la que cliente y servidor se preparan para la transmisión del bloque y negocian el número de segmentos de

cada bloque. Después de transmitir un bloque es posible comprobar la veracidad de los datos transmitidos por medio del Checksum derivado de los datos transmitidos. La transmisión en bloques es más rápida que la segmentación cuando los datos a enviar tienen una longitud elevada ya que sólo se confirma la recepción del bloque a diferencia de la transferencia segmentada en la que se confirma cada trama.

Cuando se trata de un upload de un bloque SDO es posible que la longitud de los datos no justifique el uso de bloques porque implica el uso de un protocolo más elevado. En este caso se puede implementar un soporte capaz de retroceder al modo segmentado o acelerado en la fase de inicialización. Como la suposición de la longitud mínima de datos para la cual el tiempo de transmisión de bloques supera a la transmisión de segmentos depende de varios parámetros, el cliente indica al servidor en la fase de inicialización el valor mínimo de la longitud de datos en bytes para la transferencia en bloques.

Para todos los modos de transmisión de SDOs el cliente es el que inicializa la comunicación y en el caso de que se produzca un error tanto el cliente como el servidor pueden tomar la decisión de abortar la transmisión del SDO.

El SDO communication parameter record (en el índice 22h) describen los parámetros de SDO. La estructura de estos tipos de datos se explica en el apartado 3.10 "*Tipos de datos complejos predefinidos*". Los parámetros de comunicación describen la capacidad de comunicación entre los Server-SDOs y Client-SDOs (CSDO). El índice del Diccionario de Objetos se calcula aplicando las siguientes formulas:

- SSDO communication parameter index = 1200h + SSDO-number -1
- CSDO communication parameter index = 1280h + CSDO-number -1

Los parámetros de comunicación son obligatorios para todos los SDOs, aunque pueden ser omitidos si sólo existe un SSDO. Las entradas al Diccionario de Objetos mencionada arriba están descritas en el artado 3.5 "Diccionario de Objetos".

A continuación se incluye un resumen de las características de los SDOs:

- Servicio confirmado (request/response) para acceso de lectura y escritura a datos de cualquier tamaño.
- Transferencia acelerada para datos menores o iguales a 4 bytes.
- Transferencia segmentada ó en bloques para datos de más de 4 bytes.
- Transferencia de datos identificados por el Multiplexor (índice y subíndice del Diccionario de Objetos).
- Posibilidad de abortar transferencia tanto por el cliente como por el servidor.

3.2.7 Servicios SDO.

El SDO sigue el modelo de comunicaciones Client/Server descrito en el apartado 3.1.5. Los atributos de este se representan en la siguiente tabla:

ATRIBUTOS DEL SDO	
Nº de SDO	Valor en el rango [1...128] para cada tipo de usuario local.
Tipo de usuario	Seleccionable {Server, Client}.
Multiplexor	Índice: UNSIGNED16. Subíndice: UNSIGNED8.
Tipo de transferncia	Depende del tipo de transferencia: • Expedited (Acelerado): para 4 o menos bytes. • Segmented ó block: para más de 4 bytes.
Tipo de datos	Condicionado por el multiplexor.

Tabla 4: Atributos del SDO.

Existen tres tipos de servicios SDO que dependen de las necesidades de la aplicación:

- **SDO Upload**, puede ser dividido en:
  - Iniciar SDO Upload.
  - Segmentado SDO Upload.
- **SDO Download**, puede ser dividido en:
  - Iniciar SDO Download.
  - Segmentado SDO Download.
- **Abortar transferencia SDO**.

El modo de transmisión segmentado es obligatorio solamente cuando el dispositivo soporta Objetos de datos de más de 4 bytes, mientras que la implementación del modo Expedited (acelerado) es de carácter obligatorio y la transferencia en bloques opcional. Cuando se utiliza el servicio de SDO segmentado, tanto para un Upload como un Download, la comunicación software es responsable de segmentar los datos. Del mismo modo el software también se encarga de secuenciar los bloques si este tipo de transferencia esta implementado.

3.2.8 Protocolo del servicio Download SDO.

A través de este servicio **el cliente descarga datos en el servidor** estableciendo una interfaz de comunicación por medio del Diccionario de Objetos. El cliente indica al servidor que datos debe descargar por medio del multiplexor y opcionalmente, si se trata de una transferencia segmentada, la longitud de los datos. Este es un servicio confirmado por medio de una trama remota que indica el éxito o fallo de la operación. En caso de fallo, la razón puede ser indicada si está implementada esta opción.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Data	mandatory	
Size	optional	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		selection
Failure		selection
Reason		optional

Tabla 5: Download SDO.

Este protocolo está dividido en dos fases. La primera es la de inicialización y que se conoce también por Expedited cuando los datos son de 4 o menos bytes. La segunda fase es la fase de segmentación que sólo tiene lugar en el caso de que los datos superen los 4 bytes. En la siguiente figura se puede observar el esquema que sigue el protocolo.

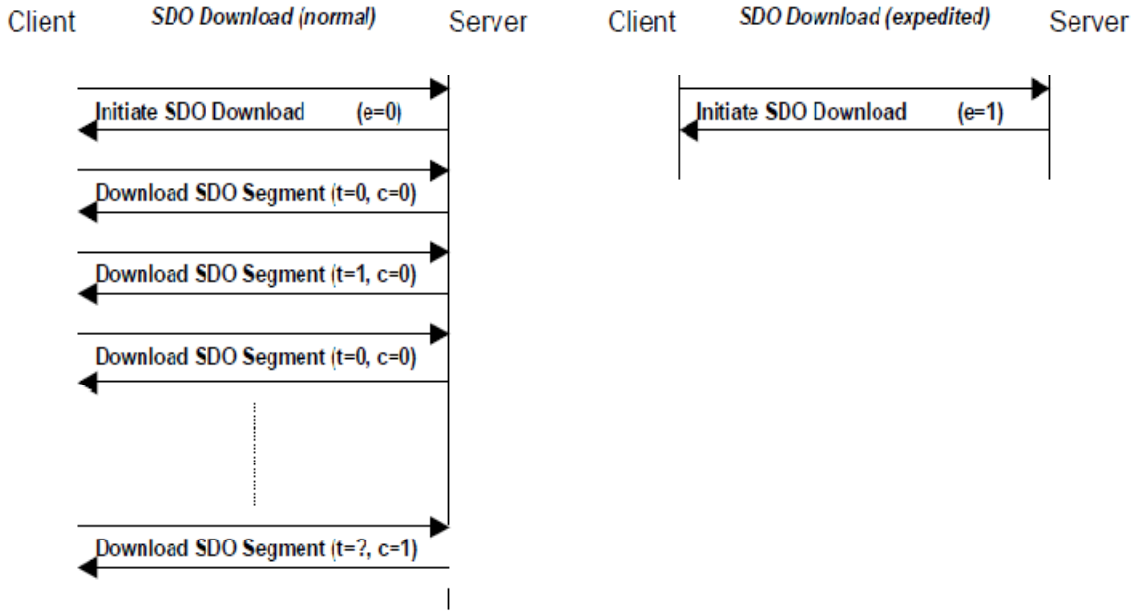


Figura 24: Protocolo Download SDO.

Este protocolo sigue la siguiente secuencia:

Envío inicial de un SDO Download Request/indication con el bit '**e'** activado si es un Expedited transfer o desactivado si es un Segmented transfer y seguido de un SDO Download response/confirm indicando que la fase de inicialización se ha realizado con éxito. El mensaje de confirmación sigue el protocolo de inicialización si **'e = 1'** o el de segmentación si **'e = 0'**.

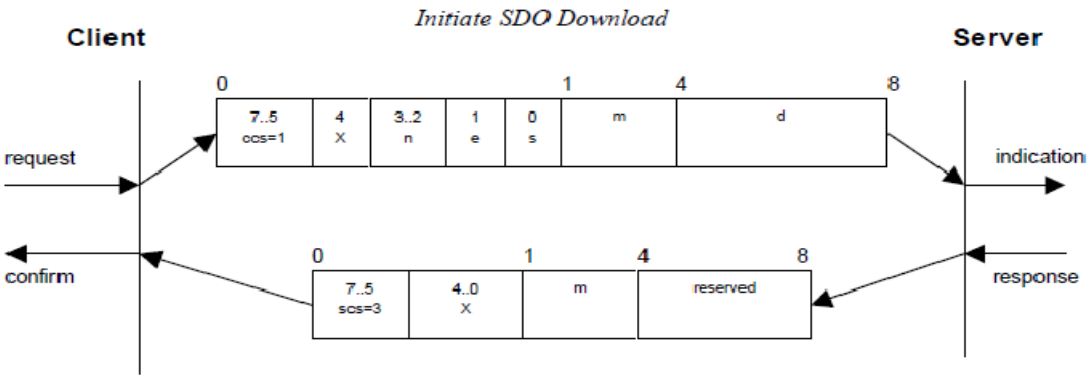


Figura 25: Protocolo Inicialización SDO Download.

ccs: comando especifico de cliente

1 → indica inicialización de Download.

scs: comando especifico de servidor

3 → indica respuesta a la inicialización de Download.

n: indica el nº de bytes que no contienen datos utilizando la siguiente expresión: bytes [8-n, 7] no contienen datos. Es válido sólo si e = 1 y s = 1.

e: tipo de transferencia

0 → transferencia normal.

1 →transferencia acelerada.

s: indicador de longitud de datos enviados

0 → la longitud de datos no está indicado.

1 → la longitud de datos está indicado.

m: multiplexor. Representa el índice y subíndice del dato transferido.

d: datos transferidos

- Si e = 0, s = 0: d está reservado para futuros usos.
- Si e = 0, s = 1: d contiene 4-n bytes de datos siendo el byte 4 el menos significativo y el 8 el más significativo.
- Si e = 1, s = 1: d contiene los datos transferidos con la longitud en bytes indicada por 4-n. la codificación depende del tipo de datos indicado por el multiplexor.
- Si e = 1, s = 0: d contiene un nº de bytes sin especificar.

x: bits no usados.

reserved: reservado para futuros usos.

Envío del resto de los datos mediante consecutivos Download SDO Segment request/indication seguidos de un Download SDO Segment response/confirm con el bit **'c = 0'** si quedan más tramas por transmitir ó en el caso de que se hayan recibido todos los datos **'c = 1'**.

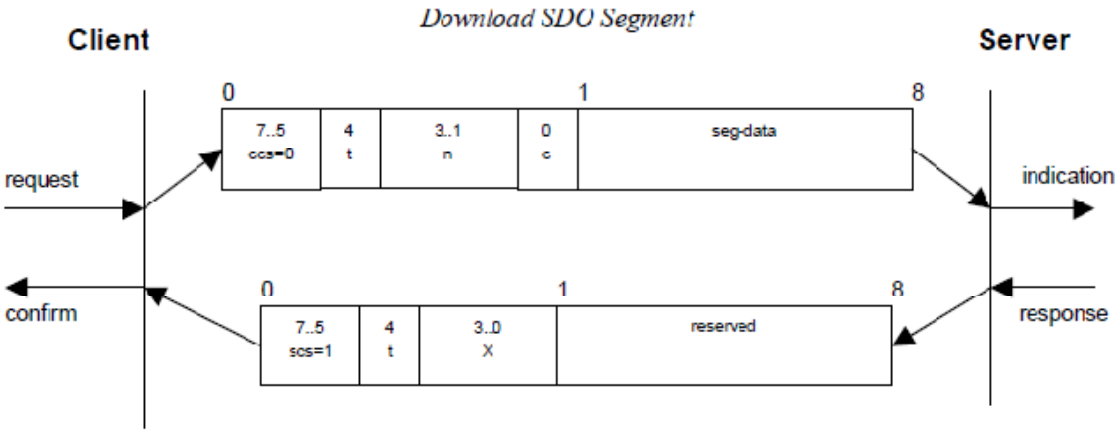


Figura 26: Protocolo segmentación Download SDO.

ccs: comando especifico de cliente

0 → indica envió de segmento SDO Download.

scs: comando especifico de servidor

1 → indica respuesta al segmento SDO Download.

n: indica el nº de bytes en seg-data que no contienen datos utilizando la siguiente expresión: bytes [8-n, 7] no contienen datos. n = 0 si el tamaño de datos no está indicado.

c: indica si quedan más segmentos por descargarse

0 → quedan más segmentos por descargarse.

1 →transferencia completada.

t (toggle bit): este bit debe alternar entre 1 y 0 entre dos segmentos consecutivos y tendrá el mismo valor tanto en la trama de petición como en la de confirmación. En el primer segmento **t** será igual a 0.

Seg-data: datos transferidos con una longitud máxima de 7 bytes donde la codificación de los datos depende del multiplexor.

x: bits no usados.

reserved: reservado para futuros usos.

3.2.9 Protocolo del servicio Upload SDO.

A través de este servicio **el cliente hace una petición de datos al servidor** siguiendo un proceso similar al Download SDO. El cliente indica al servidor que datos debe enviar por medio del multiplexor. Este es un servicio confirmado por medio de una trama remota que indica el éxito o fallo de la operación. En caso de fallo, la razón puede ser indicada si esta implementada esta opción.

Parameter	*Request / Indication*	*Response / Confirm*
Argument SDO number Multiplexor	**Mandatory** mandatory mandatory	
Remote Result Success Data Size Failure Reason		**Mandatory** selection mandatory optional selection optional

Tabla 6: Upload SDO.

Al igual que el Download SDO está dividido en dos fases. La primera es la de inicialización y la segunda fase es la fase de segmentación que sólo tiene lugar en el caso de que los datos superen los 4 bytes. En la siguiente figura se puede observar el esquema que sigue el protocolo.

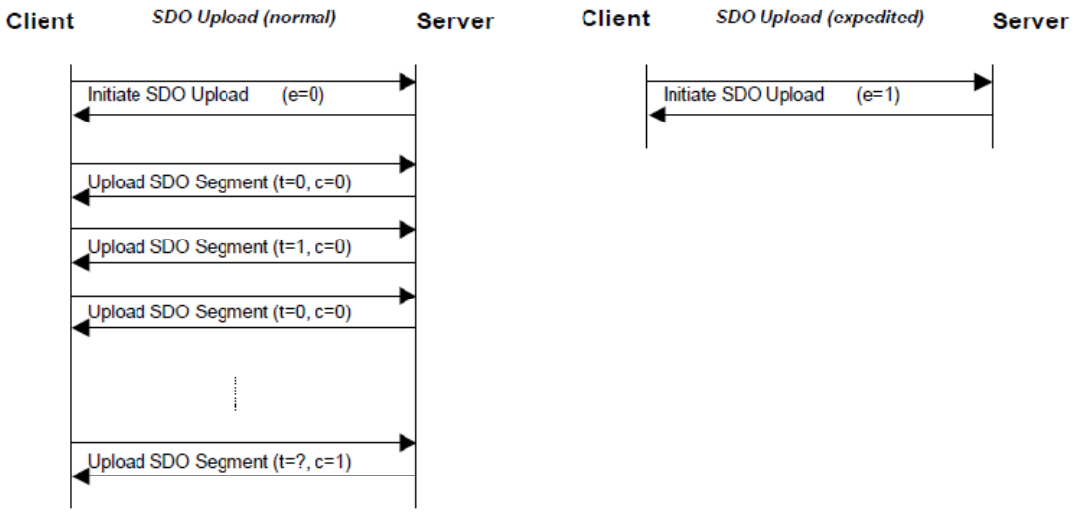


Figura 27: Protocolo Upload SDO

Este protocolo sigue la siguiente secuencia:

Envío inicial de un SDO Upload Request/indication solicitando los datos deseados, seguido de SDO Upload response/confirm con el bit **'e'** activado si se trata de un Expedited transfer o desactivado si es un Segmented transfer.

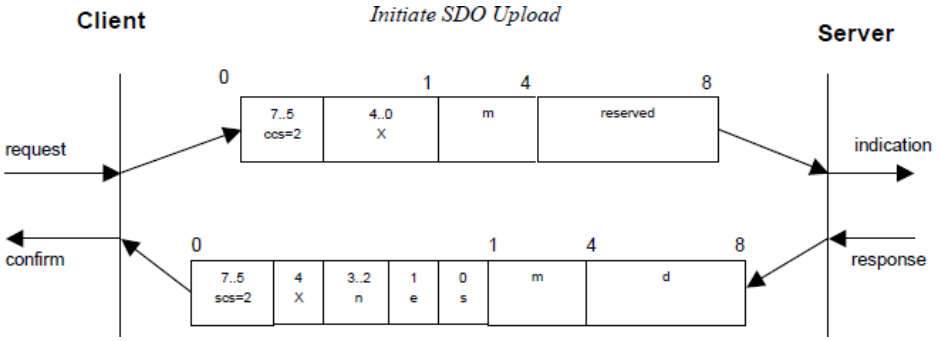


Figura 28: Protocolo inicialización Upload SDO.

ccs: comando especifico de cliente

2 → indica inicialización de Upload.

scs: comando especifico de servidor

2 → indica respuesta a la inicialización de Upload.

n: indica el nº de bytes que no contienen datos utilizando la siguiente expresión: bytes [8-n, 7] no contienen datos. Es válido sólo si e = 1 y s = 1.

e: tipo de transferencia

0 → transferencia normal.

1 →transferencia acelerada.

s: indicador de longitud de datos enviados

0 → la longitud de datos no está indicado.

1 → la longitud de datos está indicado.

m: multiplexor. Representa el índice y subíndice del dato solicitado.

d: datos transferidos

- Si e = 0, s = 0: d está reservado para futuros usos.
- Si e = 0, s = 1: d contiene 4-n bytes de datos siendo el byte 4 el menos significativo y el 8 el más significativo.
- Si e = 1, s = 1: d contiene los datos solicitados con la longitud en bytes indicada por 4-n. la codificación depende del tipo de datos indicado por el multiplexor.
- Si e = 1, s = 0: d contiene un nº de bytes sin especificar.

x: bits no usados.

reserved: reservado para futuros usos.

Petición del resto de los datos solicitados mediante consecutivos Upload SDO Segment request/indication seguidos de un Upload SDO Segment response/confirm con el bit **'c = 0'** si quedan más tramas por transmitir ó **'c = 1'** si se han trasmitido todos los datos.

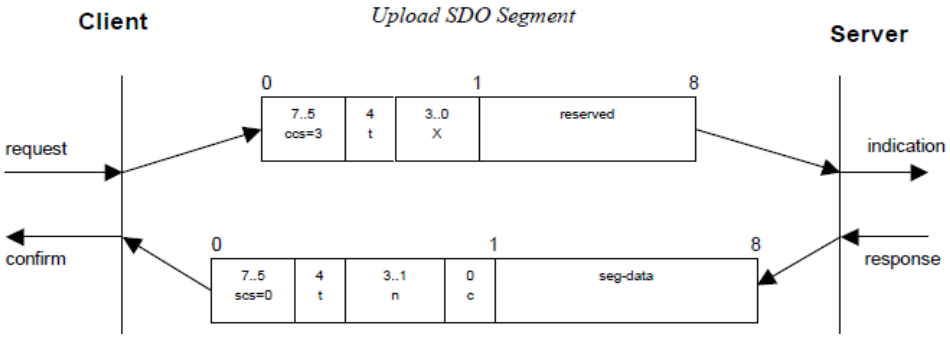


Figura 29: Protocolo segmentación Upload SDO.

ccs: comando especifico de cliente

3 → indica petición de segmento SDO Upload.

scs: comando especifico de servidor

0 → indica respuesta al segmento SDO Upload.

n: indica el nº de bytes en seg-data que no contienen datos utilizando la siguiente expresión: bytes [8-n, 7] no contienen datos. n = 0 si el tamaño de datos no está indicado.

c: indica si quedan más segmentos por transferir

0 → quedan más segmentos por transferir.

1 →transferencia completada.

t (toggle bit): este bit debe alternar entre 1 y 0 entre dos segmentos consecutivos y tendrá el mismo valor tanto en la trama de petición como en la de confirmación. En el primer segmento **'t'** será igual a 0.

Seg-data: datos transferidos con una longitud máxima de 7 bytes donde la codificación de los datos depende del multiplexor.

x: bits no usados.

reserved: reservado para futuros usos.

3.2.10 Protocolo del servicio Abortar Transferencia SDO.

Este protocolo se implementa para finalizar la transferencia de un SDO en el caso de que se produzca un error ó se superen los tiempos establecidos para realizar la transmisión de los datos. Opcionalmente se puede enviar un código de error codificado en 4 bytes (UNSIGNED32). Es un servicio sin confirmar y tanto el cliente como el servidor pueden ejecutar el servicio.

Abort code	DESCRIPTION
0503 0000h	Toggle bit not alternated.
0504 0000h	SDO protocol timed out.
0504 0001h	Client/server command specifier not valid or unknown.
0504 0002h	Invalid block size (block mode only).
0504 0003h	Invalid sequence number (block mode only).
0504 0004h	CRC error (block mode only).
0504 0005h	Out of memory.
0601 0000h	Unsupported access to an object.
0601 0001h	Attempt to read a write only object.
0601 0002h	Attempt to write a read only object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility in the device.
0606 0000h	Access failed due to an hardware error.
0607 0010h	Data type does not match, length of service parameter does not match
0607 0012h	Data type does not match, length of service parameter too high
0607 0013h	Data type does not match, length of service parameter too low
0609 0011h	Sub-index does not exist.
0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0609 0036h	Maximum value is less than minimum value.
0800 0000h	General error
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

Tabla 7: Tabla de códigos de error.

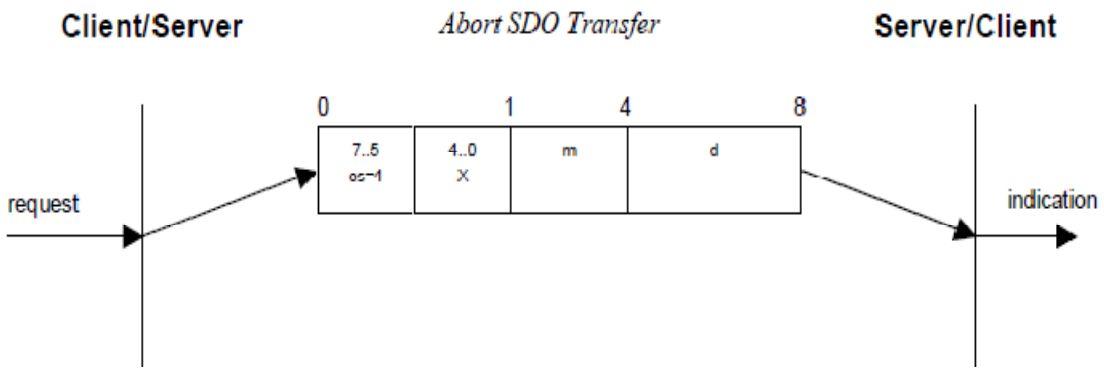


Figura 30: Protocolo Abortar SDO.

cs: comando especifico

4 → indica abortar SDO.

x: bits no usados.

m: multiplexor del SDO.

d: contiene 4 bytes de datos que indican el código de error producido.

En la siguiente tabla se observan los posibles códigos de error transmitidos por el servicio de aborto de un SDO.

3.2.11 Synchronisation Object (SYNC).

El Objeto de sincronización es emitido periódicamente por un Nodo productor de sincronismo, con el objetivo de generar un reloj básico en el Bus CANopen. El periodo de transmisión entre mensajes de sincronismo esta especificado por los parámetros estándares del **periodo de ciclo de transmisión en el objeto 1006h** y puede ser modificado en el proceso de inicialización del Bus. Cuando un Nodo consumidor de mensajes de sincronismo recibe el mensaje comienza a realizar las tareas síncronas. En general la transmisión de PDOs ligados al mensaje de sincronismo garantiza que los dispositivos sensores puedan muestrear variables de proceso de forma coordinada con las necesidades de los dispositivos actuadores sobre dichos procesos.

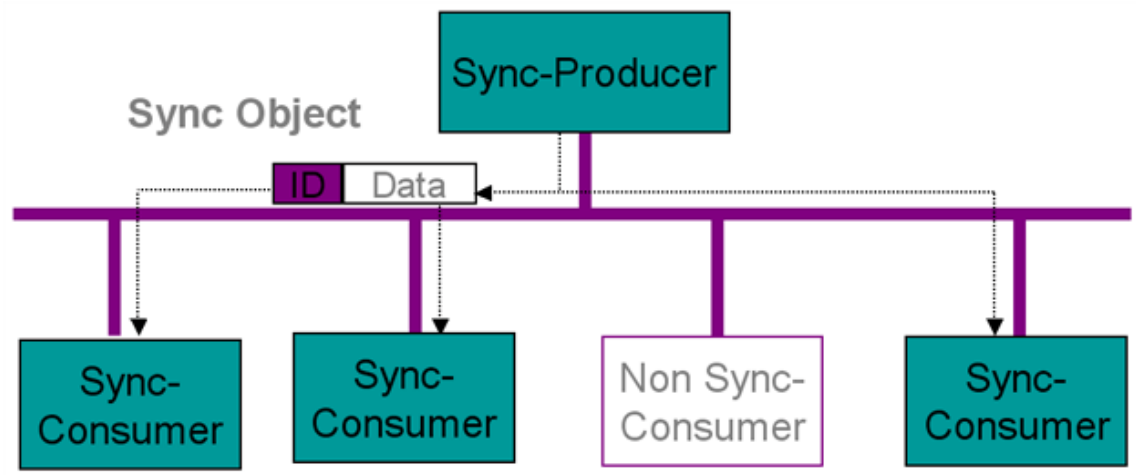


Figura 31: Objeto de sincronización.

Para garantizar el acceso inmediato del mensaje de sincronismo al Bus, este debe tener asignado un identificador con un alto grado de prioridad. El identificador del mensaje de sincronismo está disponible en el objeto 1005h.

3.2.12 Protocolo del SYNC Object.

El SYNC Object sigue el modo Push del modelo de comunicación producer/consumer siendo este un servicio sin confirmar. El mensaje está compuesto por el identificador localizado en el objeto 1005h, como ya se ha comentado anteriormente, y cuyo valor por defecto es el 128 (80h) y no contiene ningún dato (L = 0).

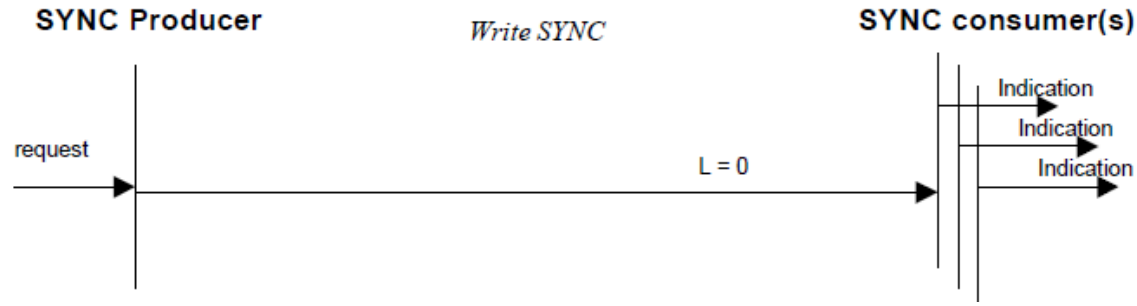


Figura 32: Protocolo de sincronismo.

Los PDOs síncronos son transmitidos dentro de un periodo de tiempo dado por la longitud de la ventana de sincronismo que está definido en el objeto 1007h. El tiempo entre dos mensajes de sincronismo consecutivos está definido en el objeto

1006h (periodo de ciclo de comunicación). Estos dos objetos pueden ser modificados en el proceso de configuración del Bus.

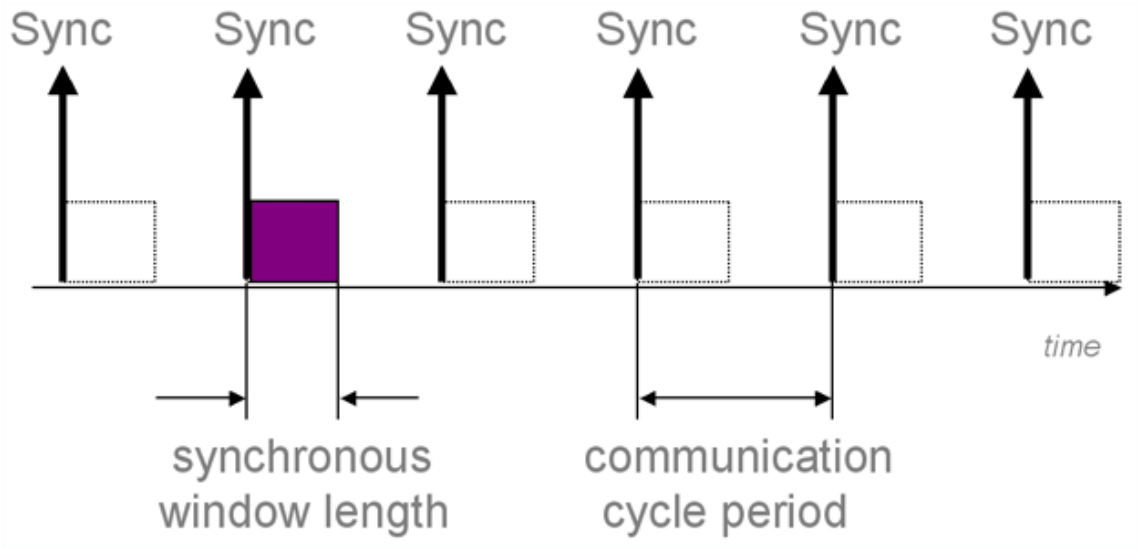


Figura 33: Definiciones de tiempos de sincronismo.

3.2.13 Time Stamp Object (TIME).

Este es un objeto que representa el tiempo absoluto en ms desde medianoche del 1 de enero de 1984. Es usado cuando el sistema requiere una alta precisión de sincronismo para ajustar la inevitable deriva de los relojes locales.

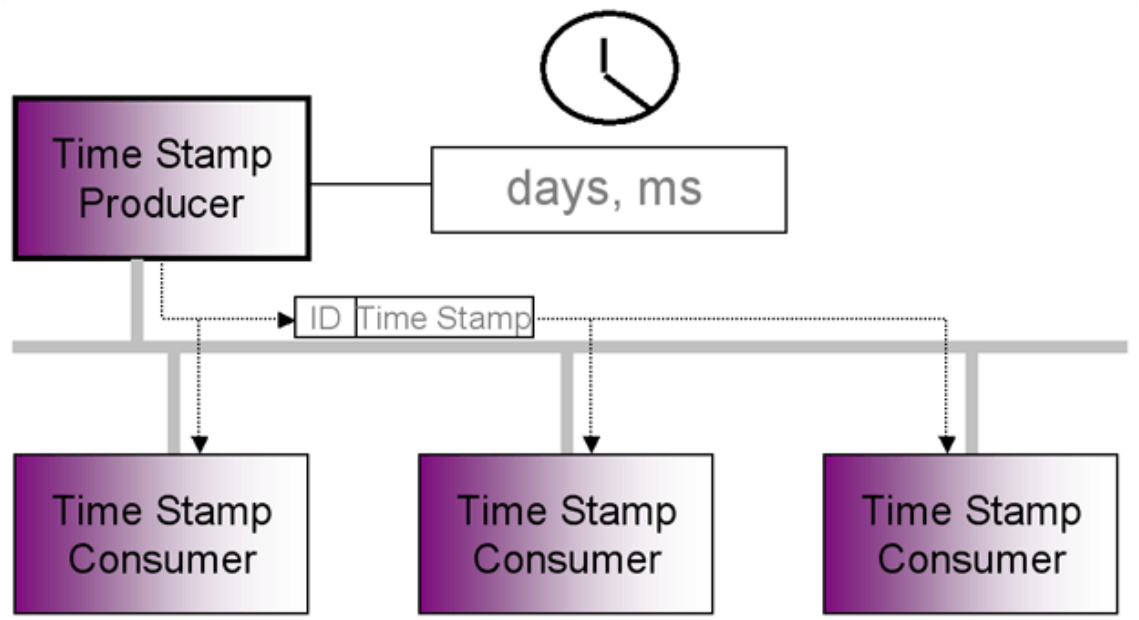


Figura 34: Time Stamp.

3.2.14 Protocolo del Time Stamp Object.

Sigue el modelo de comunicaciones producer/consumer y contiene un valor de tipo TIME_OF_DAY. El identificador de dicho objeto, cuyo valor por defecto es el 256 (100h), está almacenado en el índice 1012h del Diccionario de Objetos.

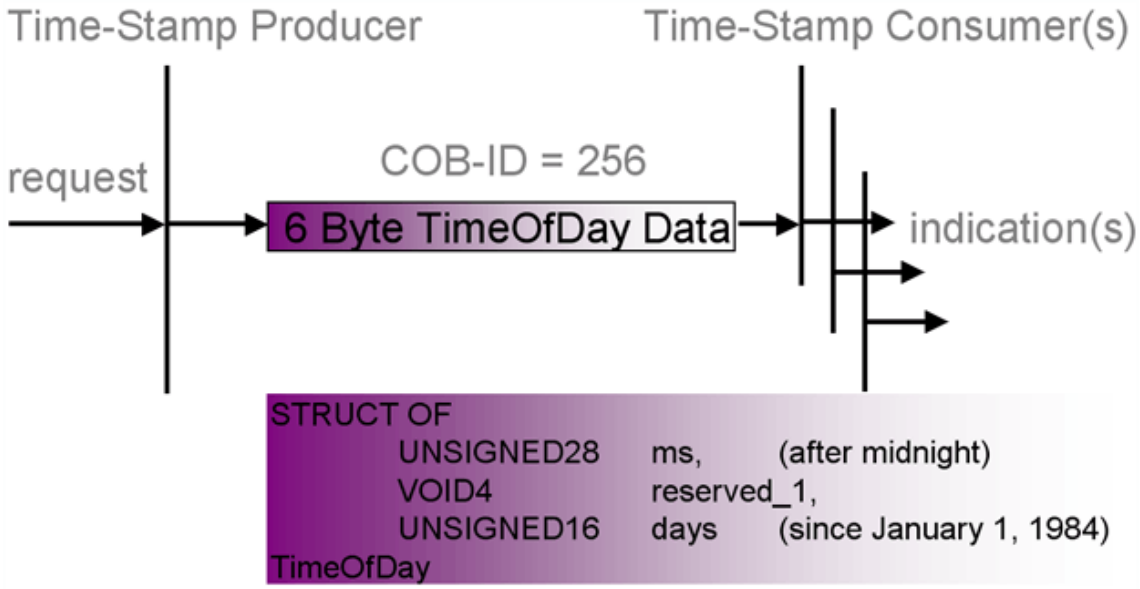


Figura 35: Protocolo Time-Stamp.

3.2.15 Emergency Object (EMCY).

Este mensaje se dispara cuando ocurre una situación de error interno no recuperable en el dispositivo y es enviado con un nivel de prioridad elevado al consumidor de la trama.

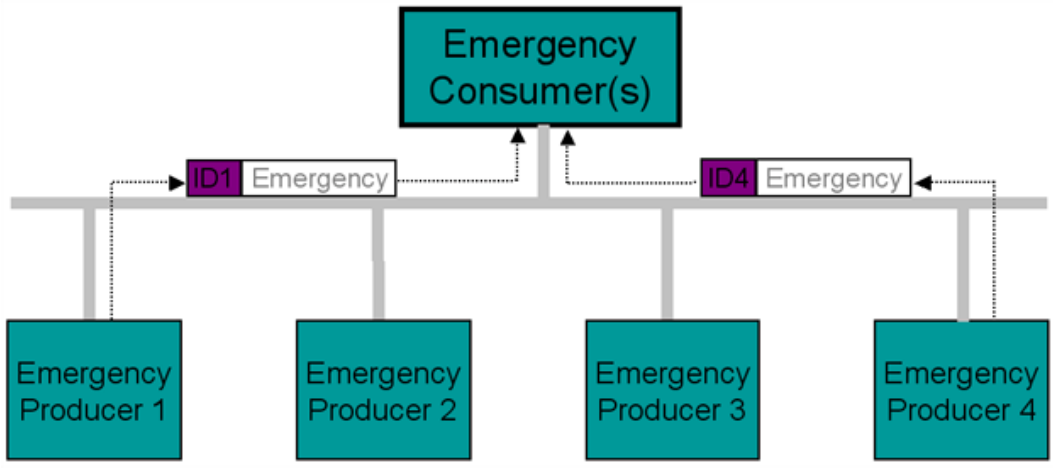


Figura 36: Objeto de Emergencia.

Un mensaje de emergencia sólo puede ser enviado una vez por cada evento de error producido, es decir, ningún mensaje de error debe ser enviado hasta que no se produzca un nuevo error en el dispositivo. En la siguiente tabla se muestran los códigos de error:

00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	device input side
22xx	inside of device
23xx	device output side
30xx	Voltage
31xx	main
32xx	inside of device
33xx	output
40xx	Temperature
41xx	ambient
42xx	device
50xx	Device Hardware

60xx	Device Software
61xx	internal
62xx	user
63xx	data set
70xx	Additional Modules
80xx	Monitoring
81xx	communication
8110	CAN overrun
8120	Error Passive (EP)
8130	Life Guard Error
8140	recovered from Bus-off
82xx	Protocol Error
8210	PDO not processed
8220	length exceeded
90xx	External Error
F0xx	Additional Functions
FFxx	Device Specific

Tabla 8: Códigos de Error.

Los códigos de errores, el registro de errores y la información adicional específica de cada componente se describen con más detalle en el perfil de dispositivo.

3.2.16 Protocolo del Emergency Object.

El objeto de emergencia es opcional y si esta implementado como mínimo debe soportar los códigos de error **00xx** (reset de error ó sin error) y **10xx** (error genérico). Es un servicio sin confirmar y sigue el modo Push del modelo de comunicaciones producer/consumer.

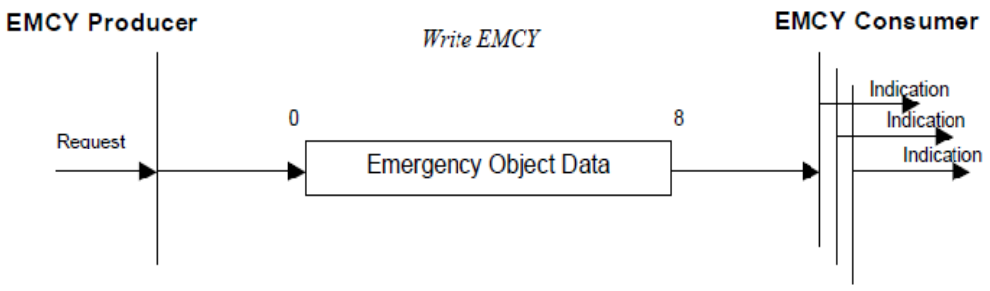


Figura 37: Protocolo del Emergency Object.

La trama enviada en un objeto de emergencia contiene el identificador con valores comprendidos entre 129 y 255, el código de error, el valor del registro de errores (objeto 1001h) y el campo de especificación de errores del fabricante.

byte	0	1	2	3	4	5	6	7
Contenido	Emergency Error Code		Error Register (object 1001h)	Manufactured SpecificError Field				

Tabla 9: Datos Objeto de Emergencia.

3.2.17 Network Management Object (NMT).

El Network Management está orientado al Nodo y siguen una estructura Maetro/Esclavo. Esto requiere que el NMT de un dispositivo funcione como Maestro en el Bus, mientras que los demás cumplan la función de NMT esclavos. El Network Management provee de las siguientes funcionalidades:

- Servicios de control de módulos para la inicialización de los NMT de los esclavos que tomen parte en la aplicación del sistema distribuido.
- Servicios de control de errores para la supervisión de Nodos y el estado de comunicación del Bus.
- Servicios de control de configuración para carga y descarga de datos de configuración de los módulos del Bus.

Un NMT esclavo representa la parte del Nodo que es responsable de la funcionalidad de este y está identificado únicamente por el Nodo-ID, con un valor comprendido dentro del rango [1… 127].

3.2.18 Protocolo del Control de Módulos.

A través del servicio de control de módulos, el NMT Maestro controla el estado de los NMT Esclavos mediante los atributos de estado que puede ser uno de los siguientes:

- PARADO.
- PRE-OPERACIONAL.

- OPERACIONAL.
- INICIALIZACIÓN.

El control del los módulos puede realizarse simultáneamente con todos los Nodos conectados al Bus (utilizando el ID 0) ó Nodo a Nodo (utilizando el Node-ID). El NMT Maestro controla su propio estado mediante el servicio local. A continuación se detallan los servicios de control de módulos:

Start Remote Node: Este servicio permite al NTM Maestro activar el estado OPERACIONAL del NMT Esclavo seleccionado.

Stop Remote Node: Permite seleccionar el estado PARADO del NMT esclavo seleccionado.

Enter Pre-Operational: Permite seleccionar el estado PRE-OPERACIONAL de los NMT esclavos.

Reset Node: A través de este servicio el NMT Maestro resetea la aplicación del NMT esclavo seleccionado, independientemente del estado en el que se encuentre este.

Reset Communication: este servicio provoca un reseteo en las comunicaciones del NMT esclavo seleccionado, y al igual que el Reset Node, no depende del estado anterior del NMT esclavo seleccionado.

Los servicio descritos anteriormente son servicios sin confirmar, obligatorio para todos los Nodos esclavos y que requiere los parámetros de la siguiente tabla.

Parameter	*Indication/Request*
Argument Node-ID All	**Mandatory** selection selection

Tabla 10: Parámetros para el control de los Módulos.

Estos servicios siguen el modelo de comunicaciones Maestro/Esclavo representado

en la siguiente figura:

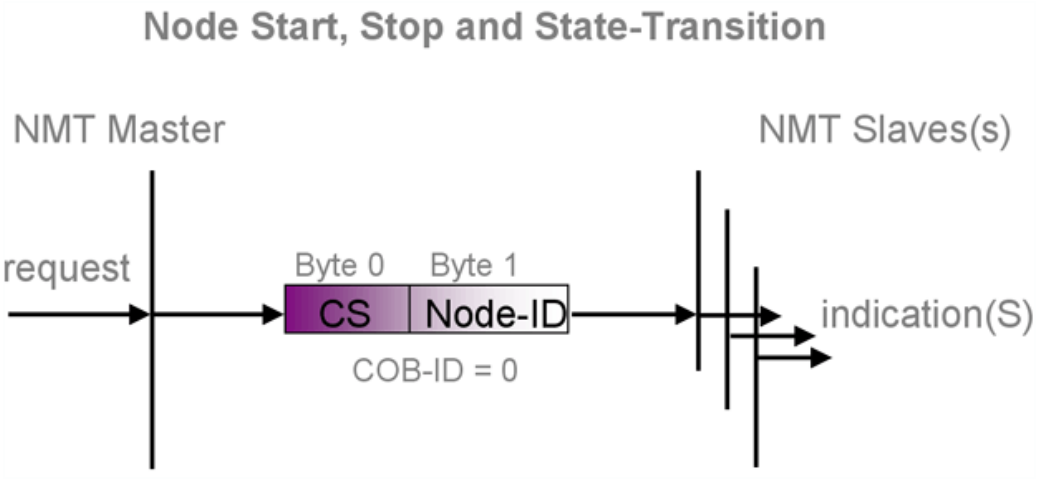


Figura 38: Protocolo de Control de Módulos.

Los distintos servicios se distinguen por el comando específico (CS):

- **Start Remote Node (CS=1),**
- **Stop Remote Node (CS=2),**
- **Enter Pre-Operational (CS=128),**
- **Reset Node (CS=129) and**
- **Reset Communication (CS=130).**

El objeto siempre tiene asignado el identificador 0 y consta de dos bytes, donde el byte 0 es el comando específico y el byte 1 es El identificador del Nodo al que se dirige la trama. Para direccionar todos los Nodos conectados al Bus simultáneamente el Node-ID debe ser igual a 0.

3.2.19 Protocolo del Control de Errores.

A través de este servicio el NMT detecta errores en el Bus, principalmente mediante emisiones periódicas de mensajes predefinidos que pueden provocar un cambio de estado ó detener procesos de reajuste en caso de detectar un error local. Existen dos posibles formas de detección de errores:

- **Node Guarding:** mediante este servicio el NMT Maestro chequea el estado de cada NMT Esclavo a intervalos de tiempos regulares. Este intervalo de tiempo está definido por dos parámetros guardados en el Diccionario de

Objetos, el **guard time** y el **life time factor**.

- **Heartbeat:** este servicio produce un mensaje cíclicamente en el que informa que no se ha caído de la red y el estado actual del Nodo sin necesidad de chequeo por parte del maestro.

Estos servicios son opcionales y en caso de que se implementen, no podrán funcionar simultáneamente. Para más información consultar [2].

3.2.20 Protocolo del Bootup.

A través de este servicio el NMT Esclavo informa de que se ha producido una transición en su estado local pasado de INITIALISING a PRE-OPERATIONAL, es decir, este servicio se produce después de un Reset Communication, Reset Aplication ó tras activar el dispositivo.

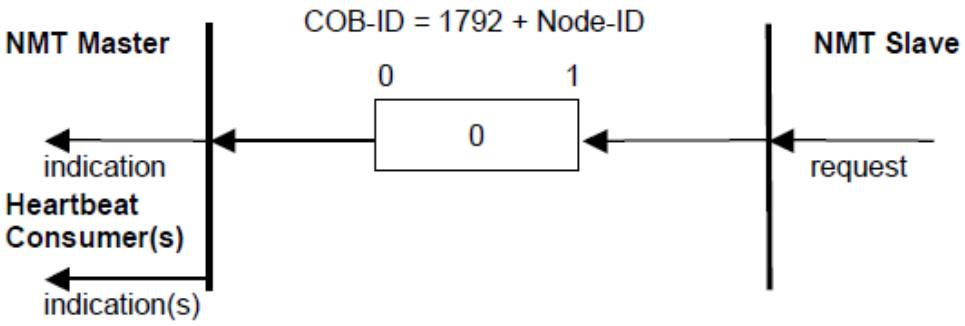


Figura 39: Protocolo Bootup.

Tras realizar el servicio de Bootup el maestro puede inicializar la configuración del Nodo si fuese necesario y solicitar que el Nodo entre en estado OPERATIONAL.

3.3 Máquina de Estado.

El NMT Esclavo de un dispositivo CANopen implementa una máquina de estados que permite después de un reset ó la activación del dispositivo entrar automáticamente en el estado PRE-OPERACIONAL desde el estado de INICIALIZACIÓN, informando de esta situación mediante el mensaje de Boot up. En este estado el Nodo es opcionalmente configurado y parametrizado mediante SDOs y posteriormente llevado al estado OPERACIONAL donde puede comenzar a emitir mensajes PDOs.

El diagrama de la máquina de estados se representa en la figura siguiente, y en la tabla se especifican los servicios requeridos para pasar de un estado a otro.

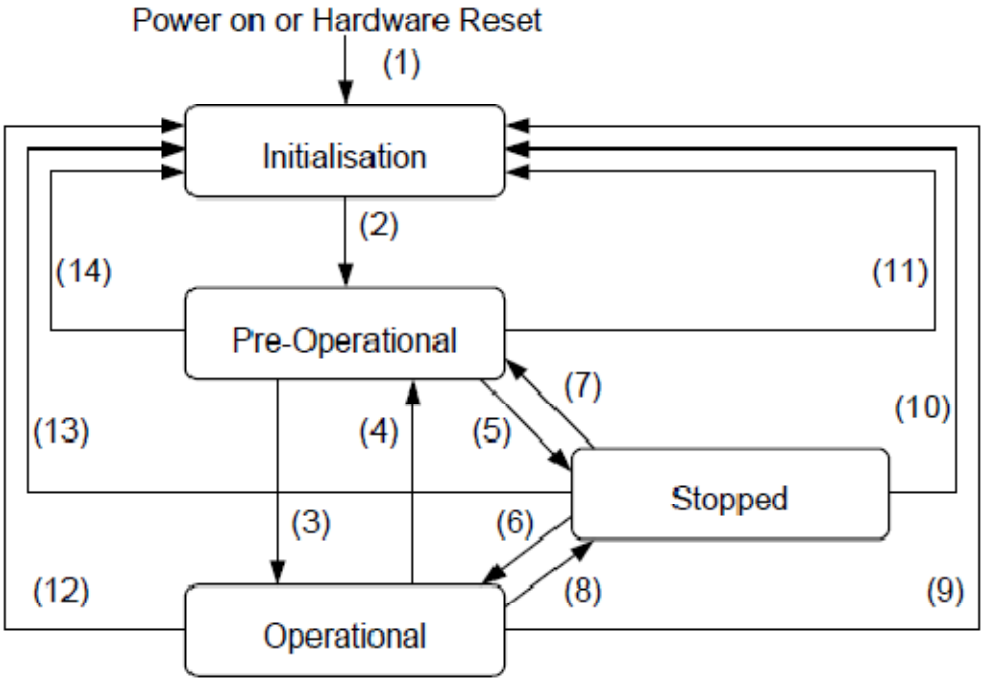


Figura 40: Diagrama de Estados.

(1)	Tras la activar el dispositivo **entra en el estado INITIALISATION automáticamente**.
(2)	Inicialización finalizada - **entra en PRE-OPERATIONAL automáticamente**.
(3),(6)	Servicio **Start_Remote_Node**.
(4),(7)	Servicio **Enter_PRE-OPERATIONAL**.
(5),(8)	Servicio **Stop_Remote_Node**.
(9),(10),(11)	Servicio **Reset_Node**.
(12),(13),(14)	Servicio **Reset_Communication**.

Tabla 11: Servicios de Transición de Estados.

En la tabla anterior se pueden observar los servicios necesarios para pasar de un estado a otro según la numeración entre paréntesis.

3.3.1 Estado de INICIALIZACIÓN.

Este estado está dividido en tres sub-estados para permitir un reset parcial ó completo del Nodo:

Initialising: es el primero de los sub-estados en el que el Nodo entra después de activar el dispositivo ó resetearlo por Hardware. Desde este sub-estado se pasa automáticamente a Reset_Application tras producirse la inicialización básica del Nodo.

Reset_Application: en este sub-estado los valores de áreas del perfil de fabricación y los valores de área de dispositivo estandarizado recuperan sus valores predefinidos y se produce la transición al estado Reset_Communication automáticamente.

Reset_Communication: al entrar en este sub-estado los parámetros del área de comunicaciones recuperan los valores predefinidos y se envía el mensaje de Bootup tras el cual el Nodo entra en el estado preoperacional.

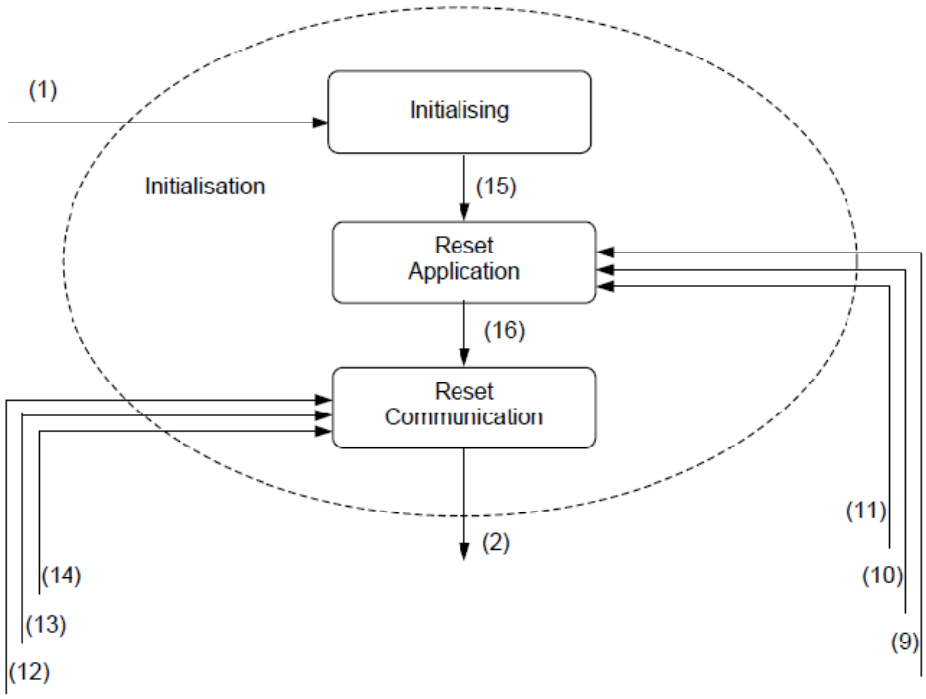


Figura 41: Sub-estados de Inicialización.

Los valores predefinidos son los especificados por el perfil de comunicaciones y dispositivos según CiA /2/, aunque si el dispositivo permite modificar

dinámicamente valores almacenados en la memoria de programa, pueden ser modificados.

La numeración entre paréntesis indica el servicio necesario para llegar al estado deseado y sigue la tabla descrita anteriormente con el añadido de los casos (15) y (16) que son transiciones que se producen automáticamente.

3.3.2 Estado PRE-OPERACIONAL:

En el estado PRE-OPERACIONAL la configuración de PDOs, los parámetros del dispositivo y la asignación de objetos de aplicación (PDO-maping) pueden ser realizados por medio de SDOs para una determinada configuración de aplicación. El dispositivo puede pasar directamente al estado OPERATIONAL por medio del servicio Start_Remote_Node si no necesita ser configurado previamente.

En este estado la comunicación por medio de PDOs no está permitida.

3.3.3 Estado OPERATIONAL:

En el estado OPERATIONAL todos los objetos de comunicación están activados permitiendo crear todos los PDOs según los parámetros descritos en el Objeto de Diccionario y acceder a este último por medio de los SDOs. Aunque requiere limitar el acceso a ciertos objetos que no pueden ser modificados durante la ejecución de la aplicación.

3.3.4 Estado PARADO:

Cuando el dispositivo entra en el estado STOPPED se detiene toda la comunicación a excepción del node guarding ó heartbeat, si están activos, y los servicios de control de módulos del NMT. Este estado es usado para realizar ciertos comportamientos de aplicación que están definidos en el perfil de un dispositivo CANopen.

3.3.5 Relación entre estados y objetos de comunicaciones.

En la siguiente tabla se muestra una relación entre los Objetos de Comunicación y el estado en el que el dispositivo puede ejecutar los servicios que permiten dichos

Objetos de Comunicación.

	INITIALISING	PRE-OPERATIONAL	OPERATIONAL	STOPPED
PDO			X	
SDO		X	X	
Synchronisation Object		X	X	
Time Stamp Object		X	X	
Emergency Object		X	X	
Boot-Up Object	X			
Network Management Objects		X	X	X

Tabla 12: Relación Estados - Objetos de Comunicaciones.

3.4 Esquema de asignación de identificadores CANopen.

Para reducir el esfuerzo de configuración y simplificar la red, CANopen define por defecto un esquema de asignación de identificadores y de carácter obligatorio. Estos identificadores predefinidos están accesibles en el estado PRE-OPERATIONAL después de la inicialización (si no se han realizado modificaciones). Los Objetos de SYNC, TIME STAMP, EMERGENCY y PDOs pueden ser eliminados y recreados por medio de una distribución dinámica en la que un dispositivo asigna los identificadores correspondientes sólo para los Objetos soportados.

Los identificadores predefinidos están formados por dos campos. El primero es el código de funcional, compuesto por los 4 bits más significativos, que determina la prioridad del Objeto. Y el segundo es el Node-ID que está formado por los 7 bits menos significativos y hace una distinción entre los distintos dispositivos con una misma funcionalidad.

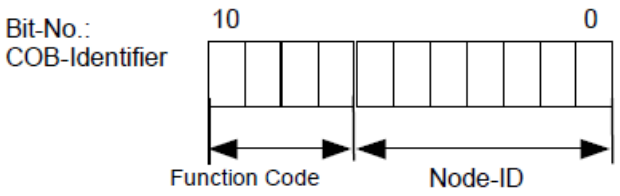


Figura 42: Esquema de asignación de identificadores.

Este esquema de asignación de identificadores permite a un Nodo maestro establecer comunicación punto a punto con hasta 127 Nodos esclavos. Además de permitir el modo de comunicación **broadcasting** con objetos de comunicación que no necesitan confirmación (NTM, SYNC y TIME_STAMP). Para realizar la comunicación **broadcasting** el Node-ID utilizado es el 0.

Por defecto el protocolo CANopen define un Objeto de Sincronismo, un Objeto de Emergencia, un SDO, un máximo de 4 Recive-PDO y 4 Transmit-PDO y el objeto de NMT según se puede observar en las siguientes tablas.

object	function code (binary)	resulting COB-ID	Communication Parameters at Index
NMT	0000	0	-
SYNC	0001	128 (80h)	1005h, 1006h, 1007h
TIME STAMP	0010	256 (100h)	1012h, 1013h

Tabla 13: Objetos de Comunicación Broadcast.

object	function code (binary)	Resulting COB-IDs	Communication Parameters at Index
EMERGENCY	0001	129 (81h) – 255 (FFh)	1014h, 1015h
PDO1 (tx)	0011	385 (181h) – 511 (1FFh)	1800h
PDO1 (rx)	0100	513 (201h) – 639 (27Fh)	1400h
PDO2 (tx)	0101	641 (281h) – 767 (2FFh)	1801h
PDO2 (rx)	0110	769 (301h) – 895 (37Fh)	1401h
PDO3 (tx)	0111	897 (381h) – 1023 (3FFh)	1802h
PDO3 (rx)	1000	1025 (401h) – 1151 (47Fh)	1402h
PDO4 (tx)	1001	1153 (481h) – 1279 (4FFh)	1803h
PDO4 (rx)	1010	1281 (501h) – 1407 (57Fh)	1403h
SDO (tx)	1011	1409 (581h) – 1535 (5FFh)	1200h
SDO (rx)	1100	1537 (601h) – 1663 (67Fh)	1200h
NMT Error Control	1110	1793 (701h) – 1919 (77Fh)	1016h, 1017h

Tabla 14: Objetos de Comunicación Punto a Punto.

3.5 Diccionario de Objetos.

La parte más importante del perfil de un dispositivo es la descripción del Diccionario de Objetos que es esencialmente un grupo de Objetos accesibles de forma predefinida a través del Bus. Los Objetos del diccionario están direccionados mediante un índice codificado en 16 bits que permite un máximo de 65536 entradas en el Diccionario de Objetos.

En este documento no se hará una descripción individual de cada Objeto del Diccionario, ya que esto supondría entenderse mucho además de que esta información puede consultarse en el protocolo DS-301[2], para los Objetos estándares, y en el perfil específico de cada dispositivo CANopen, para los Objetos específicos. El perfil específico de un encoder está definido en el documento DS-406[4].

3.6 Uso de Index y Sub-Index (Multiplexor).

En el caso de que la entrada referencie una variable simple, el índice (**Index**) referencia el valor de esta entrada directamente. Si los datos referenciados son de tipo complejo, el índice referencia a toda la estructura y para permitir el acceso individual a los elementos de una estructura compleja se ha definido el subíndice (**Sub-Index**). El cual referencia a un campo de datos dentro de una estructura apuntada por el índice principal. El subíndice tendrá el valor 00h cuando los datos referenciados sean de tipo simples (UNSIGNED8, BOOLEAN, INTEGER32, etc.). El conjunto formado por el índice y subíndice se conoce como **Multilexor.**

En el caso de que la entrada sea de tipo compleja el subíndice 00h contendrá el valor de los subíndices soportados por el Objeto, codificado como un UNSIGNED8, sin tener en cuenta los subíndices 00h y FFh. El subíndice FFh opcionalmente se utiliza para describir la entrada por medio del tipo de datos y tipo de Objeto de la entrada, codificado como un UNSIGNED32 según se muestra en la siguiente tabla.

	MSB		LSB
bits	31-16	15-8	7-0
valor	reservado (valor: 00 00h)	Data Type	Object Type
codificado	-	UNSIGNED8	UNSIGNED8

Tabla 15: Estructura del Subíndice FFh.

Los tipos de datos están descritos en la tabla 39 y los tipos de Objetos en la tabla 37 del protocolo DS-301.

3.7 Estructura general del Diccionario de Objetos.

Las estradas al Diccionario de Objetos siguen la siguiente tabla.

Index (hex)	Object (Symbolic Name)	Name	Type	Attrib.	M/O

Tabla 16: Formato de Cabecera del Diccionario de Objetos.

La columna **Index** indica la posición del Objeto dentro del Diccionario actuando como una dirección para referenciar el campo de datos deseado. El **Sub-Index,** utilizado para referenciar campos de datos dentro de una estructura compleja, no está referenciado en esta tabla.

La columna **M/O** define si el objeto es de carácter obligatorio (**M**andatory) ó opcional (**O**ptional). Los Objetos obligatorios deben ser implementados en todos los dispositivos y los opcionales sólo se implementaran para proporcionar características adicionales. Esto puede requerir implementar otros Objetos relacionados.

La columna **Name** representa una descripción textual de la funcionalidad del Objeto y la columna **Type** define el tipo de dato que contiene el Objeto según el apartado 9.1 del protocolo DS-301. Mientras que **Attribute** define el correcto acceso al Objeto desde el punto de vista del Bus, pudiendo tomar los valores de la siguiente tabla.

Attribute	Description
rw	read and write access
wo	write only access
ro	read only access
Const	read only access, value is constant

Tabla 17: Atributos de Acceso a los Objetos de Datos.

La columna **Object** contiene el nombre del Objeto de acuerdo con la siguiente tabla

y es usado para describir el tipo de Objeto que contiene un índice particular dentro del Diccionario de Objetos.

Object Name	Comentario	Código de Objeto
NULL	Entrada en el Diccionario sin campo de datos.	0
DOMAIN	Gran cantidad de datos variables (ej. código ejecutable).	2
DEFTYPE	Representa tipos de datos definidos (boolean, UNSIGNED16, float, etc.).	5
DEFSTRUCT	Define un nuevo tipo de dato record (ej. Estructura del PDOMaping del índice 20h)	6
VAR	Representa un valor simple (ej. UNSIGNED8, Boolean, float, etc.).	7
ARRAY	Representa un Objeto con múltiples campos de datos donde cada campo es una variable simple de algún tipo de dato básico (ej. ARRAY de UNSIGNED16). El subíndice 00h es un UNSIGNED8 que no forma parte del ARRAY.	8
RECORD	Representa un Objeto con múltiples campos de datos donde los campos de datos pueden ser cualquier combinación de variables simples. El subíndice es un UNSIGNED8 que no forma parte del RECORD.	9

Tabla 18: Tipos de Objetos del Diccionario.

3.8 Componentes del Diccionario de Objetos.

Los Objetos del Diccionario estándar siguen una estructura muy similar a otros protocolos de comunicación serie industriales. Esta estructura se muestra en la siguiente tabla:

Index (hex)	Object
0000	not used
0001-001F 0020-003F	Static Data Types Complex Data Types
0040-005F	Manufacturer Specific Complex Data Types
0060-007F	Device Profile Specific Static Data Types
0080-009F	Device Profile Specific Complex Data Types
00A0-0FFF	Reserved for further use
1000-1FFF	Communication Profile Area
2000-5FFF	Manufacturer Specific Profile Area
6000-9FFF	Standardised Device Profile Area
A000-BFFF	Standardised Interface Profile Area
C000-FFFF	Reserved for further use

Tabla 19: Estructura del Diccionario de Objetos.

Los tipos de datos estáticos contenidos desde el índice 0001h al 001Fh contienen datos de tipo estándar (booleanos, enteros, cadenas de caracteres, números en coma flotante, etc.). Estas entradas se incluyen sólo como referencia y no pueden ser leídos ni escritos.

Desde el índice 0020h hasta el 003Fh se encuentran los tipos de datos complejos que son estructuras predefinidas compuestas por tipos de datos estándares y que son comunes a todos los dispositivos CANopen. Hasta el momento sólo está definido el rango [20h… 23h] y los demás están reservados para futuros usos.

Los tipos de datos complejos de especificaciones de fabricante se encuentran desde el índice 0040h al 005Fh y al igual que los anteriores están compuestos de tipos de datos estándares pero son específicos para cada dispositivo en particular. En este rango el fabricante puede definir libremente sus propias estructuras de datos.

El perfil de dispositivo puede definir adicionalmente tipos de datos específicos que pueden ser estáticos (contenidos del índice 0060h al 007Fh) ó de tipo complejos (contenidos del índice 0080h al 009Fh).

El dispositivo puede opcionalmente proveer una estructura de los tipo de datos complejos soportados (rangos de índices [0020h… 005Fh] y [0080h… 009Fh]) que permite el acceso de lectura a los correspondientes índices. En este caso el subíndice 0h contendrá el número de entradas que contiene dicho índice y los siguientes subíndices contendrán la información del tipo de dato que soportan codificado como un UNSIGNED8.

Subindex	Value	(Description)
0h	04h	(4 sub indices follow)
1h	07h	(UNSIGNED32)
2h	05h	(UNSIGNED8)
3h	06h	(UNSIGNED16)
4h	05h	(UNSIGNED8)

Tabla 20: Ejemplo de Tipos de Datos Complejos.

El área de perfil de comunicaciones (índices del 1000h al 1FFFh) contiene los parámetros específicos de comunicaciones para el Bus y son comunes para todos

los dispositivos.

El rango de índices del 2000h al 5FFFh está libre para definir perfiles específicos del fabricante.

El área de perfil estandarizado (desde el índice 6000h al 9FFFh) contiene todos los Objetos de datos comunes de un dispositivo que pueden ser leídos ó escritos desde el Bus. El perfil del dispositivo puede usar estas entradas para describir sus parámetros y funcionalidad utilizando un máximo de 800h (2048) índices para definir cada perfil, lo que permite definir hasta 8 perfiles en cada Nodo CAN. En el caso de que se defina más de un perfil hablaríamos de Módulos de dispositivos Múltiples y el rango de índices [6000h… 9FFFh] estaría dividido según la siguiente tabla.

6000h to 67FFh	device 0
6800h to 6FFFh	device 1
7000h to 77FFh	device 2
7800h to 7FFFh	device 3
8000h to 87FFh	device 4
8800h to 8FFFh	device 5
9000h to 97FFh	device 6
9800h to 9FFFh	device 7

Tabla 21: Rangos de Índices para Perfil de dispositivo.

3.9 Especificaciones de los tipos de datos.

Los tipos de datos estáticos están introducidos en el Diccionario sólo para propósitos de definición. Sin embargo los índices en el rango [0001h… 0007h] pueden ser mapeados para definir el espacio apropiado en un RPDO. Mientras que los índices en el rango [0009… 000Bh] y el 000Fh no pueden ser mapeados.

Los dispositivos CANopen no necesitan implementar todos los tipos de datos predefinidos, sino que solamente deberán soportar los tipos de datos utilizados en los objetos comprendidos en el rango [1000h… 9FFFh]. La representación de los tipos de datos utilizados está definido en el apartado 9.1 del protocolo DS-301 y el

orden de estos se puede observar en la tabla 39 del mismo documento.

Los datos de tipo complejos están definidos después de los estáticos y hasta el momento sólo están definido 4 tipos, los parámetros de comunicación PDO (PDO_COMMUNICATION_PARAMETER), los parámetros de mapeado PDO (PDO_MAPPING), los parámetros SDO (SDO_PARAMETER) y las especificaciones de Identidad del dispositivo (IDENTITY), todos ellos de tipo record. Estos tipos de datos se encuentran en los índices 20h, 21h, 22h y 23h respectivamente.

3.10 Tipos de datos complejos predefinidos.

Este apartado define los tipos de datos complejos predefinidos usado en la comunicación. En las siguientes tablas se muestra la estructura de estos datos y los rangos de valores y significado de los mismos se detallan en la definición de los Objetos que utilizan estos tipos de datos.

Index	Sub-Index	Field in PDO Communication Parameter Record	Data Type
0020h	0h	number of supported entries in the record	UNSIGNED8
	1h	COB-ID	UNSIGNED32
	2h	transmission type	UNSIGNED8
	3h	inhibit time	UNSIGNED16
	4h	reserved	UNSIGNED8
	5h	event timer	UNSIGNED16

Tabla 22: Especificaciones de los Parámetros de Comunicación PDO.

Index	Sub-Index	Field in PDO Parameter Mapping Record	Data Type
0021h	0h	number of mapped objects in PDO	UNSIGNED8
	1h	1st object to be mapped	UNSIGNED32
	2h	2nd object to be mapped	UNSIGNED32
...	...	...	...
	40h	64th object to be mapped	UNSIGNED32

Tabla 23: Especificaciones de los Parámetro de Mapeado PDO.

Index	Sub-Index	Field in SDO Parameter Record	Data Type
0022h	0h	number of supported entries	UNSIGNED8
	1h	COB-ID client -> server	UNSIGNED32
	2h	COB-ID server -> client	UNSIGNED32
	3h	node ID of SDO's client resp. server	UNSIGNED8

Tabla 24: Especificaciones de los Parámetros SDO.

Index	Sub-Index	Field in Identity Record	Data Type
0023h	0h	number of supported entries	UNSIGNED8
	1h	Vendor-ID	UNSIGNED32
	2h	Product code	UNSIGNED32
	3h	Revision number	UNSIGNED32
	4h	Serial number	UNSIGNED32

Tabla 25: Especificaciones de Identidad.

Capítulo 4

ENCODER ABSOLUTO.

4. ENCODER ABSOLUTO.

En este capítulo se explicará en detalle los pasos seguidos para el diseño del Nodo CANopen con el perfil específico de un encoder absoluto. Como ya se ha comentado anteriormente, el perfil específico de un encoder CANopen está definido por CiA en el documento DS-406 (Device profile for encoders). Este documento es común para todos los tipos de encoder, por lo que, en primer lugar, hay que tener claro el tipo de encoder que se va a diseñar, que en este caso es un **encoder absoluto rotacional** con las siguientes características:

- Bajo peso.
- Tamaño reducido.
- Fácil integración.
- Alta resolución.
- Capaz de medir posición de 0 a 360º.

El capítulo estará dividido en un primer apartado en el que se justificará la elección de los componentes utilizados y seguidamente se hará una descripción del diseño Hardware y Software. Por último se hará una descripción de la implementación y puesta en marcha del encoder.

4.1 Selección de componentes.

El componente principal y más importante del diseño del Nodo es el encoder comercial que se utilice. Este elemento va a condicionar todos los parámetros relacionados con la lectura de la posición, como pueden ser precisión, velocidad de lectura, fiabilidad, etc. Otra de las características a tener muy en cuenta es el modo de lectura de los datos que proporcione dicho encoder y la posibilidad de conexión con los demás elementos que formen el Nodo CANopen.

Teniendo en cuenta lo mencionado, se ha seleccionado un encoder absoluto de la casa Agilent Technologies y en concreto el modelo **AEAS-7000**. Este encoder, además de caracterizarse por su reducido tamaño, alta precisión y montaje modular formado por un disco y la cabeza lectora, permite una fácil adaptación a

nuestro diseño. Este encoder proporciona la lectura de posición en código gray y con una precisión de hasta 16 bits.

Figura 43: Encoder Absoluto Agilent AEAS-7000.

Una vez que tenemos claro cuál es el encoder comercial que se va a utilizar, el siguiente paso es seleccionar todos los componentes que posibilitaran la lectura del mismo y la integración en un Bus CANopen. Estos elementos ya han sido descritos anteriormente en el apartado "Elementos que componen un Nodo CANopen" y son los siguientes:

- Microcontrolador.
- Controlador CAN-Bus.
- Transmisor/Receptor (Transceiver).

En cuanto a la lectura, el AEAS-7000 incorpora un puerto serie denominado SPI (Serial Peripheral Interface) y que está muy extendido entre todos los fabricantes de microcontroladores, por lo que este no va a ser un factor determinante en la elección del micro. Lo mismo pasaría con el controlador CAN y el transceiver, ya que son componentes estandarizados y que muchas empresas suministran con unos costes muy similares. Por lo que el factor determinante a la hora de decidir que modelos serian utilizados, fue el soporte que los fabricantes de estos componentes ponen a disposición de los diseñadores de estos sistemas y en este sentido **Microchip®** es una gran opción.

Microchip® además de ofrecer una completa línea de productos para el diseño de aplicaciones embebidas usando el protocolo CAN, también ofrece un entorno de programación muy potente, especialmente por las librerías que incorpora, entornos de desarrollo y application notes fáciles de utilizar, de bajo coste y que simplifican y facilitan en gran medida el desarrollo de estos diseños.

Dentro de la línea de productos CAN que Microchip® pone a disposición del diseñador nos encontramos con dos opciones. La primera de ellas es una solución basada en cualquiera de microcontrolador de 8 ó 16 bits que incorporen al menos dos puerto SPI, un controlador CAN modelo MCP2515 que se comunicara con el micro a través de uno de puerto SPI y un transceiver modelo MCP2551. El otro puerto SPI estaría dedicado a establecer la conexión entre el micromontrolador y el encoder ya que una de las especificaciones es que la lectura de la posición se realice a través de este puerto.

La segunda opción es utilizar un microcontrolador que integre un módulo CAN que cumpla con las funciones de controlador CAN. En la siguiente figura se puede observar el esquemático de ambas soluciones.

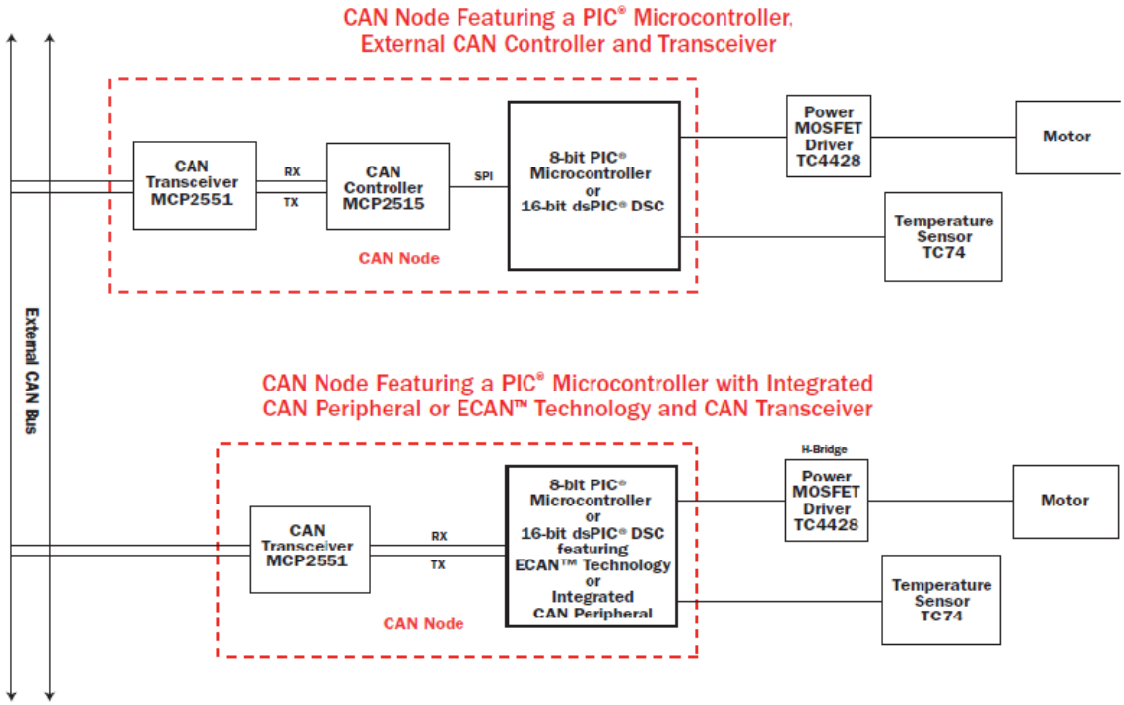


Figura 44: Esquemático de Soluciones CAN.

En este diseño se ha optado por la segunda opción, un microcontrolador que integre el módulo CAN y un tranceiver **MCP2551** por varias razones:

1. Este diseño permite ahorrar espacio en la placa al integrar el controlador CAN en el micro.
2. Simplifica el diseño Software ya que no es necesario programar el puerto SPI que comunica el microcontrolador con el controlador CAN.
3. El tiempo de ejecución de la aplicación se reduce ya que se elimina la necesidad de que todas las tramas CAN pasen a través del puerto SPI.
4. El coste total de los componentes se reduce.

Teniendo en cuenta que entre las características del microcontrolador han de estar presentes al menos un puerto SPI, para la lectura del encoder, e incorporar un módulo CAN se ha seleccionado el modelo **PIC18F2580**. Que además de reunir las características descritas, incorpora una memoria de programa de 32K, un TIMER de 8 bits y 3 de 16 bits. Características más que suficientes para afrontar el diseño del Nodo CAN.

4.1.1 Características del encoder Agilent AEAS-7000.

En la siguiente tabla se presentan las principales características eléctricas del encoder seleccionado, destacando especialmente su bajo consumo de corriente y que su tensión de alimentación es de +5v, la cual está accesible en el robot, por lo que facilita su integración.

Parámetro	Símbolo	Mínimo	Típico	Máximo	Unidades
Tensión de alimentación	VD	+45	+5.0	+5.5	V
Temperatura de funcionamiento	TA	-25	+25	+85	ºC
Nivel alto de entrada	Vih	0.7*VD	-	VD	V
Nivel bajo de entrada	Vil	0	-	0.3* VD V	
Corriente total	Itotal	-	+25	-	mA
Salida a nivel alto	Voh	VD- 0.5V	-	VD	V
Salida a nivel bajo	Vol	0	-	+0.5	V
Frecuencia de reloj	fclock	-	-	16	Mhz

Tabla 26: Principales Características Eléctricas del Encoder AEAS-7000.

Este encoder presenta diversas entradas y salidas a través de sus pines y mediante los cuales se puede configurar el dispositivo. En la siguiente figura se puede ver el Pinout del componente, explicándose a continuación la función de cada uno de ellos.

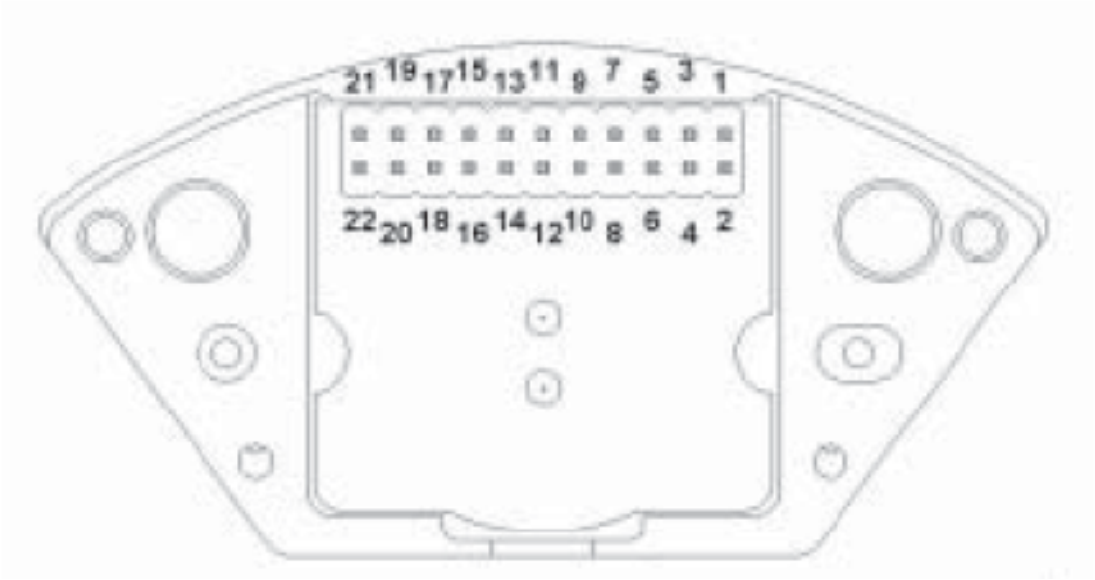


Figura 45: Pinout del Encoder.

- **Pin 1**: No se conecta.
- **Pin 2**: Entrada digital **KORR**, activa un código corrector de código Gray al encoder haciendo a la medida más real. El inconveniente es que para que esté activo, la resolución total del encoder es de 12 bits (4096 pulsos por vuelta) en vez de 16 bits. Por tanto, esta señal se pondrá a nivel alto si se quiere activar el código corrector o a nivel bajo si se deja desactivo y se quieren más pulsos por vuelta.
- **Pin 3, 4 y 5**: Señales **PROBE ON**, **PCL** y **STCAL**, estas entradas permiten la calibración del encoder la primera vez que se coloca para conseguir una perfecta alineación entre electrónica y disco, acción muy importante en elementos ópticos, para garantizar una lectura correcta.
- **Pin 6:** Entrada digital **MSBINV**, puesto a nivel alto esta señal se hace que se modifique el sentido de envío de los bits de lectura, es decir, si está a nivel bajo el bit más significativo del registro de lectura será enviado primero, invirtiéndose si el estado de la señal es a nivel alto.

- **Pin 7**: Entrada digital **DIN**, permite la configuración de varios encoders colocados en anillo para obtener la medida del que se desee.
- **Pin 8**: Entrada digital **NSL**, habilita el envío del dato a través del pin DOUT. Cuando se pone a nivel bajo el envío queda habilitado.
- **Pin 9**: Entrada digital **SCL**, marca la velocidad de lanzamiento de cada uno de los 16 bits que forman la lectura de la posición. En cada flanco de bajada lanzará un bit por la señal DOUT si NSL está habilitado.
- **Pin 10**: Salida digital **DOUT**, por esta patilla el encoder lanzará en serie los 16 bits del dato.
- **Pin 11 y 12**: Salida digital **D0** y **D09**, señales digitalizadas de A0 y A09, señales seno y coseno aportada por el encoder que pueden hacer que también se utilice como encoder relativo para el uso de un controlador de motores.
- **Pines 13 y 14**: Señales de alimentación, **+5v** y **GND** respectivamente.
- **Pin 15**: Señal **A09P**, señal de seno para realizar controles en amplificadores de motores.
- Pin 16: GND
- **Pin 17** señal **A0P** de coseno para realizar controles en amplificadores de motores.
- **Pin 18**: Señal **A09N**, señal de seno negativo para realizar controles en amplificadores de motores.
- **Pin 19**: Señal **+Vdda**, alimentación de referencia analógica.
- **Pin 20**: Señal **A0N**, señal de coseno negativo para realizar controles en amplificadores de motores.
- **Pin 21**: Salida digital **LERR**, bits de error. Avisa si se produce un fallo en la alimentación del fotodiodo, es decir si ésta es menor de 2.5v.
- **Pin 22**: Salida digital **LERD**, permite monitorizar el estado del fotodiodo.

4.1.2 Características de los componentes Microchip®.

Como ya se ha justificado en el capítulo dedicado a la selección de los componentes se utilizaran un microcontrolador **PIC18F2580** y un transceiver **MCP2551**, ambos de Microchip® y cuyas características se detallaran a continuación.

4.1.3 Microcontrolador PIC18F2580.

Las principales características del microcontrolador PIC18F2580 están resumidas en la tercera columna del siguiente cuadro en el que cabe destacar el **Módulo ECAN** y el **puerto de comunicaciones serie MSSP** que se describirán más detalladamente. Además este micro incluye 4 TIMER, de los cuáles uno será utilizado para controlar ciertas características temporales de los Nodos CANopen, y una memoria de programa de 32K, que permite hasta un máximo de 16384 instrucciones simples de programa.

Features	PIC18F2480	PIC18F2580	PIC18F4480	PIC18F4580
Operating Frequency	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz
Program Memory (Bytes)	16384	32768	16384	32768
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	1	1	1	1
Enhanced Capture/ Compare/PWM Modules	0	0	1	1
ECAN Module	1	1	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Parallel Communications (PSP)	No	No	Yes	Yes
10-Bit Analog-to-Digital Module	8 Input Channels	8 Input Channels	11 Input Channels	11 Input Channels
Comparators	0	0	2	2
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT
Programmable High/ Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set Enabled	75 Instructions; 83 with Extended Instruction Set Enabled	75 Instructions; 83 with Extended Instruction Set Enabled	75 Instructions; 83 with Extended Instruction Set Enabled
Packages	28-pin SPDIP 28-pin SOIC 28-pin QFN	28-pin SPDIP 28-pin SOIC 28-pin QFN	40-pin PDIP 44-pin QFN 44-pin TQFP	40-pin PDIP 44-pin QFN 44-pin TQFP

Figura 46: Características del PIC18F2580.

Módulo ECAN:

- Permite una tasa de transferencia de hasta 1Mbps.
- Cumple la normativa 2.0B según las especificaciones dadas por Bosch.
- Totalmente compatible con los módulos CAN incorporados en los PIC18XXX8.
- Tres modos de operación (Legacy, Enhanced Legacy, FIFO).
- Tres buffers dedicados a transmisión.
- Dos buffers dedicados a recepción.
- Seis buffers programables para transmisión ó recepción.
- Tres mascaras 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.

Puerto de comunicaciones serie MSSP: las siglas MSSP significan Master Synchronous Serial Port. Este puerto es capaz de soportar los 4 modos del puerto serie según el protocolo SPI y los modos Master y Slave del protocolo I^2C.

4.1.4 Transceiver MCP2551.

Este componente tiene como principales características que soportar la tasa de transferencia máxima de 1 Mbps que se especifica en el protocolo CANopen, implementa 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. A continuación se define el patillaje y funcionalidad de los mismos.

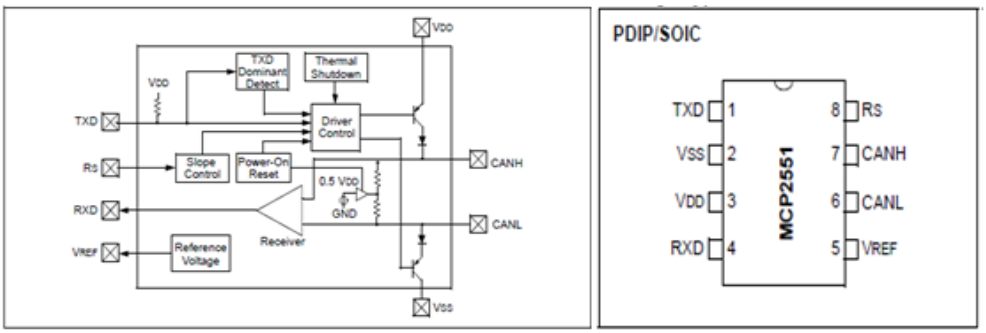


Figura 47: Diagrama de Bloques y patillaje del MCP2551.

Pin Number	Pin Name	Pin Function
1	TXD	Transmit Data Input
2	VSS	Ground
3	VDD	Supply Voltage
4	RXD	Receive Data Output
5	VREF	Reference Output Voltage
6	CANL	CAN Low-Level Voltage I/O
7	CANH	CAN High-Level Voltage I/O
8	RS	Slope-Control Input

Tabla 27: Descripción de los Pines del MCP2551.

- **Pin 1: TXD** entrada digital que recibe los datos del microcontrolador que van a ser enviados por el Bus.
- **Pin 2: V_{ss}** , referencia de tierra.
- **Pin 3: V_{dd}**, alimentación.
- **Pin 4: RXD**, señal digital que transmite los datos al microcotrolador recibidos del Bus.
- **Pin 5: V_{ref}**, 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: Rs**, 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** permitidas en el sistema. La tasa máxima de transferencia es controlada por medio de una resistencia externa (R_{EXT}) 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 RS 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 RS. Este modo se selecciona conectando una resistencia (REXT) entre el pin RS y masa. La gráfica siguiente ilustra los valores típicos de slew rate en función del valor de la resistencia externa.

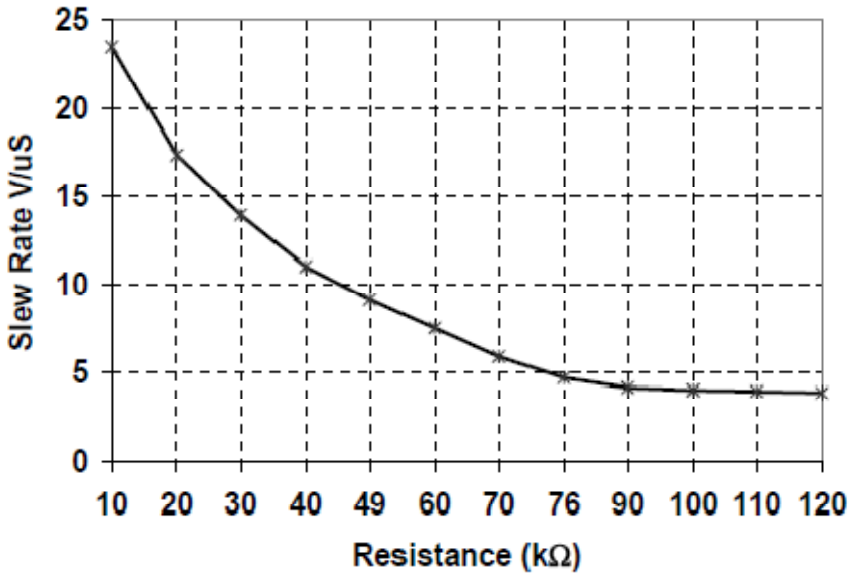


Figura 48: Slew Rate VS. REXT.

- **Standby**, permite introducir el transceiver en modo standby ó sleep cuando el valor de RS 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.

4.2 Diseño hardware.

Debido a que el robot tiene que estar operativo y sólo se puede implementar el encoder una vez que todo funcione de forma correcta, se ha tenido que desarrollar

una estructura mecánica, que incorpora un motor de continua, y permite colocar y centrar en una posición correcta la electrónica del encoder y el disco.

La estructura mecánica fija la cabeza lectora del encoder, cumpliendo la centralidad con respecto al eje donde ira posicionado el disco del encoder y la altura adecuada del fotodiodo al disco. Para cumplir esto, la cabeza trae dos tornillos que tienen que ir exactamente a un radio de 21.2 mm con respecto al centro de giro del sistema según se observa en la figura siguiente.

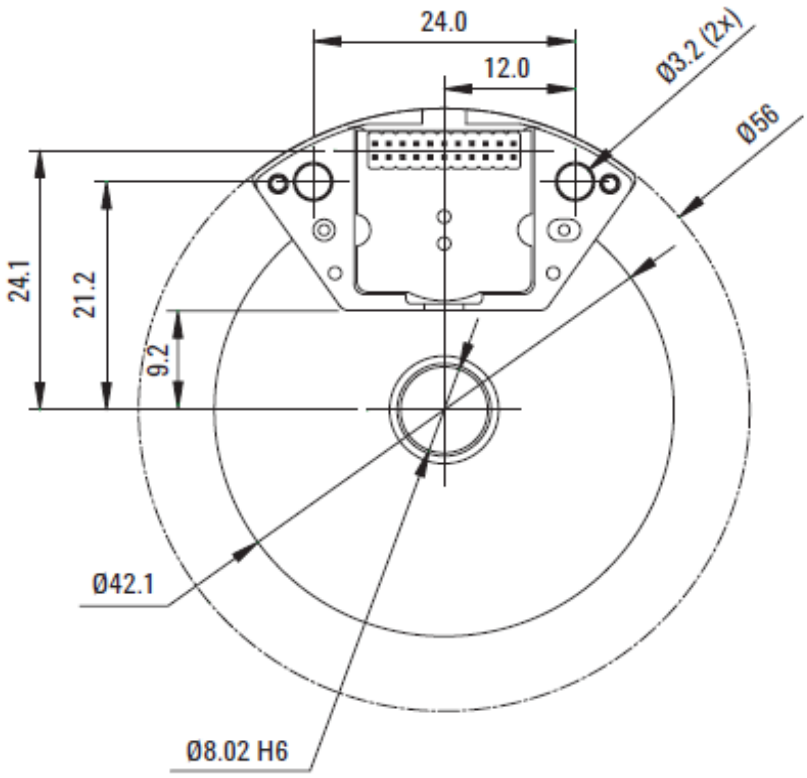


Figura 49: Vista en Planta de la Colocación del Encoder.

El disco ira colocado sobre un eje mediante un tornillo prisionero que hace que ambos giren solidarios y permite el acople al eje del motor de continua. En la siguiente figura se puede observar, en a) una foto de la estructura para situar la cabeza lectora, en b) una foto de la pieza que se adapta al eje y en c) el sistema final con el encoder ya colocado sobre el motor.

Figura 50: Sistema Mecánico Diseñado para el Desarrollo del Encoder.

El diseño de la placa PCB se ha diseñado de forma que permita una fácil integración de la cabeza lectora, por lo que se le ha dado la forma geométrica de una circunferencia. Además esta ha de ser lo más pequeña posible, siendo el diámetro de esta aproximadamente de 56 mm que es el diámetro de la circunferencia que forma el exterior del encoder con respecto al eje central.

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.

En cuanto al encoder la configuración es la siguiente:

- Salida **KOOR** conectada a nivel bajo, para tener una lectura de 16 bits.
- **PROBE_ON** conectado a **GND**, ya que, por recomendaciones del fabricante, es aconsejable que esté a Gnd durante su funcionamiento normal.
- **MSBINV** conectado a **GND** para que envíe primero el bit más significativo.
- El puerto **SPI** del encoder ha de estar conectado al mismo puerto del microcontrolador. Es decir, **SCL** debe conectarse al mismo pin del microcontrolador, el pin **DOUT** se conectara al pin **SDI** (Serial Data Input) del microcontrolador y el pin **NSL** se conectara a cualquier pin 11 del micro y hará la función de chip select.

Al microcontrolador, además de conectarse el encoder según se ha definido anteriormente, se conectaran también:

- El circuito de reset.
- Un cristal de cuarzo de 20MHz.
- Un LED para indicar el estado del Nodo.
- Un conector micro-macth de 6 señales, del que se utilizaran 5 para implementar el circuito de programa del microcontrolado que se describirá más adelante.
- Y las señales **CANRX** y **CANTX** se conectaran a las patillas **RXD** y **TXD** del transmisor MCP2551 respectivamente.

El transceiver MCP2551:

- Estará conectado al micro mediante las patillas **RXD** y **TXD** según se ha definido anteriormente.
- El pin RS se conectara a masa para seleccionar el modo **High-Speed**.
- Y los pines **CANL** y **CANH** se conectaran a dos conectores, uno de dos pines que permitirá conectar otros dispositivos al Bus CANopen y otro de tipo micro-macth de 4 señales, que además de conectarse al Bus, recibe las señales de alimentación de la placa (+5V y GND).

4.2.1 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 micro-macth de 6 pines que permite tener acceso a las patillas de programación del microntrolador. A este modo de programación se le conoce como programación ICSP (*In-Circuit 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.

En la siguiente figura se observan las conexiones del conector de programación con los pines del PIC, a través de los cuales se produce la reprogramación del microcontrolador.

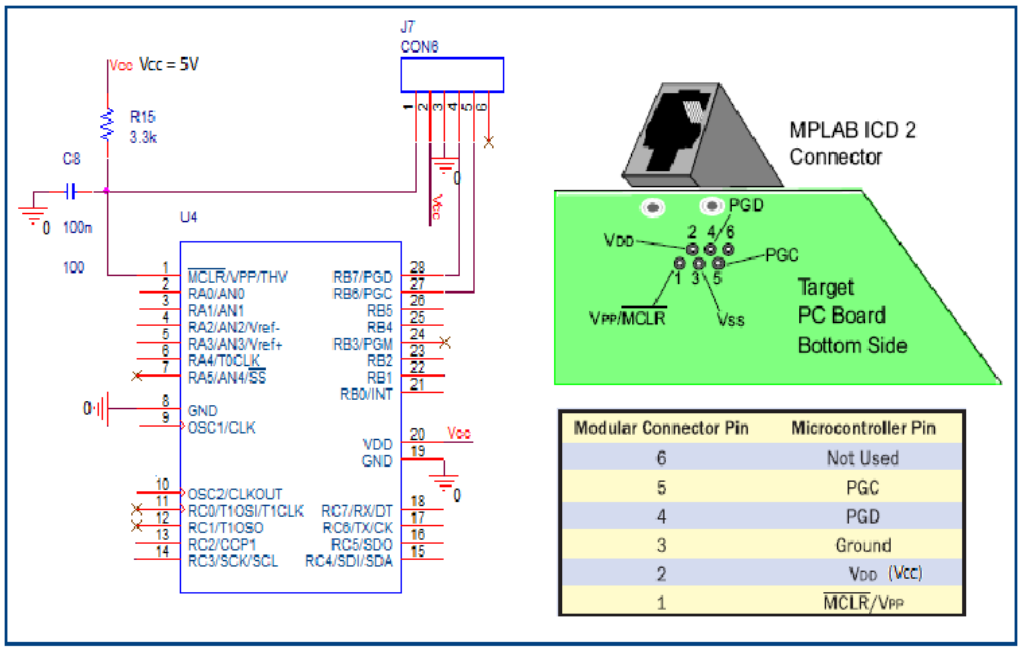


Figura 51: Conexiones del Circuito de Programación.

4.2.2 Esquemático final.

A continuación, en la siguiente figura se muestra el esquemático final, que

implementa todos los circuitos diseñados según lo descrito durante este capítulo. A partir de este esquemático se diseñará la placa de circuito impreso con *OrCAD Layout*, teniendo en cuenta los tamaños de los componentes, con el fin de conseguir una placa de tamaño mínimo para que el diseño final sea lo más compacto posible. El esquemático se añadirá a los anexos en formato A4.

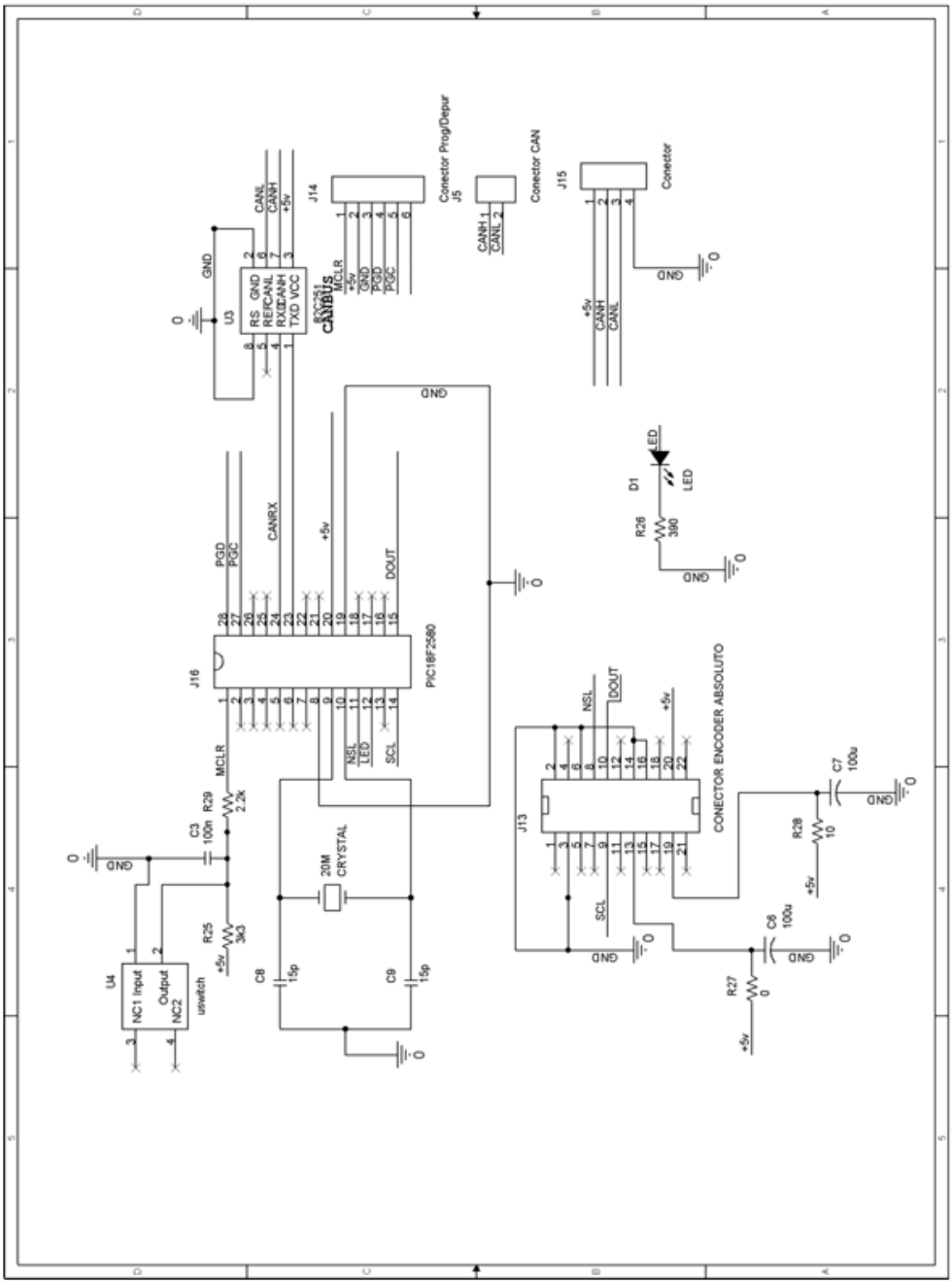


Figura 52: Esquemático Final.

Teniendo esto en cuenta se consiguió crear una placa de 56 mm de diámetro, cuyos

fotolitos se adjuntarán a los anexos de este proyecto, y la imagen principal se muestra en la *Figura* siguiente.

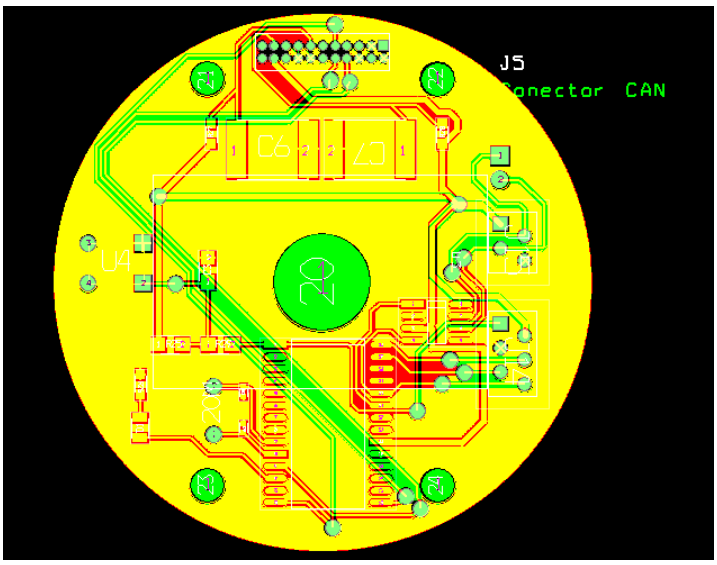


Figura 53: Global Layer.

4.3 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 de simplifica la tarea del programador con respecto a las instrucciones de ensamblador.

El programa se ha desarrollado de forma estructurada permitiendo la modificación del mismo con facilidad y basándose en la Application Note 945 (**AN945 "A CANopen Stack for PIC18 ECAN Microcontrollers**). La pila de Microchip AN945 es una librería de archivos C que implementan un Nodo CANopen de Entradas/Salidas y que junto con la documentación del protocolo CANopen (DS-301) dan soporte para implementar cualquier dispositivo que trabaje bajo dicho protocolo.

En este capítulo se hará una descripción detallada de la pila y de cómo utilizarla.

4.3.1 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). 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, es decir, un módulo de I/O genérico.

Todo el código suministrado con 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.

4.3.2 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 siguiente tabla se muestra una lista completa de los archivos fuente.

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.

Tabla 28: Ficheros Fuente de la Pila CANopen.

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".

El firmware está dividido en tres niveles, según se muestra en la siguiente figura. En el nivel más bajo se encuentra el driver ECAN que da soporte para la configuración Hardware del módulo CAN. El nivel de gestión de las comunicaciones es la primera interfaz entre el driver y el tratamiento individual de los distintos endpoint.

Entre la capa de aplicación y la de comunicaciones, también se encuentra el Diccionario de Objetos que está directamente conectado al endpoint del SDO.

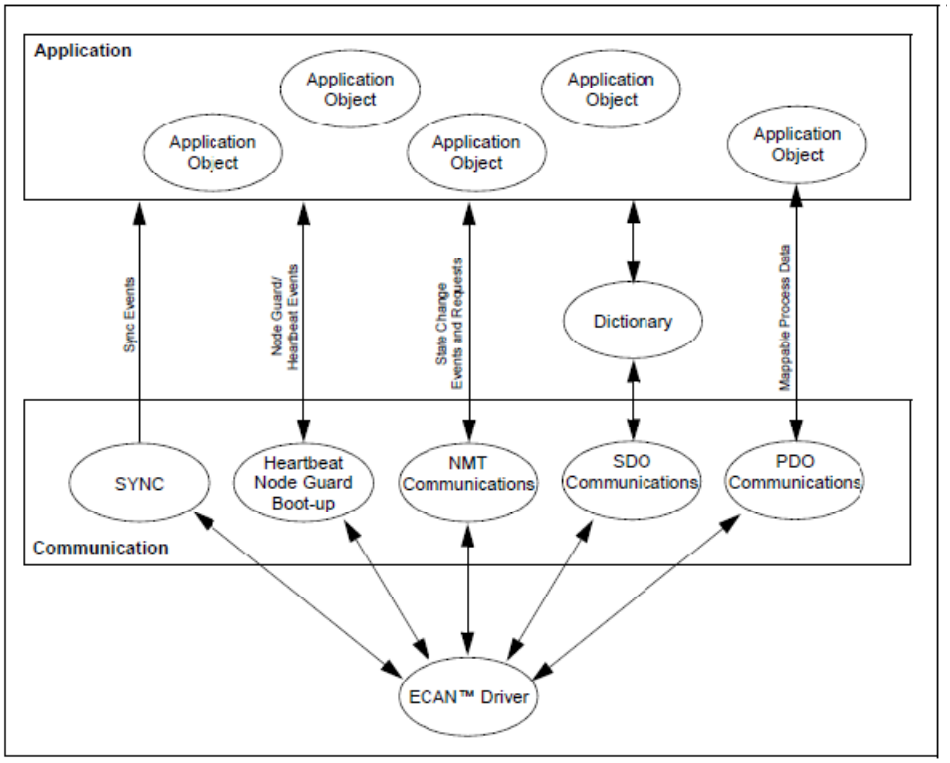


Figura 54: Modelo Básico del Firmware de la Pila CANopen.

4.3.3 Driver ECAN.

El driver proporciona una interfaz de configuración de todas las funcionalidades del Hardware ECAN, incluyendo también la configuración de los filtros que forman parte del protocolo CANopen. Está implementado en los archivos **CO_CANDRV.c** y **CO_CANDRV.h**.

4.3.4 Communications Management.

La gestión de comunicaciones está formada por todos los Objetos de comunicaciones. Se encarga de capturar cualquier evento producido en la capa física ó en la de aplicaciones, además de lanzar los sub-objetos de comunicaciones y funciones requeridos por dichos eventos. Esencialmente gestiona la apertura, cierre, transmisión y recepción de los endpoint. Tiene el conocimiento del estado de todos los endpoint y del estado global del Nodo. Esto le permite bloquear mensajes dirigidos a un endpoint si es necesario en función del estado local ó global del dispositivo.

Otra característica es que usa un método de tratamiento de los mensajes basado en un único byte, que le permite acelerar el proceso para descifrar la funcionalidad de los mensajes ya que es mucho más rápido tratar este byte que los 11 ó 29 bits del identificador CAN.

El Communications Management está implementado en los archivos **CO_COMM.c** y **CO_COMM.h**.

4.3.5 Endpoints.

El protocolo CANopen define varios endpoint que están directamente asociados con los tipos de Objetos. Un endpoint no es más que la configuración de un filtro para que las tramas sean aceptadas por el buffer de entrada al que está asociado el filtro.

Cuando un Nodo recibe una trama, esta es almacenada en un buffer intermedio (Message Assembly Buffer Identifier) y el identificador del mensaje es comparado con el valor almacenado en el filtro. Si estos coinciden el mensaje es almacenado en el buffer de entrada asociado a dicho filtro. Adicionalmente es posible configurar

también una máscara que indica que bits del identificador deben ser comparados con los correspondientes bits del filtro. En la tabla siguiente se muestra la lógica seguida por este proceso, y en la figura el esquema del circuito lógico.

Mask bit n	Filter bit n	Message Identifier bit n001	Accept or Reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Tabla 29: Tabla de Verdad Implementada en los Endpoint.

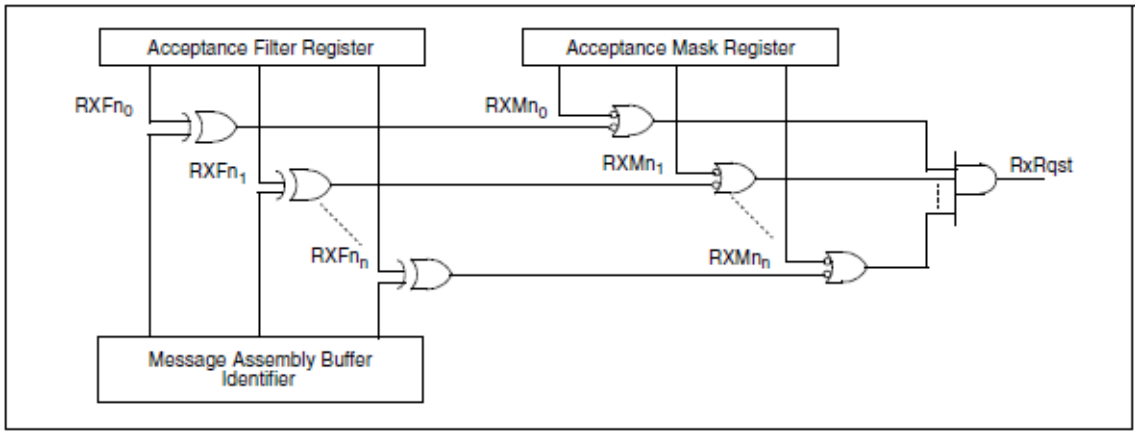


Figura 55: Circuito Lógico Implementado en los Endpoint.

Las tramas recibidas llevan asociadas un tipo de tratamiento según el endpoint que habilite su acceso al buffer de entrada. En está application note están implementados los cinco endpoint de los objetos numerados a continuación y pueden ser ampliados por el usuario.

- **Servidor del SDO**, se ha de implementar por defecto según el protocolo. El SDO está directamente lincado con el Diccionario de Objetos y permite decodificar, validar y si es válido, ejecutar los datos contenidos en el mensaje. El SDO está implementado en los archivos **CO_SDO1.c** y **CO_SDO1.h**.
- **Un máximo de 4 PDOs estáticos**, que deberán ser lincados (por el usuario) directamente con uno ó varios Objetos de Aplicación. Los datos son mapeados internamente con los Objetos de forma estática (compilados),

aunque existe la posibilidad de un mapeo dinámica (en tiempo de ejecución).

Esta application note soporta los 4 PDO que el protocolo especifica por defecto. Aun así, los servicios generales están implementados en los archivos **CO_PDO.c** y **CO_PDO.h**, mientras que el tratamiento individual de cada PDO ha de ser desarrollado por el usuario en los archivos **CO_PDOn.c** y **CO_PDOn.h** (donde n es el número de PDO y puede tomar el valor de 1 a 4) ya que estos últimos archivos están suministrados en forma de plantilla.

- **Consumidor de SYNC**, es un servicio que simplemente genera un evento que la aplicación utiliza para generar cualquier PDO que haya sido sincronizado. Está implementado en los archivos **CO_SYNC.c** y **CO_SYNC.h**.
- **Network Management Slave**, recibe todos los comandos para cambiar el estado del dispositivo y resetear las comunicaciones y/ó aplicaciones. En la figura siguiente se observa la Máquina de Estados CANopen y los distintos servicios necesarios para cambiar el estado de esta. El NMT está implementado en los archivos **CO_NMT.c** y **CO_NMT.h**.

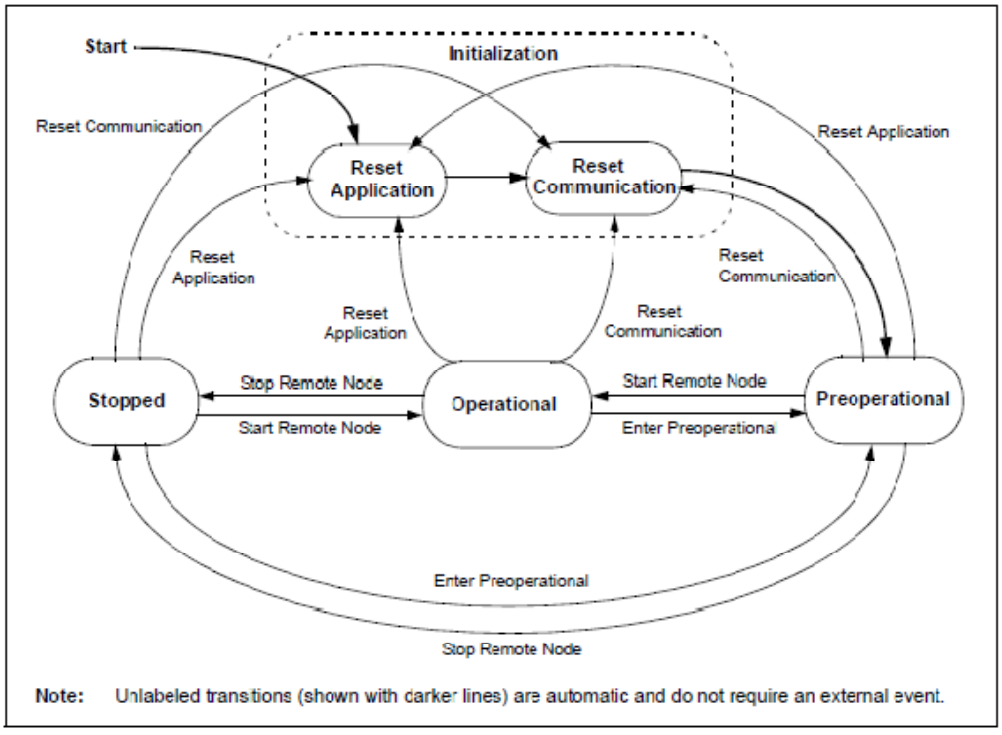


Figura 56: Máquina de Estados y Servicios Asociados.

- **Node Guard ó Heartbeat**, este endpoint está implementado según el

protocolo y aunque ambos conviven en el código, sólo uno de estos Watchdog puede estar activos al mismo tiempo. Funcionalmente están implementados en los archivos **CO_NMTE.c** y **CO_NMTE.h**.

4.3.6 Diccionario de Objetos.

Todo lo relacionado con el Diccionario de Objetos ya se ha comentado anteriormente en el apartado dedicado a este, por lo que solo queda decir que está implementado en los siguientes archivos:

- **CO_DICT.c**
- **CO_DICT.h**
- **CO_DICT.def**
- **CO_STD.def**
- **CO_MFTR.def**
- **CO_PDO.def**

4.3.7 Otros archivo fuentes.

Aunque en el modelo básico del firmware mostrado en la figura 54 no aparecen, la AN945 incluye los Objetos estándar que proporcionan información sobre el tipo de dispositivo, tales como estatus, nombre, fabricante, número de serie, etc. Toda esta información debe ser editada por el diseñador y se encuentra en los archivos **CO_DEV.c** y **CO_DEV.h**.

Por otro lado, también se incluyen una serie de archivos que son utilizados para definir los tipos de datos estándar de CANopen, los errores, funciones para copiar datos de memoria y herramientas de conversión, entre el formato de Microchip y el de CANopen, para los COB-ID.

- **CO_TOOLS.c**
- **CO_TOOLS.h**
- **CO_MEMIO.c**
- **CO_MEMIO.h**
- **CO_ABERR.h**

- **CO_TYPES.h**

En el nivel más alto de la pila tenemos los Objetos de Aplicación, que deben ser desarrollados e incluidos en el Diccionario de Objetos por el usuario ya que estos son específicos para cada tipo de dispositivo.

4.3.8 Uso de la pila CANopen.

En este aparatado se hará una descripción más detallada de los archivos que componen la pila CANopen, prestando especial atención a las funciones que forman estos y dando información de cómo deben ser modificados y tratados para el diseño de sistemas CANopen basados en microcontroladores PIC18 de Microchip®.

El archivo **main.c** contiene el **main** 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.

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.
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 Protocolo del Control de Errores, 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 EncoderInit (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 PRE-OPERATIONAL después de envía el mensaje de Boot-up.

```
// Enviar mensaje de boot-up.
NMTE_BOOT_SERVICE = 1;
mCANIsPutReady(0);
_CO_COMM_NMTE_TXEvent();
mCANSendMessage();

// Entrar en estado PRE-OPERATIONAL.
COMM_STATE_PREOP = 1;
```

La ejecución de los procesos asociados a la aplicación y comunicaciones se realizan a través del código incrustado dentro de un bucle cerrado *while* de la función **main**. En este apartado hay tres partes 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 EncoderProcessEvents (void)* y se encuentra en el archivo AppEncoder.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 char TimerIsOverflowEvent(void)* devuelve el valor TRUE.

```
// 1ms timer events
if (TimerIsOverflowEvent())
{
        // Process timer related events
        mCO_ProcessAllTimeEvents();
}
```

Los puntos 1 y 2 han de ser ejecutados al menos 2 veces antes de que se produzca el overflow del timer_0. En el caso de que esto no sea posible en 1 milisegundo, este periodo puede ser aumentado simplemente modificando la variable TIMER_PERIOD_MS que se encuentra en el archivo **Timer.c** y que representa el tiempo en milisegundos con el que se desborda dicho timer.

4.3.9 Funciones de configuración.

Una vez descrita la secuencia del programa principal, se definirá a continuación las distintas funciones que son llamadas por este en la etapa inicial de configuración del Nodo y que el usuario a de editar para el diseño del dispositivo.

void TimerInit(void)

Esta función configura el timer_0 como un reloj de 8 bits y calcula el valor de precarga de este para generar un evento periódicamente. El valor de la precarga depende de las variables:

- *TIMER_PERIOD_MS*, especifica el periodo del ciclo en milisegundos.
- *TIMER_FOSC*, representa la frecuencia de oscilación del reloj.
- *TIMER_PRESCALE*, selecciona un valor de escala del timer con respecto a los ciclos del reloj.

Estas variables, al igual que la función *void TimerInit(void)*, se encuentran en el archivo **timer.c**.

mSYNC_SetCOBID(SYNC_COB)

Esta función almacena el identificador del Objeto de sincronismo (*SYNC_COB*) en la variable interna *_uSYNC_COBID*, que posteriormente es usada para crear el endpoint para que el Nodo pueda implementar las opciones de un SYNC consumer. Se han de hacer dos llamadas a esta función en el main siguiendo el paso 2 de la descripción de configuración del dispositivo.

mCO_SetNodeID(NodeID)

Esta función almacena el número identificador del Nodo en la variable *_uCO_nodeID*, donde *NodeID* es un número comprendido en el rango [1… 127] y es único para cada dispositivo conectado al Bus.

mCO_SetBaud(bitrate)

La función *mCO_SetBaud* almacena el valor *bitrate*, que no es más que un valor

compredido en el rango [1… 8], en la variable *_uCO_baud*. Esta variable indica los valores que serán asignados a los registros de configuración de la tasa de transferencia (BRGCON1, BRGCON2 y BRGCON3) a través de las variables CAN_BITRATEn, que se encuentran en el archivo **CO_DEFS.def** y han de ser calculados por el usuario en función del reloj usado para diseñar el dispositivo y la tasa de transferencia deseada (ver apartado 4.4 "*Tasa de transferencia*").

mNMTE_SetHeartBeat(HeartBeat)

La función *SetHeartBeat* configura el **Heartbeat** del dispositivo mediante la variable *HeartBeat* que representa el tiempo en milisegundos con el que el Nodo emite una trama indicando que no se ha caído del Bus y el estado del mismo. En el caso de que se desee desactivar esta opción, se le asignara le valor 0x00 a la variable *HeartBeat*.

mNMTE_SetGuardTime(GuardTime)

mNMTE_SetLifeFactor(LifeFactor)

Estas dos funciones configuran el **Node Guard** del dispositivo. En el caso de que no se desee activar, se asignara el valor 0x00 a las dos variables (*GuardTime* y *LifeFactor*). La variable *GuardTime* representa el tiempo en milisegundos con el que el Maestro ha de consultar el estado del dispositivo y *LifeFactor* es un factor que multiplicado por *GuardTime* representa el tiempo máximo tras el cual se interpreta que:

- Ha ocurrido un error en el Nodo si no se ha producido la respuesta al chequeo del Maestro.
- Ó ha ocurrido un error en el Maestro si no se recibe el mensaje de chequeo.

Como ya se ha indicado anteriormente los dos sistemas de control de errores (**Node Guard** y **Heartbeat**) no pueden ser activados al mismo tiempo.

void EncoderInit (void)

EncoderInit es una función desarrollada por el usuario y que configura todos los parámetros del Microcontrolador relacionados con la lectura del encoder absoluto y el LED que indica el estado del Nodo. Estará localizada en un archivo que el diseñador creara para desarrollar todo lo relacionado con la aplicación (en este caso: AppEncoder.c).

En primer lugar, configura los pines 11 y 12 del micro como salidas. El pin 11 se utilizara como señal de habilitación de la lectura del encoder absoluto (**NSL**) y el 12 para el control del LED.

En segundo lugar, llamamos a la función *void OpenSPI(unsigned char sync_mode, unsigned char bus_mode, unsigned char smp_phase)* de las librerías del entorno de desarrollo de Microchip (MPLAB) y a la que debemos pasarle tres parámetros:

- *sync_mode*: configura el modo de lectura (Master ó Slave), el modo de operación de los pines y la frecuencia del reloj de lectura del puerto serie respecto del oscilador del sistema.
  En este caso se seleccionara el modo Master, lo que implica que esta variable sólo puede tomar el valore *SPI_FOSC_n* donde n es igual a 4, 16 ó 64 e indica el factor por el cual se divide la frecuencia de oscilación del reloj externo (clock_lectura = Fosc/n).
  El pin 14 (**SLC** = clock_lectura) es configurado como patilla de salida, mientras que el 15 (**SDI** "Serial Data In") se configura como entrada.
- *bus_mode*: esta variable configura la forma de onda del reloj del puerto serie a través de dos bits (**CKP** y **CKE**) para conseguir la forma de onda adecuada para realizar la lectura del encoder. CKP indica el estado de espera del reloj y CKE el momento de transición entre el estado de espera y el opuesto.
  Teniendo en cuenta la siguiente figura que representa el diagrama de tiempos de lectura del encoder dada por el fabricante tenemos que estas variables toman los siguientes valores:

  CKP = 1, estado de espera a nivel alto.

CKE = 1, el reloj mantiene estado de espera en el primer semi-ciclo tras habilitar lectura.

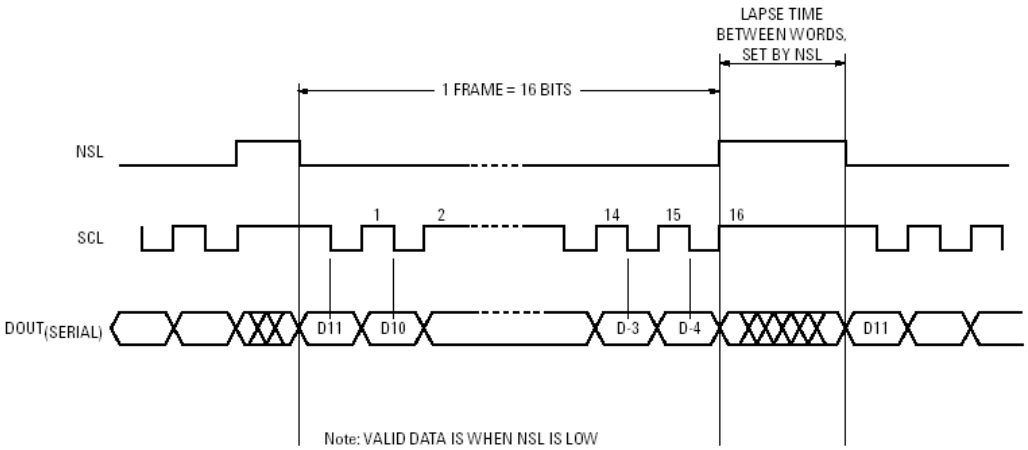


Figura 57: Diagrama de Tempos de Lectura del Encoder.

Para configurar las variables CKP Y CKE según lo expuesto *bus_mode* toma el valor *MODE_10*.

- *smp_phase*: indica el punto de muestreo de los datos con respecto al ciclo del reloj y es seleccionable entre *SMPMID* (muestreo a mitad del ciclo) y *SMPEND* (muestreo al final del ciclo). Para la correcta lectura del encoder se ha de seleccionar el muestreo a mitad de ciclo "*SMPMID*".

Tras configurar el puerto serie, el siguiente paso que realiza la función *EncoderInit* es configurar los modos de sincronismo de los TPDOs según el perfil de dispositivo de un encoder. 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 posición y el PDO_2 enviara el mensaje de posición de forma sincronizada con el SYNC Object.

Esta configuración se logra a través de las variables *uEncoderSyncSet_1*, *uEncoderSyncCount_2* y *uEncoderSyncSet_2*. *uEncoderSyncSet_1* configurara el TPDO_1 y las otras dos variables el TPDO_2 adoptando los siguientes valores.

uEncoderSyncSet_1 = 254, configura el TPDO_1 para transmisión asíncrona.

uEncoderSyncCount_2 = uEncoderSyncSet_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 posición 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. Este último paso se tratara con más detalle en el capítulo "*Implementación de un PDO*".

4.3.10 Desarrollo de la aplicación de lectura del encoder.

La aplicación de lectura del encoder se desarrolla en el archivo **AppEncoder.c** y además de estar compuesta por la función de inicialización *void EncoderInit (void)* que se ha explicado anteriormente, está compuesta por otras tres funciones que son:

- *void EncoderProcessEvents (void)*, es la función principal de la lectura del encoder y a través de la cual se gestiona todo el proceso.
- *void GrayToBinary (UNSIGNED16 *pIn, UNSIGNED16 *pOut)*, convierte la lectura del encoder de gray a binario.
- *void CO_COMMSyncEvent(void)*, sincroniza los TPDOs de acuerdo con el modo de sincronismo para el que se ha configurado coda uno de los TPDOs. Esta función se explicara en el apartado 4.3.5 "*Implementación de un PDO*".

void EncoderProcessEvents (void)

La primera parte de esta función consta de las declaraciones necesarias para desarrollar y detectar cambio en la lectura del encoder. Para ello se declaran 4 variables que cumplen con las siguientes funciones:

- *unsigned char length = 2*, indica en bytes la resolución de la lectura que será de 16 bits (2 bytes).
- *UNSIGNED16 gray*, almacena la lectura del encoder proporcionada por el puerto serie en código gray.

- *UNSIGNED16 uEncoderPosition*, almacena la lectura del encoder una vez que es convertido a código binario.
- *static UNSIGNED16 uEncoderPositionOld*, guarda la lectura anterior del encoder para detectar cambios con respecto a la actual (*uEncoderPosition*).

El siguiente paso es realizar la lectura del encoder. Esto se realiza a través del SPI aprovechado las librerías suministradas por Microchip® y en concreto la función *void getsSPI(unsigned char *rdptr, unsigned char length)*, donde *rdptr* es la dirección de la variable en la que se guardaran los datos leídos (*gray*) y *length* indica el tamaño de estos. Antes de realizar la lectura del encoder es necesario habilitar el dispositivo para dicha lectura realizando un reset en la patilla NSL.

```
// Habilitar lectura del encoder.
NSL = 0;
// Leer posición del encoder.
getsSPI((unsigned char *)&gray.bytes.B1, length );
// Deshabilitar lectura del encoder.
NSL = 1;
```

Tras la lectura del encoder se realiza la conversión de gray a binario llamando a la función *void GrayToBinary (UNSIGNED16 *pIn, UNSIGNED16 *pOut)*. *pIn* es la dirección de la variable *gray* y *pOut* la dirección de *uEncoderPosition*. Esta función devuelve en *uEncoderPosition* el valor de posición en binario.

Tras realizar la conversión 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 *uEncoderPositionOld*.

```
// Indicar si se produce un cambio.
if (uEncoderPositionOld.word != uEncoderPosition.word)
{
// Bits de estado.
        uEncoderState.bits.b1 = 1;   // TPDO_1
        uEncoderState.bits.b4 = 1;   // TPDO_2
```

```
        // Almacenar posición en buffers de TPDO.
*(UNSIGNED16 *)uLocalTPDO2Buffer = *(UNSIGNED16
*)uLocalTPDO1Buffer = *((UNSIGNED16 *)(&uEncoderPosition.word));
// Guardar nueva posición.
        uEncoderPositionOld.word = uEncoderPosition.word;
}
```

La variable *uEncoderState* esta 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 encoder a variado.
- b2, cuando la variable *uEncoderSyncSet_1 = 0* indica al TPDO_1 que la lectura esta 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 encoder a variado.
- b5, cuando la variable *uEncoderSyncSet_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 encoder, el siguiente paso es chequear el modo de sincronismo y en función de este enviar la lectura. Si estamos en modo asíncrono (*uEncoderSyncSet* igual a 254 ó 255) envía la lectura inmediatamente y en el modo síncrono (*uEncoderSyncSet* en el rango [1…253]) espera la recepción de tantos mensajes de sincronismo como indique el valor de *uEncoderSyncSet*.

Existe la posibilidad de que se le asigne a la variable *uEncoderSyncSet* 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 (uEncoderState.bits.b1)
{
```

```
            switch (uEncoderSyncSet_1)
            {
                  case 0:          // transmisión acíclica síncrona.
                        // Activar flag de transmisión síncrona.
                        uEncoderState.bits.b2 = 1;
                        break;

                  case 254:                     // Transmisión asíncrona.
                  case 255:
                        // Activar flag de transmisión asíncrona.
                        uEncoderState.bits.b0 = 1;
                        break;
            }
      }
```

Cuando el modo de transmisión seleccionado es síncrono, la lectura de posición se actualiza en cada ciclo de ejecución del programa de tal forma que siempre se envíe el dato actualizado y el control de envío del dato se realiza a través de la función *void CO_COMMSyncEvent(void)* que se explicará en el apartado "Implementación de un PDO".

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

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

En este trozo de código no aparece el control de envío del TPDO_2 ya que la única diferencia con el TPDO_1 es que cambian las variables.

void GrayToBinary (UNSIGNED16 *pIn, UNSIGNED16 *pOut)

El encoder suministra la lectura de posición en código gray, por lo que es necesario transformar esta a código binario ya que es mucho más fácil de tratar y simplifica las operaciones de control del microprocesador que se encarga del control del robot.

Si tratamos esta función como una caja negra no es más que la implementación de una OR exclusiva de 16 bits en la que el MSB de salida es igual al de entrada y los sucesivos se calculan en función del calculado anteriormente y la correspondiente posición del bit de entrada.

```
void GrayToBinary (UNSIGNED16 *pIn, UNSIGNED16 *pOut)
{
	pOut->bytes.B1.bits.b7 = pIn->bytes.B1.bits.b7;
	pOut->bytes.B1.bits.b6 = pIn->bytes.B1.bits.b6 ^ pOut->bytes.B1.bits.b7;
	pOut->bytes.B1.bits.b5 = pIn->bytes.B1.bits.b5 ^ pOut->bytes.B1.bits.b6;
	pOut->bytes.B1.bits.b4 = pIn->bytes.B1.bits.b4 ^ pOut->bytes.B1.bits.b5;
	pOut->bytes.B1.bits.b3 = pIn->bytes.B1.bits.b3 ^ pOut->bytes.B1.bits.b4;
	pOut->bytes.B1.bits.b2 = pIn->bytes.B1.bits.b2 ^ pOut->bytes.B1.bits.b3;
	pOut->bytes.B1.bits.b1 = pIn->bytes.B1.bits.b1 ^ pOut->bytes.B1.bits.b2;
	pOut->bytes.B1.bits.b0 = pIn->bytes.B1.bits.b0 ^ pOut->bytes.B1.bits.b1;
	pOut->bytes.B0.bits.b7 = pIn->bytes.B0.bits.b7 ^ pOut->bytes.B1.bits.b0;
```

```
        pOut->bytes.B0.bits.b6 = pIn->bytes.B0.bits.b6 ^ pOut-
>bytes.B0.bits.b7;
        pOut->bytes.B0.bits.b5 = pIn->bytes.B0.bits.b5 ^ pOut-
>bytes.B0.bits.b6;
        pOut->bytes.B0.bits.b4 = pIn->bytes.B0.bits.b4 ^ pOut-
>bytes.B0.bits.b5;
        pOut->bytes.B0.bits.b3 = pIn->bytes.B0.bits.b3 ^ pOut-
>bytes.B0.bits.b4;
        pOut->bytes.B0.bits.b2 = pIn->bytes.B0.bits.b2 ^ pOut-
>bytes.B0.bits.b3;
        pOut->bytes.B0.bits.b1 = pIn->bytes.B0.bits.b1 ^ pOut-
>bytes.B0.bits.b2;
        pOut->bytes.B0.bits.b0 = pIn->bytes.B0.bits.b0 ^ pOut-
>bytes.B0.bits.b1;
}
```

La función devuelve en código binario la lectura del encoder. Esta es almacenada en la dirección de la variable *uEncoderPosition*.

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.

```
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;
        }
```

LED_PERIOD_MS indica la frecuencia con la que el LED parpadea y *CO_TICK_PERIOD* indica con que frecuencia en milisegundos es llamada la función.

4.3.11 Implementación de un PDO.

Esta es una de las partes más importantes en el desarrollo de un Nodo CANopen basada en la pila de Microchip® ya que es la que permite enviar los datos adquiridos por el dispositivo en Tiempo Real. A diferencia del SDO por defecto que está implementado en su totalidad (exceptuando pequeños matices que no son de carácter obligatorio), el PDO requiere ser implementado ya que la pila CANopen sólo proporciona un conjunto de archivos que hacen las veces de plantilla para desarrollar hasta un máximo de 4 PDOs.

En este apartado se describirán los pasos a seguir que son necesarios para implementar un PDO.

1. El primer paso es crear los COB-IDs de los TPDOs en el archivo **AppEncoder.c**, dentro de la función *void EncoderInit (void)* teniendo en cuenta que el valor de los identificadores siguen las formulas siguientes:
   TPDO_1 = 0x180 + NodeID
   TPDO_2 = 0x280 + NodeID
   En adelante se describirá solamente la implementación del PDO_1 ya que el proceso no varía a la hora de implementar otros PDOs.
   Para crear el COB-ID del TPDO_1 se hace una llamada a la función *mTOOLS_CO2MCHP(CO_COB)*, donde *CO_COB* representa el identificador y ha de ser una variable de 32 bits (UNSIGNED32) que es guardada en el registro *_uCOB_ID_in* en formato de Microchip.

```
        // Convertir el identificador a formato Microchip.
```

```
mTOOLS_CO2MCHP(mCOMM_GetNodeID().byte + 0xC0000180L);
```

mCOMM_GetNodeID().byte devuelve el valor de NodeID y la '*C*' en el byte más significativo indica que el TPDO en principio esta cerrado, por lo que deberá abrirse antes de ser usado.

2. Una vez creado el COB-ID el siguiente paso es almacenarlo en la variable *uTPDOComm1* el valor de este. Esto se consigue invocando a la función *mTPDOSetCOB(PDOn, tpdoCOB)*, donde *PDOn*, que es un valor comprendido en el rango [1… 4], representa el número de PDO y *tpdoCOB* es el identificador creado previamente.

```
// Almacenar el COB-ID del TPDO1
mTPDOSetCOB(1, mTOOLS_GetCOBID());
```

3. Se ha de asociar el buffer de transmisión que previamente ha sido creado é inicializado en la fase de configuración. Para el TPDO_1 este buffer es el uLocalTPDO1Buffer.

```
// Asociar buffer del PDO1
mTPDOSetTxPtr(1, (unsigned char *)(&uLocalTPDO1Buffer[0]));
```

El buffer ha de estar declarado como una variable externa en el archivo de cabecera de nuestra aplicación **AppEncoder.h** ya que será utilizado por otras funciones que no se encuentran en el archivo que contiene la aplicación.

```
extern unsigned char uLocalTPDO1Buffer[8];
```

4. Editar la función *void CO_COMMSyncEvent(void)* para el tratamiento de sincronismos.

```
void CO_COMMSyncEvent(void)
{
// TPDO1
    // Procesar sólo en modo síncrono
    if ((uEncoderSyncSet_1 == 0) && (uEncoderState.bits.b2))
    {
```

```
                // Activar flag para envío de mensaje.
                uEncoderState.bits.b2 = 0;
                uEncoderState.bits.b0 = 1;
        }
        else
        if ((uEncoderSyncSet_1 >= 1) && (uEncoderSyncSet_1 <= 240))
        {
                // Cuenta de mensajes de sincronismo para el modo síncrono.
                uEncoderSyncCount_1--;
                // Habilitar envío de mensaje.
                if (uEncoderSyncCount_1 == 0)
                {
                        // Reset del contador de mensajes de sincronismo.
                        uEncoderSyncCount_1 = uEncoderSyncSet_1;
                        // Activar flag de transmisión.
                        uEncoderState.bits.b0 = 1;
                }
        }
}
```

5. Definir el número de PDOs que se desea implementar en el archivo **CO_DEFS.def** editando la variable *CO_NUM_OF_PDO*. Esta variable puede tomar un valor comprendido entre 1 y 4 que es el máximo número de PDOs posibles.

6. Crear la función *void CO_COMM_TPDO1_COBIDAccessEvent(void)* en el archivo **CO_PDOn.c**, donde '**n**' representa el numero de PDO. Esta función proporcionan una interfaz con el Diccionario de Objetos que permite consultar el COB-ID del PDO, además de abrirlo y cerrarlo en función de las necesidades del sistema de control.

   Existe un ejemplo de esta función en la pila CANopen por lo que no se incluirá en este documento.

7. Crear en el mismo fichero que en el paso anterior la función *void CO_COMM_TPDO1_TypeAccessEvent*, la cual permite cambiar el modo de sincronismo del PDO modificando la variable *uEncoderSyncSet_n*, siendo '*n*' el número de PDO.
   Al igual que la función *void CO_COMM_TPDO1_COBIDAccessEvent(void)*, existe un ejemplo en la pila.

8. Incluir las funciones *void CO_COMM_TPDO1_COBIDAccessEvent(void)* y *void CO_COMM_TPDO1_TypeAccessEvent* en el Diccionario de Objetos en el index 1800h. El sub-index 00h contendrá el máximo valor de índices soportados por este Objeto, el 01h y 02h las direcciones de dichas función.
   Para ello el primer paso es crear una entrada en el Diccionario de Objetos para el PDO que se este implementando. Esto se realiza agregando el siguiente trozo de código en el archivo **CO_dict.c**.

```
rom DICT_OBJECT_TEMPLATE _db_pdo1_tx_comm[] =
    {DICTIONARY_PDO1_TX_COMM};
```

   Seguidamente se mapeará las direcciones de las función en el Objeto introduciendo el siguiente código en el archivo **CO_PDO.def** con las propiedades *RW* (lectura/escritura) y *FUNC* (tratamiento de función). El tamaño de los objetos será de 4 bytes para la primera función y 1 byte para la segunda.

```
#define DICTIONARY_PDO1_TX_COMM
    \\
{0x1800,0x00,CONST,1,{(rom unsigned char *)&rMaxIndex2}},
        \\
{0x1800,0x01,RW|FUNC,4,{(rom unsigned char
*)&CO_COMM_TPDO1_COBIDAccessEvent}}, \\
{0x1800,0x02,RW | FUNC,1,{(rom unsigned char
*)&CO_COMM_TPDO1_TypeAccessEvent}}
```

9. El último paso es incluir los archivos de cabecera que contengan las declaraciones de las distintas variables y funciones en los archivos donde sean necesarios para que el programa pueda compilar sin problemas.

4.4 Tasa de transferencia.

El cálculo de la tasa de trasferencia nominal representa el número de bits transmitidos por segundos y está definido por el protocolo CANopen según la siguiente tabla, siendo la tasa máxima permitida de 1 Mbps.

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)

Tabla 30: Tasas de Transferencia Recomendadas por el Protocolo CANopen.

Esta tabla está calculada basándose en un reloj de 16 MHz y los valores especificados son orientativos.

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 adecuadamente.

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

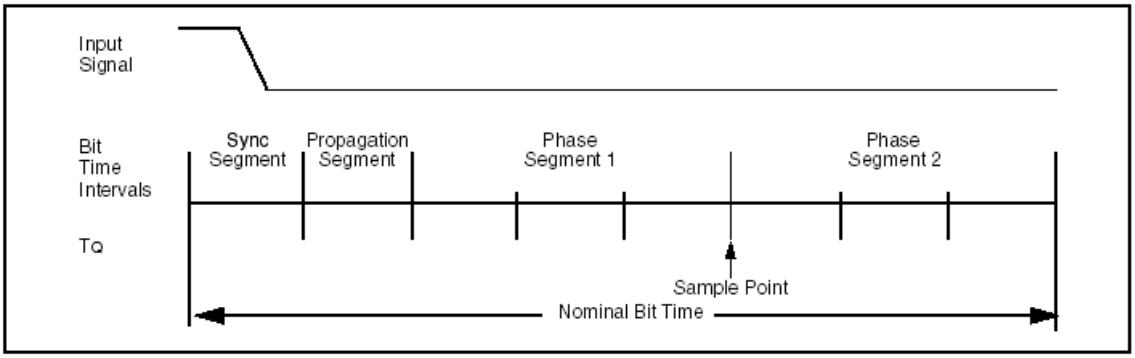


Figura 58: Partición del Tiempo de Bit Nominal.

El tiempo de bit nominal esta dividido en 4 segmentos de tiempo no solapados, y que a su vez, están subdivididos en unidades de tiempos denominados Time Quanta (T_Q). Por definición, el tiempo de bit nominal ha de ser programado con un mínimo de 8 T_Q 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 T_Q.
- **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 T_Q.
- **Phase Buffer Segment 1 (Phase_Seg1)**, determina el punto de muestreo de la señal del Bus y su duración es de 1 a 8 T_Q.
- **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 T_Q, su longitud mínima debe ser de 2 T_Q debido a tiempos de procesado de los datos.

El **punto de muestreo** es el punto en el cual el valor del bit es leído y debe estar aproximadamente al 80% del tiempo de bit nominal, pero sin sobrepasar este porcentaje.

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 T_Q 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 T_Q.

Las formulas usadas para realizar los cálculos son las siguientes:

- T_Q (μs) = (2 * (BRP+1))/F_{OSC} (MHz)
- Nominal Bit Time (μs) = T_Q * (Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2)
- T_{BIT} (MHz) = 1/Nominal Bit Time

Todos los datos calculados en este apartado se asignaran a los registros BRGCON1, BRGCON2 y BRGCON3. La frecuencia del reloj externo es dividida por el factor denominado BRP.

Teniendo en cuenta que la F_{OSC} es de 20 MHz, que el valor de BRG seleccionado es igual a 0 y el valor de SJW igual a 2 T_Q, se han obtenido los siguientes valores para una tasa de transferencia de **500 MHz**.

T_Q (μs) = 0.1 μs

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 μs

T_{BIT} (MHz) = 500 MHz

Estos resultados implican los siguientes valores:

BRGCON1 = 40h BRGCON2 = B7h BRGCON3 = 83h

Capítulo 5

PUESTA EN MARCHA.

5. PUESTA EN MARCHA.

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 NTM 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.

COB-ID	COMANDO	DATO
0x00	0x01	NodeID

Tabla 31: Mensaje Start Nodo.

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 PRE-OPERATIONAL 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 el caso de un encoder absoluto consisten 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.

Aunque pueda parecer una función innecesaria el hecho de abrir y cerrar un TPDO, esta es una función muy útil ya que al activar cualquier sistema de control, el primer paso es configurarlo y para esto es suficiente con disponer de un TPDO síncrono. Mientas que una vez que el sistema este trabajando, la opción más útil es el TPDO asíncrono siempre y cuando no sature el Bus.

El encoder tiene implementados dos TPDOs,

- el TPDO_1 que se configura a través del Objeto de Diccionario 1800h y cuyo COB-ID es el 180h + NodeID. Inicialmente está configurado para transmitir en modo asíncrono.
- y el TPDO_2 al que le corresponde el Objeto de configuración 1801h y su COB-ID es el 280h + NodeID. Inicialmente configurado para transmisión síncrona.

5.1 Cambiar modo de sincronismo.

El modo de sincronismo depende directamente de la variable *uEncoderSyncSet* 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 *uEncoderSyncSet*.
- '**254**' y '**255**', modo asíncrono. Envía un TPDO cada vez que se detecta un cambio de posición.

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

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 32: Cambiar Modo SYNC TPDO_1.

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

Tabla 33: Cambiar Modo SYNC TPDO_2.

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 *uEncoderSyncSet*.

D2, D3, D4, siempre 0x00.

5.2 Abrir TPDO.

En las tablas siguientes están representados los datos para abrir los TPDOs.

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 34: Open TPDO_1.

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

Tabla 35: Open TPDO_2.

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.

5.3 Cerrar TPDO.

Esta herramienta de configuración es prácticamente idéntica a la de abrir un TPDO con la excepción de que cambia el comando abrir TPDO por el de cerrar TPDO, según se puede observar en las siguientes tablas.

COB-ID	COMANDO	L_INDEX	H_INDEX	SUB-INDEX	D1	D2	D3	D4
0x600+NodeID	0x23	0x00	0x18	0x01	0x80+NodeID	0x01	0x00	0xC0

Tabla 36: cerrar TPDO_1.

COB-ID	COMANDO	L_INDEX	H_INDEX	SUB-INDEX	D1	D2	D3	D4
0x600+NodeID	0x23	0x01	0x18	0x01	0x80+NodeID	0x01	0x00	0xC0

Tabla 37: Cerrar TPDO_2.

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 cerrar TPDO.

5.4 Estructura de los datos transmitidos por el encoder.

Los datos de posición 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 e puede comprobar en la siguiente tabla.

COB-ID	L_POSICION	H_POSICION	
0x180 + NodeID	0xXX	0xXX	TPDO_1
0x280 + NodeID	0xXX	0xXX	TPDO_2

Tabla 38: Estructura de Datos del TPDO.

Donde,

COB-ID, representa al identificador de TPDO.

L_POSICION, 'XX' es el valor del byte bajo de la posición transmitida.

H_POSICION, 'XX' es el valor del byte alto de la posición transmitida.

5.5 Integración del encoder absoluto en un péndulo invertido.

Después de realizar el montaje de la placa y desarrollar el hardware explicado en los apartados anteriores, el siguiente paso es alimentar el circuito a **+5 V** y programar el microcontrolador PIC18 con la herramienta de programación MPLAB ICD2 de Microchip®. Para este fin se han diseñado dos cables de conexión.

Uno de ellos para la programación, que se ha fabricado según la descripción de la documentación del ICD2. Y otro, que además de permitir la alimentación de la placa incorpora un conector DB-9 para la conexión al Bus CAN (ya que este es el

conector estandarizado por el protocolo CANopen) y dos conectores cocodrilos que facilitan la conexión de la alimentación con cualquier fuente de tensión. En la siguiente tabla se puede observar el pinout del conector DB-9 y la figura muestra una imagen de la placa con los cables de conexión.

PINOUT DB-9	SEÑAL
2	CAN_L
3	CAN_GND
6	GND
7	CAN_H

Tabla 39: Pinout DB-9.

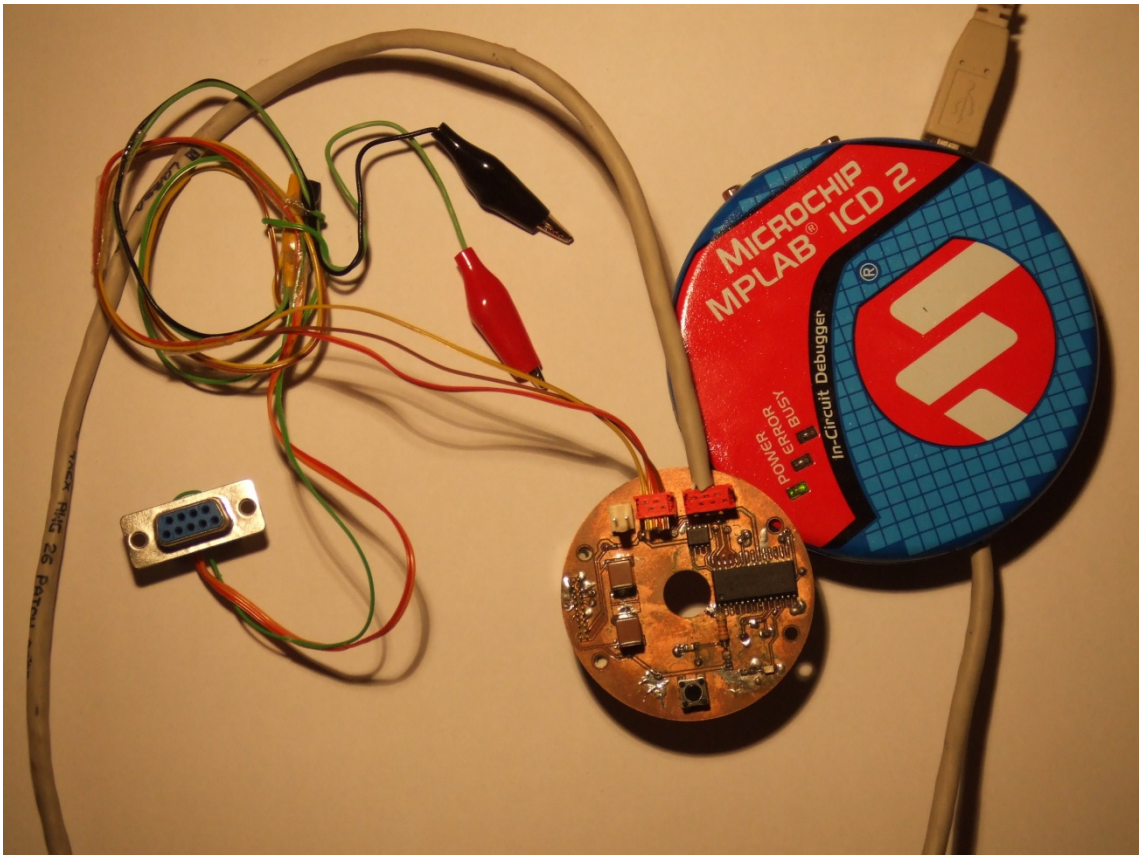


Figura 59: Cables de conexión.

El último paso es realizar la integración del dispositivo en un sistema real que permitirá comprobar si este cumple con todas las expectativas y especificaciones requeridas.

Para ello, y debido a que PASIBOT ha sido desmontado para realizar cambios estructurales y no ha estado accesible en el momento de las pruebas del encoder absoluto, se ha decidido integrar el encoder en el péndulo invertido diseñado para el robot **RH-2** (nueva versión del RH-1 que esta siendo diseñado por el Departamento de Ingeniería de Sistemas y Automática de la UCIIIM).

El péndulo invertido es una estructura creada para realizar las funciones de tobillo del nuevo RH que incorpora dos encoders absolutos con características muy similares al diseñado en este proyecto. Uno para controlar la inclinación frontal y el otro para la inclinación lateral. El montaje del encoder se ha realizado sobre uno de sus ejes de giro, con el objetivo de realizar un control simple que permita verificar el correcto funcionamiento del Nodo. En la siguiente figura se puede observar dicho montaje.

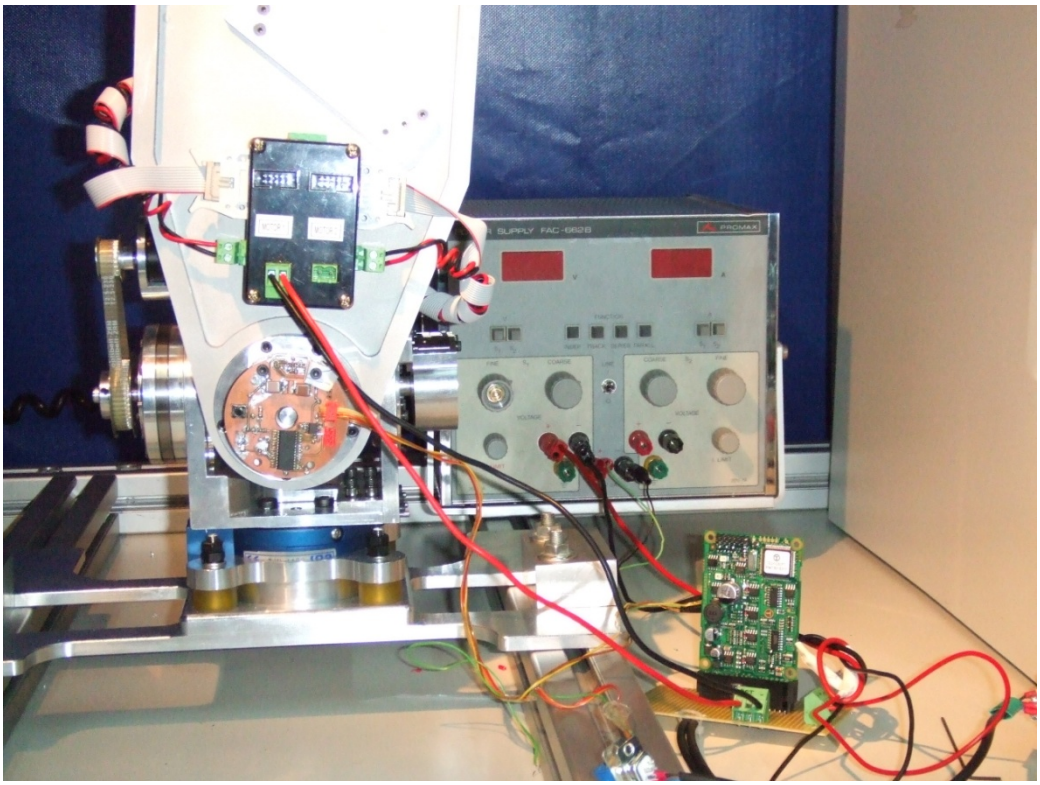


Figura 60: Montaje del Encoder en el Péndulo.

El sistema completo estaba formado, además de por el péndulo invertido y el encoder, por un driver de control del motor, un micro pc y una tarjeta de red CAN para comunicar el micro pc con los distintos dispositivos CAN. El programa de

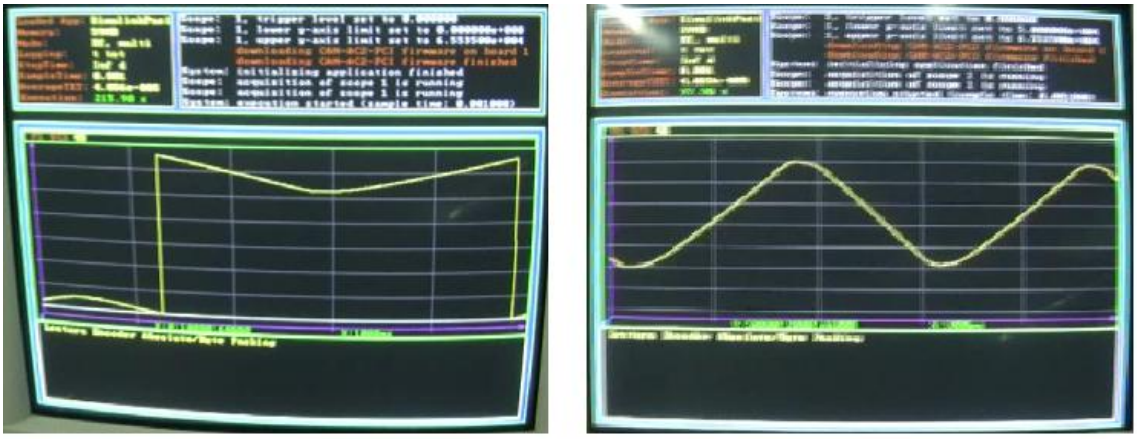
control implementado en el micro pc se realizo en MATLAB/SIMULINK y consistió en mover el péndulo invertido de un lado a otro de tal forma que este movimiento estuviera delimitado por dos posiciones angulares. En la siguiente figura puede observarse el sistema completo.

Figura 61: Sistema de Control Completo.

Tras realizar las primeras pruebas, el primer contratiempo que surgió fue que en la lectura del encoder se producían grandes escalones que no se correspondían con el desplazamiento real de la estructura. Esto se debe a que a pesar de que el fabricante asegura que el montaje del encoder no es crítico para obtener una buena lectura, la realidad es que el cabezal debe estar perfectamente alineado con el disco y se soluciono ajustando el cabezal de lectura con el juego que permiten los tornillos de sujeción de este.

Otro contratiempo que surgió fue que, en ciertas posiciones que no variaban, el encoder introducía pequeña interferencia en la lectura. Esto se soluciono tras limpiar el disco ya que este presentaba pequeñas manchas.

Una vez solucionados todos los problemas, se consiguió hacer el control deseado obteniendo unos resultados óptimos. En la siguiente figura se puede observar dos gráficas que se corresponden con distintas lecturas de encoder absoluto en tiempo de ejecución. La gráfica a) se representa un recorrido del péndulo mayor que la de la gráfica b).



a) Control1. b) Control2.

Figura 62: Gráficas de Control.

Capítulo 6

RESULTADOS Y CONCLUSIONES.

6. RESULTADOS Y CONCLUSIONES.

En este capítulo se hará un análisis crítico de los resultados finales y posteriormente se presentarán las conclusiones obtenidas.

Una de las principales preocupaciones es que el sistema sea capaz de comportarse como un sistema de **Tiempo Real**, consiguiendo cerrar el lazo de control en los tiempos predefinidos. Teniendo esto en cuenta se comparará la cadencia con la que el Nodo diseñado es capaz de introducir las tramas PDOs en el Bus con respecto a lo esperado teóricamente. Y finalmente se aportaran las conclusiones pertinentes.

6.1 Resultados.

Los sistemas de comunicaciones basados en CAN tienen muy definido el acceso al Bus por lo que se garantiza que este pueda estar en todo momento recibiendo tramas sin que una colisión suponga perdida de tiempo en la transmisión de las tramas como se puede observar en la siguiente figura que representa el método de arbitración de acceso al Bus.

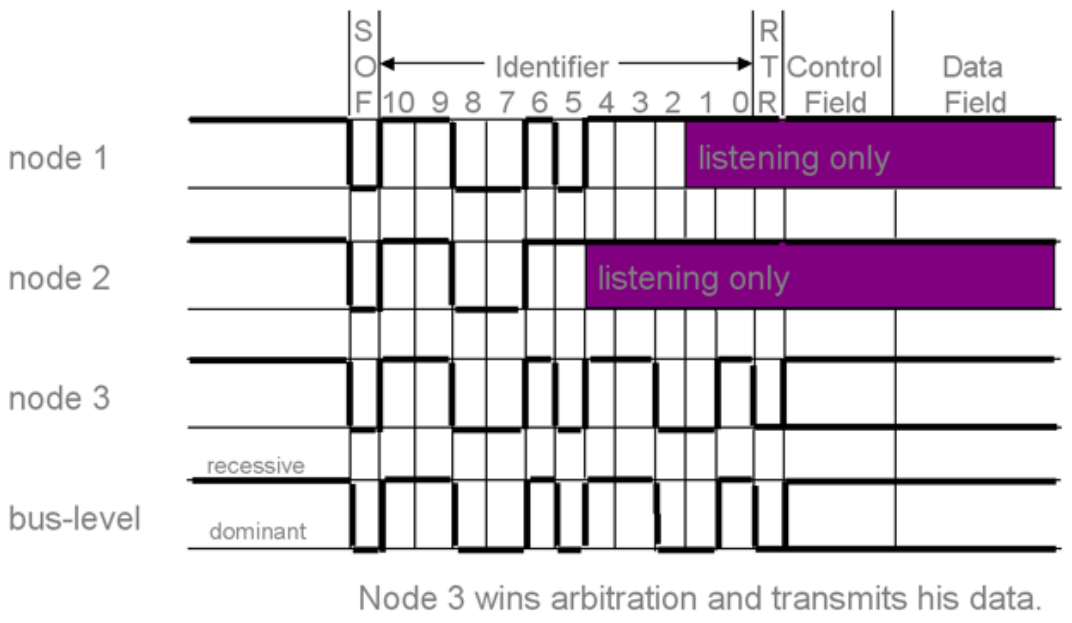


Figura 63: Método de Arbitración de Acceso al Bus.

Una vez que un Nodo está transmitiendo no puede ser desalojado del Bus hasta que termine la transmisión y cuando el bus queda libre todos los Nodos que deseen

enviar datos comenzaran la transmisión simultáneamente ganando el acceso el Nodo cuyo identificador de la trama a trasmitir sea el mas prioritario. Esto es posible debido a que los Nodos monitorizan el Bus a la vez que transmiten y si detectan un bit dominante mientras están transmitiendo uno recesivo, automáticamente abortan la transmisión, cediendo el acceso al identificador con mayor preferencia.

Teniendo en cuenta esto se puede realizar un sencillo cálculo que nos da ha conocer la máxima cadencia con la que un Nodo es capaz de introducir tramas en el Bus suponiendo que el identificador es el más prioritario y en función del tamaño de los datos enviados.

En el caso del encoder los datos de posición son transmitidos en un TPDO en formato estándar que implica el envío de 44 bits, donde se incluyen:

- El bit de start.
- El campo de arbitración (identificador de 11 bit + RTR).
- El campo de control de 6 bits.
- El campo de chequeo de 16 bits.
- El campo acknowledge de 2 bits.
- El campo de final de trama de 7 bits.

Más 2 bytes de dato (16 bits), 3 bits del espacio interframe y un máximo de 12 bits stuffing (calculado con la formula **s_max = (34 + 8 dlc -1) / 4**). Lo que implica que en cada trama pueden ser enviados hasta 75 bits con una tasa de transferencia de **500 MHz.**

Por lo que teóricamente la cadencia máxima con la que el Nodo puede introducir tramas en el Bus es de:

Cadencia $_{máx}$ = 75 bits / 500 MHz = 150 μs

Lo que equivale a una frecuencia de transmisión máxima de aproximadamente **6.7×10^3 tramas por segundo**.

Para que esta cadencia máxima se cumpla también es necesario que el programa que corre en el microcontrolador PIC18 tenga un tiempo de ciclo de ejecución menor de 150 μs.

Uno de los puntos más críticos en este aspecto es la lectura de los 16 bits de la posición del encoder por el SPI. Teniendo en cuenta que la frecuencia de oscilación del reloj de lectura ha de ser menor de 16 MHz, según las características técnicas del Agilent AEAS-7000, y es igual a ***clock_lectura = Fosc/n***. Se ha seleccionado 'n' igual a 4, con lo que obtenemos un reloj de lectura del SPI de **5 MHz** ya que la '*Fosc*' es igual a 20 MHz. Esto permite obtener la lectura del encoder en **3.2 μs**. Por lo que el micro dispone de aproximadamente 146 μs para correr el resto del programa.

En este tiempo el microcontrolador puede ejecutar hasta 730 instrucciones simples y para comprobar si estas son suficientes para ejecutar un ciclo de programa completo se ha decidido programar una de las patillas libres del micro para que alterne su estado entre nivel alto y bajo en cada ciclo de programa, obteniéndose la siguiente gráfica.

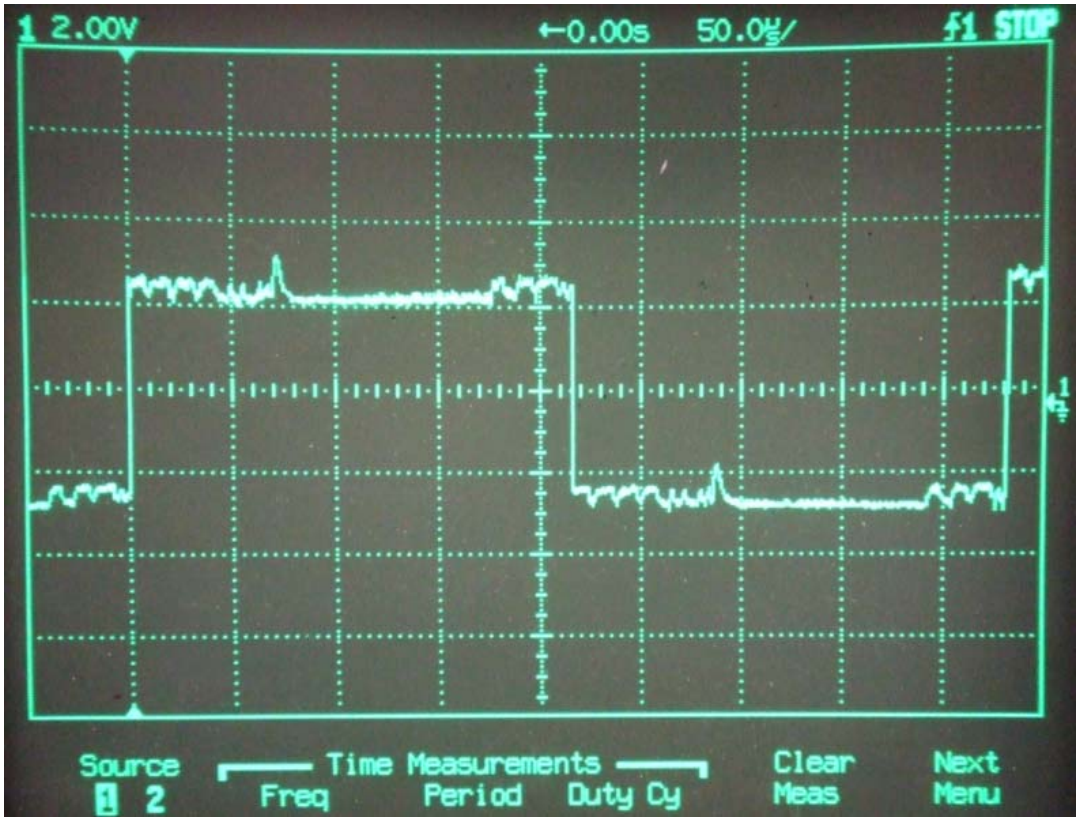


Figura 64: Frecuencia Ciclo de Programa.

En esta gráfica se puede comprobar que el ciclo de programa dura aproximadamente **215 μs**, por lo que la cadencia máxima con la que el nodo puede introducir tramas en el Bus se reduce teóricamente a **4.7×10^3 tramas por segundo**.

Para contrastar los datos teóricos y asegurar que estos se cumplen en la práctica se han realizado ciertas modificaciones en la programación del encoder, de forma que este transmita los datos de posición en cada ciclo de programa. Además se ha diseñado un Nodo capaz de recibir todos los mensajes transmitidos por el Bus, contarlos y enviar este dato al HyperTerminal a través del puerto USART. Los resultados obtenidos se muestran en las siguientes figuras.

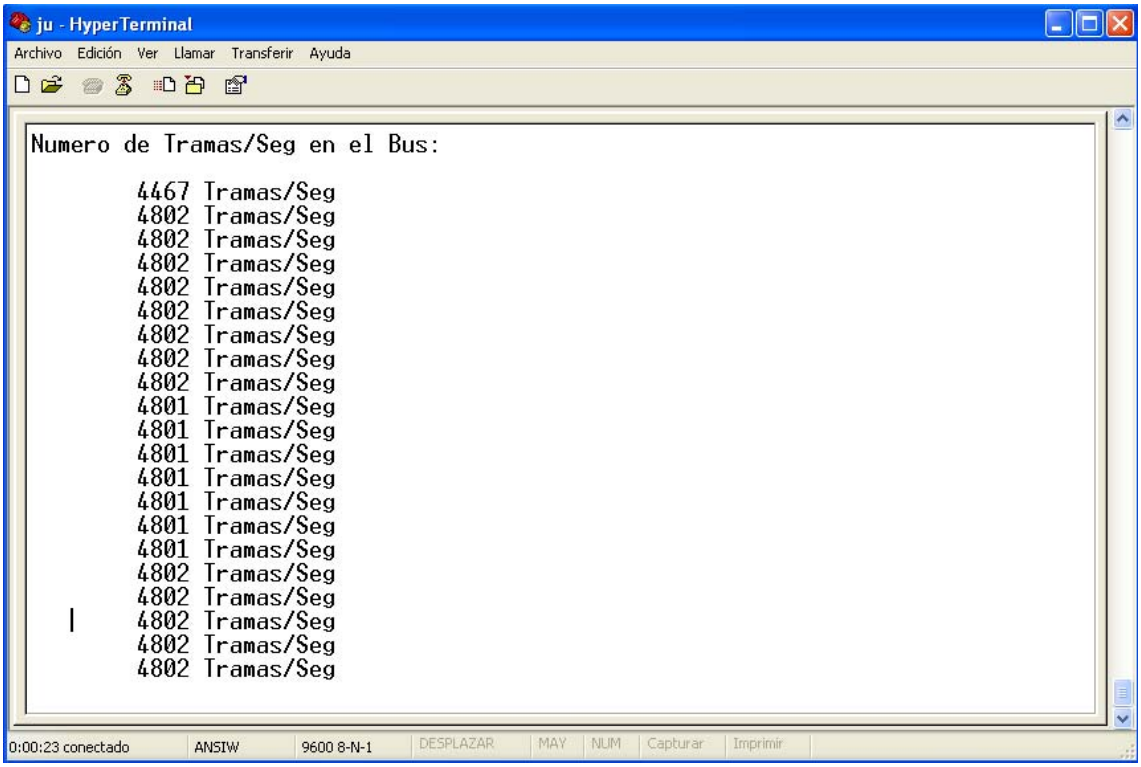


Figura 65: Nº de tramas por Segundo con un Nodos.

Esta prueba confirma los cálculos teóricos anteriores ya que el número de mensajes recibidos es muy similar al calculado. Aun así, se ha decidido realizar esta misma prueba con dos encoder conectados al Bus para confirmar que en la prueba anterior no existen limitaciones en el ancho de banda calculado para el Bus ó que se estén perdiendo tramas. Como se puede comprobar en la siguiente figura, los resultados obtenidos confirman que los cálculos teóricos del ancho de banda del

Bus también se corresponden con los obtenidos.

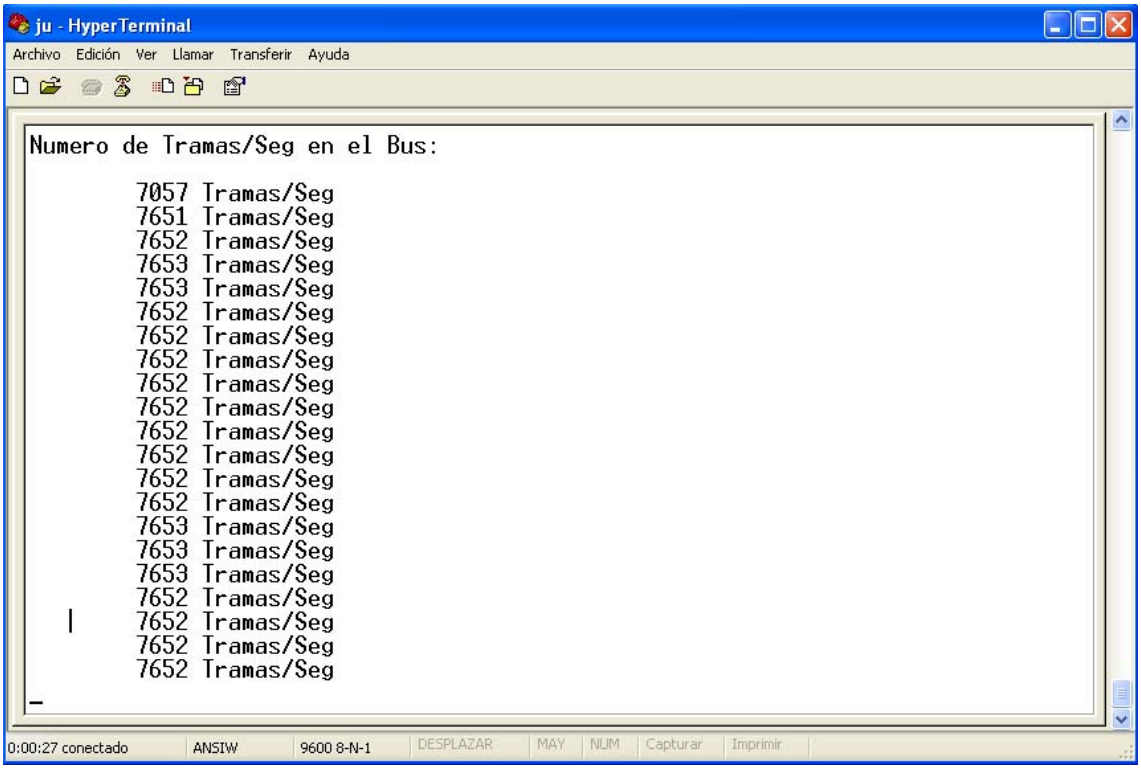


Figura 66: Nº de tramas por Segundo con 2 Nodos.

6.2 Conclusiones.

Los resultados obtenidos en el proyecto han sido satisfactorios ya que, además de conseguir diseñar un encoder absoluto con todas las funcionalidades de un Nodo CANopen, se ha logrado un diseño muy compacto y ligero. Estas características permiten una fácil integración en cualquier sistema de control robótico, donde el peso y tamaño de los componentes es un factor muy determinante.

Otro de los aspectos a tener muy en cuenta es que los sistemas CAN son muy robustos y fiables por lo que es uno de los métodos de comunicaciones a tener en cuenta en cualquier sistema de control.

Por otro lado el diseño software se ha desarrollado de tal forma que se ha conseguido un sistema determinista en el que los datos transmitidos por el Nodo siempre son actualizados y enviados tras una petición del sistema.

La cadencia máxima con la que el Nodo es capaz de introducir la lectura de posición en el Bus es muy aceptable (≈ 215 μs) permitiendo realizar control de sistemas en Tiempo Real. Aunque en este aspecto hemos de tener en cuenta el número de Nodos conectados al Bus y el periodo de muestreo del sistema. Este aspecto depende de la tasa de transferencia del Bus, que puede ser aumentada a un máximo de **1 Mbps**. La cadencia máxima podría ser reducida modificando el oscilador principal del Nodo.

Entre los aspectos negativos debemos destacar la sensibilidad del encoder en cuanto a colocación se refiere y la importancia de disponer de un disco de lectura libre de rayones y suciedad. Para solucionar el problema de colocación del cabezal de lectura con respecto al disco seria suficiente con diseñar un sistema que permita ajustar la posición del cabezal mediante tornillos sin fin.

El disco ha de ser tratado con cuidado para que no sufra rayones y el ambiente de trabajo ha de estar libre de partículas y polvo suspendido en el aire para evitar que estas se depositen sobre el disco, e interfieran en la lectura. En el caso de que se detecten anomalías se deberá comprobar que el disco esta en perfecto estado.

6.3 TRABAJOS FUTUROS Y AMPLIACIONES.

En este apartado se plantearan futuras mejoras del encoder absoluto que permitan obtener más prestaciones y una mayor usabilidad del dispositivo. Entre estas prestaciones cabe destacar la implementación de los objetos opcionales descritos en el perfil de dispositivo DS-406 para que el encoder se comporte como un dispositivo de tipo clase 2.

Además de la implementación de dichos objetos, es posible aprovechar las características del Encoder AEAS-7000 para detectar posibles fallos de corriente en el dispositivo y por otro lado existe la posibilidad de un modo de lectura de precisión que aunque proporciona esta con menos resolución (12 bits), es más fiable.

En cuanto a la programación del microcontrolador, podría aprovecharse las características de la memoria FLASH para modificar ciertos parámetros del Nodo

como puede ser el NodeID, la tasa de transferencia ó incluso la implementación de mapeo dinámico de PDOs sin la necesidad de reprogramar el micro.

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.

Por último, mencionar que el ciclo de programa podría ser reducido considerablemente aumentando la frecuencia del oscilador externo en el caso de que fuera necesario obtener lecturas del encoder con un periodo inferior a 215 μs. Aunque se a de tener en cuenta que el número de transmisiones está realmente limita por la tasa de transmisión, ya que el encoder no será el único dispositivo que transmita por el Bus.

Capítulo 7.

BIBLIOGRAFÍA.

7. BIBLIOGRAFÍA.

ESPECIFICACIONES:

[1] 'BOSCH Controller Area Network protocol specification', Revision 2.0, Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, 1991.

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

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

[4] 'Device profile for encoders' Draft Standard DS-406. Revision 3.2, CAN in Automation Group, December 2006.

[5] 'CAN Physical Layer for Industrial Applications' Draft Standard 102. Revision 2.0, CAN in Automation Group, April 1994.

ARTICULOS:

[6] Alberto Jardón Huete; Javier Gonzalez-Quijano Alvarez; Antonio Gimenez Fernandez, 'Optimal gait synthesis of biped robot PASIBOT using artificial intelligence based predictive control'.

[7] M. Farsi; K. Ratchiff; M. Barbosa, 'An Introduction to CANopen', Computing & Control Engineering Journal, Vol. 10, No. 4, pp. 161-168, June 1999.

[8] M. Farsi; K. Ratcliff; M. Barbosa, 'An Overview of Controller Area Network', *Computing & Control Engineering Journal*, Vol. 10, pp. 113-120, June 1999.

[9] K. W. Tindell; A. Burns; A. J. Wellings, 'Calculating Controller Area Network (CAN) messages response times' *Control Eng. Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[10] M Farsi; K Ratcliff, 'CANopen: Configure and device testing'. Factory Communication Systems, 1997. Proceedings.1997 IEEE International Workshop on 1-3 Oct. 1997 Page(s):373 – 380.

[11] Minkoo Kang; Kiejin Park; Bongjun Kim, 'PDO Packing Mechanism for Minimizing CANopen Network Utilization'. Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE 10-13 Nov. 2008 Page(s):1516-1519

PROYECTOS:

[12] Víctor Placer De Miguel, 'Implementación de las comunicaciones internas mediante CANbus en el robot ASIBOT_V1.5'. Universidad Carlos III de Madrid, 2008. Director: Alberto Jardón Huete.

[13] Aitor Olarra Urberuaga, 'Diseño e implementación de nodos de comunicación basados en el bus CAN y su aplicación a un móvil autónomo'. Escuela Técnica Superior de Ingenieros de Bilbao, 2001.

[14] Javier García Juberías, 'Desarrollo de un joystick wifi para teleoperación del robot asistencial ASIBOT'. Universidad Carlos III de Madrid, 2008. Director: Alberto Jardón Huete.

PÁGINAS WEB:

[15] http://www.softing.com. Última fecha acceso: 17/06/09

[16] http://www.microchip.com. Última fecha acceso: 22/05/09

[17] http://www.la-roomba.com. Última fecha acceso: 3/07/09

OTROS:

[18] Ross M. Fosler, 'A CANopen Stack for PIC18 ECAN™ Microcontrollers'. Microchip Technology Incorporated.

[19] Christian Dressler; Olga Fischer; Monika Mack; Reiner Zitzmann, 'CANdictinary' Revision 4.0, May 2007.

Capítulo 8.

ANEXOS.

8. ANEXOS.

8.1 Presupuesto.

A continuación se muestra un resumen del presupuesto de los componentes necesarios para la implementación del encoder absoluto CANopen. En este presupuesto no se tendrá en cuenta la mano de obra y los costes de diseño del dispositivo.

Resumen de Presupuesto de materiales	**Importe**
Encoder Agilent AEAS-7000.	112,32 €
Microcontrolador PIC18F2580.	6,91 €
Transceiver MCP2551.	1,79 €
Placa PCB y otros componentes. (incluye conectores, condensadores, resistencias, led, interuptor y oscilador)	247,76 €
TOTAL PRESUPUESTO	368,78 €
IVA 16%	59,00 €
TOTAL PRESUPUESTO CON IVA	**427,78 €**

Tabla 40: Resumen de Presupuesto.

8.2 Listado de componentes.

En la siguiente tabla se muestran todos los componentes necesarios para la implementación del encoder absoluto CANopen.

COMPONENTES	VALOR	CANTIDAD	MODELO	FOOTPRINT	DESCRIPCION
Encoder absoluto	-	1	Agilent AEAS-7000	CONN_RCPT_11X2	Encoder absoluto
µcontrolador	-	1	PIC18F2580	SOG.050/28/WG.480/L.700	µcontrolador
Transceiver	-	1	MCP2551	SOG.050/8/WG.244/L.175	Transceiver
XTAL	20MHz	1	Cristal,20MHz HC49/4H	QUARTZ20	Cristal de cuarzo
Diodo led	-	1	Rojo	LEDROJO	Indicador estado
Pulsador	-	1	4 pines	MICROSWITCH	Circuito reset
Conectores					
Conector micro-macth	Hembra	1	6 pines	CON6_2.54	Conector programación
Conector micro-macth	Hembra	1	4 pines	CON4_2.54	Conector CANbus + alimentación
Conector	Macho	1	2 pines	CLEMA2_2.54	Conector aux. CANbus
Condensadores					
C	15pF	2	ceramic cap, 0402, 15pF	0402	XTAL
C	100µF	1	Tantalum capacitor	COND100U_REV1	Filtro RC del encoder
C	2,2µF	1	Tantalum capacitor	COND100U_REV1	Filtro RC del encoder
C	100n	1	Ceramic cap, 0805, 100nF	R0805_REV2	Circuito reset
Resistencias	Ω				
R	3K3	1	Chip precisión 0805 RN, 3k3	R0805	Circuito reset
R	2K2	1	Chip precisión 0805 RN, 2k2	R0805	Circuito reset
R	390	1	Chip precisión 0603 RN, 390R	R0603	Polarización led
R	10	2	Chip precisión 0603 RN, 10R	R0603	Filtro RC del encoder

Tabla 41: Listado de Componentes.

8.3 Circuitos impresos y esquemático de la placa.

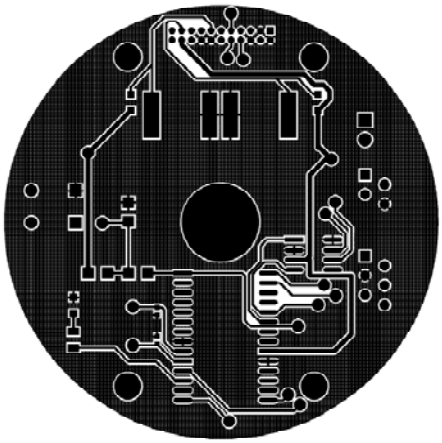


Figura 67: Cara TOP de la Placa.

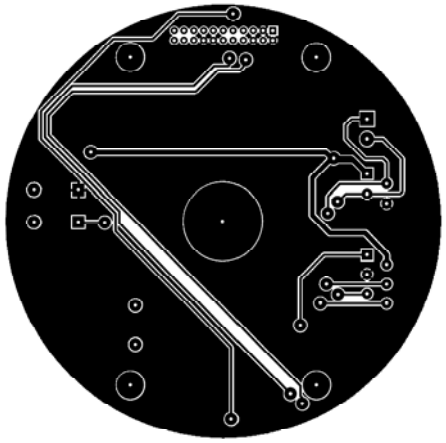


Figura 68: Cara BOTTOM de la Placa.

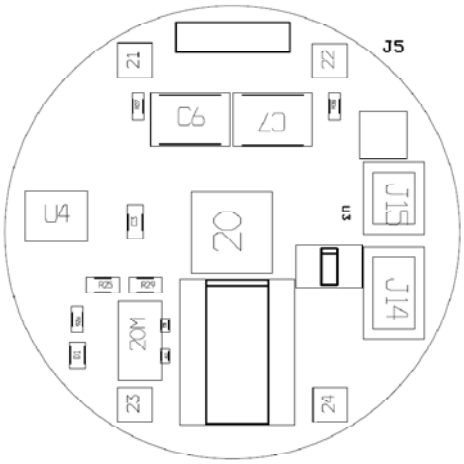


Figura 69: Serigrafía SSTOP de la Placa.

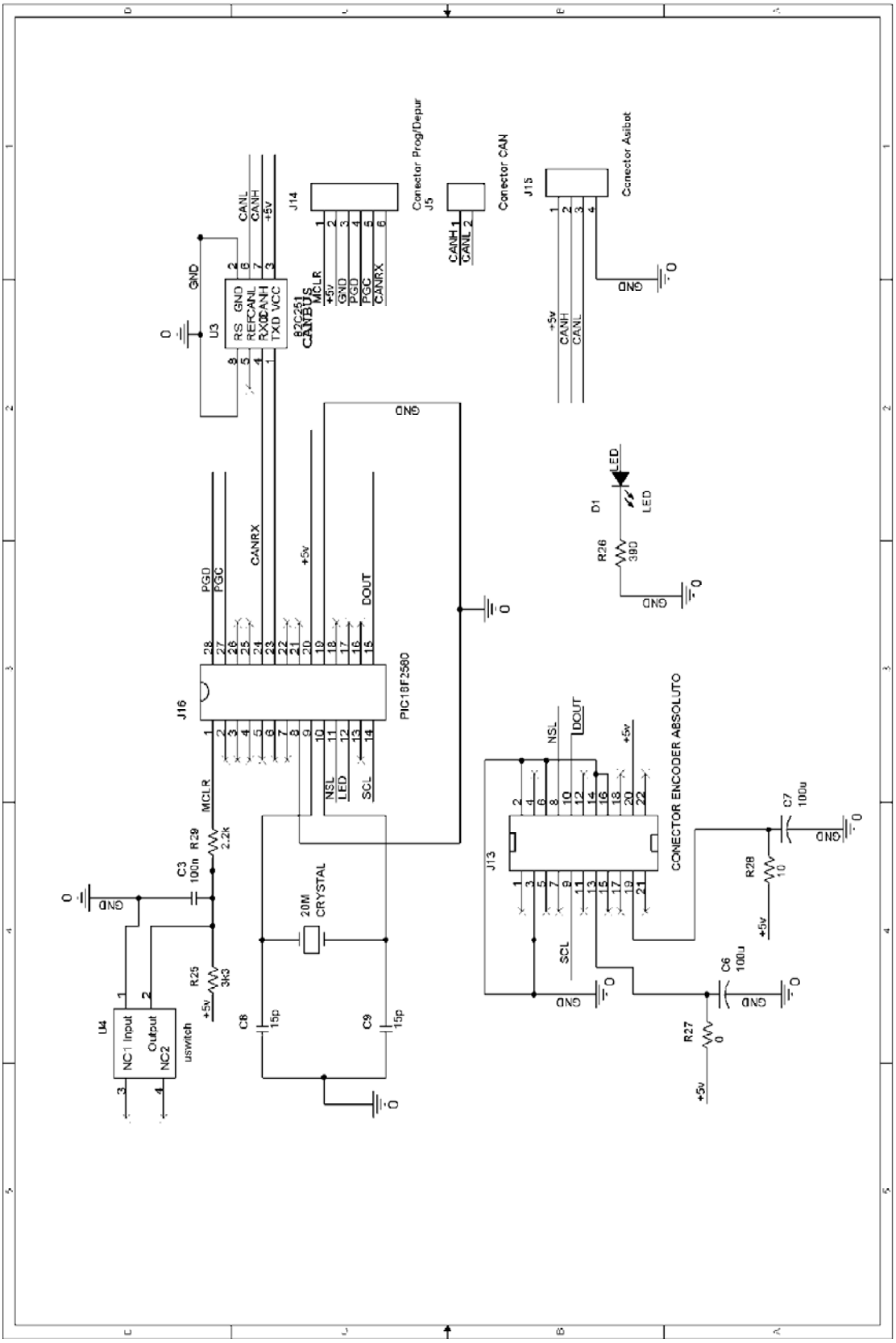


Figura 70: Esquemático de la Placa.

8.4 Encoder Agilent AEAS-7000.

Agilent AEAS-7000
Plug and Play Ultra-Precision Absolute Encoder 16-bit Gray Code
Data Sheet

Description

The encoder IC consists of 13 signal photo diode channels and 1 monitor photo diode channel and is used for the optical reading of rotary carriers (i.e., discs). The photodiodes are accompanied with precision amplifiers plus additional circuitry.

The monitor channel is used to drive a constant current source for the highly collimated IR illumination system.

Functional Description

Background

The 13 signal channels are set up as:

1. Two precision defining signals (A0, A09), which are two 90° electrical shifted sine, cosine signals. These signals are conditioned to be compensated for offset and gain errors. After conditioning they are on chip interpolated (4 bit) and computed to an absolute 6 bit Gray code. Additionally, these Sin/Cos signals can be tapped as two true-differential analog outputs to be used at the system designer's choice.
2. 11 analog (A1-A11) channels which are directly digitized by precison comparators with hysterisis tracking. The digitized signals are called D1-D11.

An internal correction and synchronization module allows the composition of a true 16 bit gray code by merging the data bits of 1) and 2) by still keeping the code monotony.

There is a Gray code correction feature for this encoder to counter any codewheel imperfection or misalignment. This Gray code correction can be disabled/enabled by the pin KORR.

The gain and offset conditioning value of the sine and cosine wave has been on-chip preset by factory. This will compensate for mechanical sensor misalignment error.

Features

- Minumum mechanical alignment during installation
- 2 Sine/Cosine true differential outputs with 1024 periods for unit alignment
- Integrated highly collimated illumination system
- 11 digital tracks plus 2 sin/cos tracks generate precise 16 bit Gray code
- Ultra fast, 1 µs cycle for serial data output word equals 16 MHz
- On-chip interpolation and code correction to compensate for mounting tolerance
- MSB can be inverted for changing the counting direction
- Internally built in monitor track for tracking the light level
- Watch dog with alarm output
- –25°C to +85°C operating temp.

Applications

- Rotary application up to 16 bit/ 360° absolute position
- Rotary application up to 11 bit user defined code patterns
- Cost effective solution for direct integration into OEM systems

Agilent Technologies

Signal-Channels A1-A11

The photocurrent of the photo diodes is fed into a trans-impedance amplifier. The analog output of the amplifier has a voltage swing of (dark/light) about 1.3 V. Every output is transformed by precision comparators into digital signals (D1-D11). The threshold is at VDD/2 (=Analog-reference),regulated by the monitor channel.

Monitor Channel with LED Control at Pins LEDR and LERR

The analog output signal of the monitor channel is regulated by the LED current. An internal bipolar transistor sets this level to VDD/2 (control voltage at pin LEDR). Thus the signal swing of each output is symmetrical to VDD/2 (=Analog-reference)

The error bit at pin LERR is triggered if the Ve of the internal bipolar transistor is larger than VDD/2.

Signals Channels A0, A09 with Signal Conditioning and Self Calibration

These two channels give out a sine and cosine wave which are 90 deg phase shifted. These signals have amplitudes which are almost constant due to the LED current monitoring. Due to amplifier mismatch the signals do have gain and offset errors. These errors are eliminated by an adaptive signal conditioning circuitry. The conditioning values are on-chip preprogrammed by factory. The analog output signals of A0 and A09 are supplied as true-differential voltage with a peak to peak value of 2.0 V at the pins A09P, A09N, A0P, A0N.

Interpolator for Channels A0, A09

The interpolator generates the digital signals D0,D09 and D-1 to D-4. The interpolated signals D-1 to D-4 extend the 12 bit Gray code of the signals D11....D0 to form a 16 bit Gray code.

D0 and D09 are digitized from A0 and A09. The channels A0-A11 and A09 have very high dynamic bandwidth, which allows a real time monotone 12 bit Gray code at 12000 RPM.

The interpolated 16 bit Gray code can be used up to 1000 RPM only. At more than 1000 RPM, only the 12 bit Gray code from the MSB side can be used.

LSB Gray Code Correction (Pin KORR)

This function block synchronizes the switching points for the 11 bit gray code of the digital signals D1 to D11 with D0 and D09 (digitized signal of A0 and A09).

This Gray code correction only works for the 12 bit MSB(4096 steps per revolution).

It does not work for the 4 excess interpolated bits of the 16 bit Gray code.

When some special applications require code patterns other than Gray code, the Gray code correction can be disabled by putting pin KORR = 0. When that happens just the 11 data bits (D1...D11) will be sent 1:1 to the DOUT serial output.

Gray code correction can be switched on or off by putting the pin KORR =1 (on) or =0 (off).

MSBINV and DOUT Pins

The serial interface consists of a shift register. The most significant bit, MSB (D11) will always be sent first to DOUT. The MSB can be inverted (change code direction) by using pin MSBINV.

DIN and NSL Pins

The Serial input DIN allows the configuration as ring register for multiple transmissions or for cascading 2 or more encoders. DIN is the input of the shift register that shifts the data to DOUT.

The NSL pin controls the shift register, to switch it between load (1) or shift (0) mode. Under load mode, DOUT will give the logic of the MSB, i.e., D11.

Under shift mode (0), coupled with the SCL, the register will be clocked, and gives out the serial word output bit by bit. As the clock frequency can be up to 16 MHz, the transmission of the full 16 bit word can be done within 1µs.

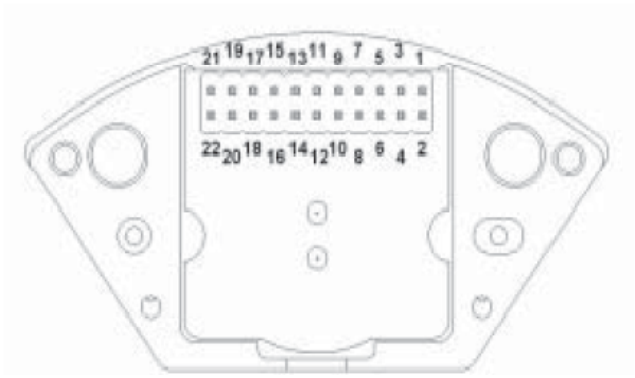
Valid data of DOUT should be read when the SCL clock is low. Please refer to timing diagram (Figure 2).

Pinout Description

No.	Pin Name	Description	Function	Notes[1]
1	NC		Internally connected to cathode of LED	Do not use
2	KORR	Digital-input	1 = Gray Code Correction Active	CMOS, internal pu
3	PROBE_ON	Digital-Input	Do not use	CMOS, internal pd
4	PCL	Digital Input Positive Edge	Do not use	CMOS, internal pu
5	STCAL	Digital Input Positive edge Negative edge	To be ground	CMOS, internal pd
6	MSBINV	Digital-Input	1 = MSB inverted	CMOS, internal pd
7	DIN	Digital Input	Shift Register input. Used for cascading only	CMOS, internal pd
8	NSL	Digital-Input	Shift-register Shift (=0) / Load (=1) Control	CMOS, internal pu
9	SCL	Digital-Input Positive Edge	Shift-register Shift Clock	CMOS, internal pu
10	DOUT	Digital Output	Shift-Register Data Out (MSB first)	CMOS, 2 mA
11	DO	Digital Output	DO signal	CMOS, 2 mA
12	DPROBE	Digital Output	DO9 signal	CMOS, 2 mA
13	VDD	Supply Voltage	+5 V Supply Digital	
14	GND	Gnd for supply voltage	GND for 5 V supply analog/digital	
15	A09P	Analog output	A09 positive (+True diff.)	CMOS, analog out
16	GND	Gnd for supply voltage	GND for 5 V supply analog/digital	
17	A0P	Analog Output	A0 positive (+True diff.)	CMOS, analog out
18	A09N	Analog output	A09 negative (–True diff.)	CMOS, analog out
19	VDDA	Supply Voltage	+5 V Supply Analog	
20	A0N	Analog Output	A0 negative (–True diff.)	CMOS, analog out
21	LERR	Digital Output	IR-LED Current Limit Signal	CMOS, 2 mA
22	LEDR	Analog Output	Do not use	CMOS, analog out

Note:
1. Internal pu/pd = internal pull-up (typ. 50 µA)/ pull-down (typ. 10 µA) CMOS-transistor-Rs.

Pinout Configuration



ESD WARNING: HANDLING PRECAUTIONS SHOULD BE TAKEN TO AVOID STATIC DISCHARGE

Using the AEAS-7000

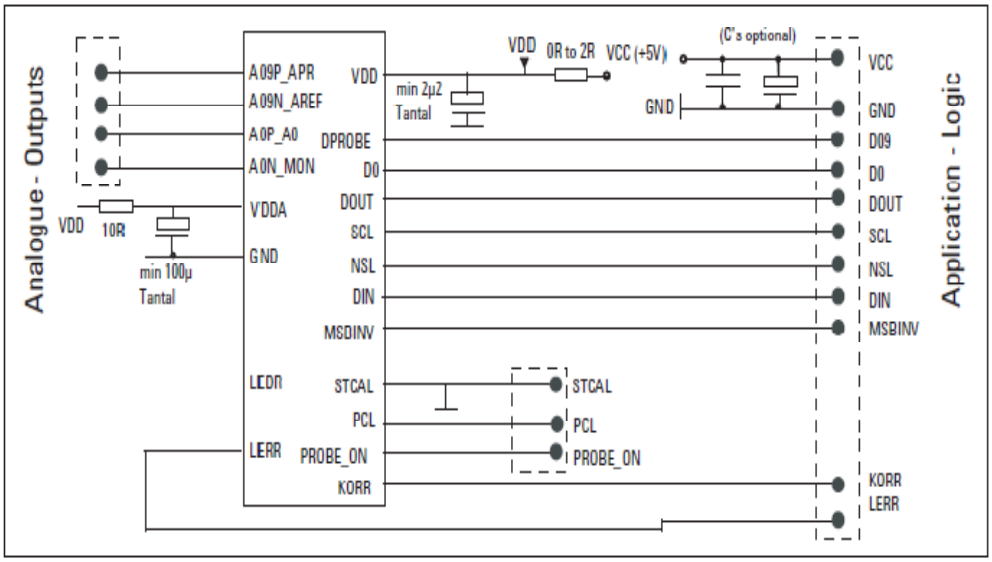


Figure 1. Schematic using AEAS 7000.

Note: The RC-filter combination, especially on VDDA, is used to filter spikes and transients and is strongly recommended. It is advised that the tantalum caps be put as close to the VDD and VDDA pins as possible.

It is recommended to ground the PROBE_ON pin during normal operation.

Leave PCL unconnected.

A09N and A0N are the negative cosine and sine waves, the negative versions of A09P and A0P.

D0 is used to check the D0 signal. D0 is the digitized signal of A0. DPROBE is used to check D09, the digitized signal of A09. Recommended to be used for testing purpose only.

KORR is for Gray Code correction for 12 bits resolution only.

MSBINV is for user to change between counting up and counting down for a given rotating direction. MSB(D11) will always be sent out to DOUT first.

LEDR, do not connect to this pin.

LERR will be high when the light output of the emitter is low. This is an indicator when light intensity is at a critical stage affecting the performance of the encoder. It is caused by contamination of the codewheel or LED degradation.

Operation

1) After powering up the unit using V_{CC} =+5 V and connecting GND to ground, trigger input pins NSL and SCL using the timing diagram below (Figure 2). NSL is a control pin for the internal shift register. When triggered to low and combined with clock pulses, the serial Gray code will be shifted out to DOUT bit by bit per every clock pulse

2) The 16 bit serial gray code can then be tapped out from the pin DOUT, most significant bit (D11) first.

The rate of the 16 bit Gray code serial transfer rate is dependent on the SCL clock frequency. The faster the clock, the faster the transfer rate. The maximum clock rate the AEAS-7000 can take is 16 MHz, which means the entire 16 bit Gray code can be serially transferred out in 1 µs.

3)Whenever NSL is high, the DOUT will have the logic of the MSB D11. After NSL goes low, the number of bits being trans-ferred out will depend on the number of clock pulses given to SCL. The default is 16 clock pulses for the 16 bit Gray code. If for other application where another number other than 16 is needed, just supply the corresponding number of clock pulses to the SCL, e.g., 12 bit, 13 bit, 14 bit or 15 bit, and you will get the corresponding length of Gray code words with the corresponding resolution.

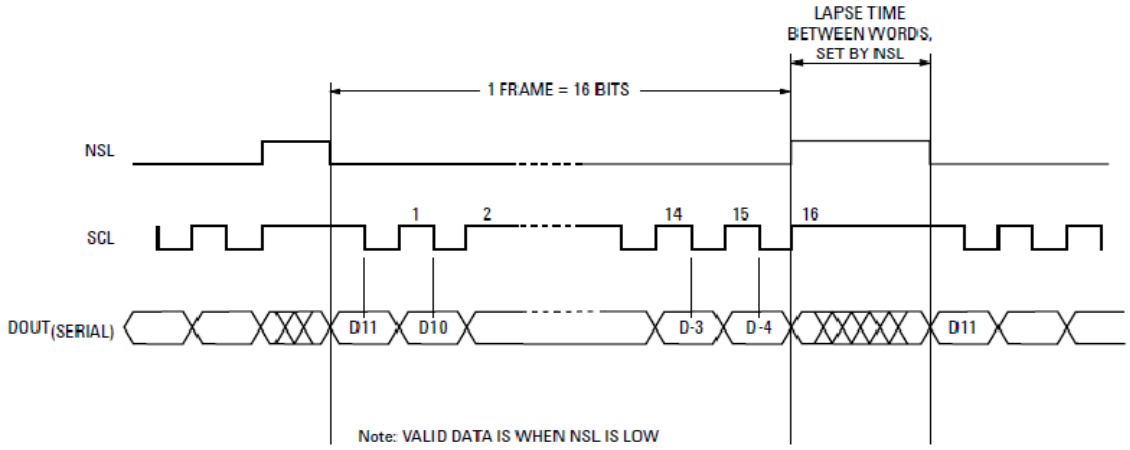


Figure 2. Timing diagram.

Absolute Limits

No.	Parameters	Symbol	Min.	Typ.	Max.	Units
1	Supply Voltage	VD	–0.3		6.0	V
2	Voltages at all Input and Output Pins	Vin , Vout	–0.3		VD + 0.3	V
3	Operating Temperature	T_A	–25		+85	°C
4	Storage Temperature	T_S	–40		+100	°C

Operating Conditions

No.	Parameters	Symbol	Min.	Typ.	Max	Units
1	Supply Voltage	VD	4.5	5	5.5	V
2	Operating Temperature	T_A	–25	25	+85	°C
3	Input-H-Level	Vih	0.7*VD		VD	V
4	Input-L-Level	Vil	0		0.3*VD	V

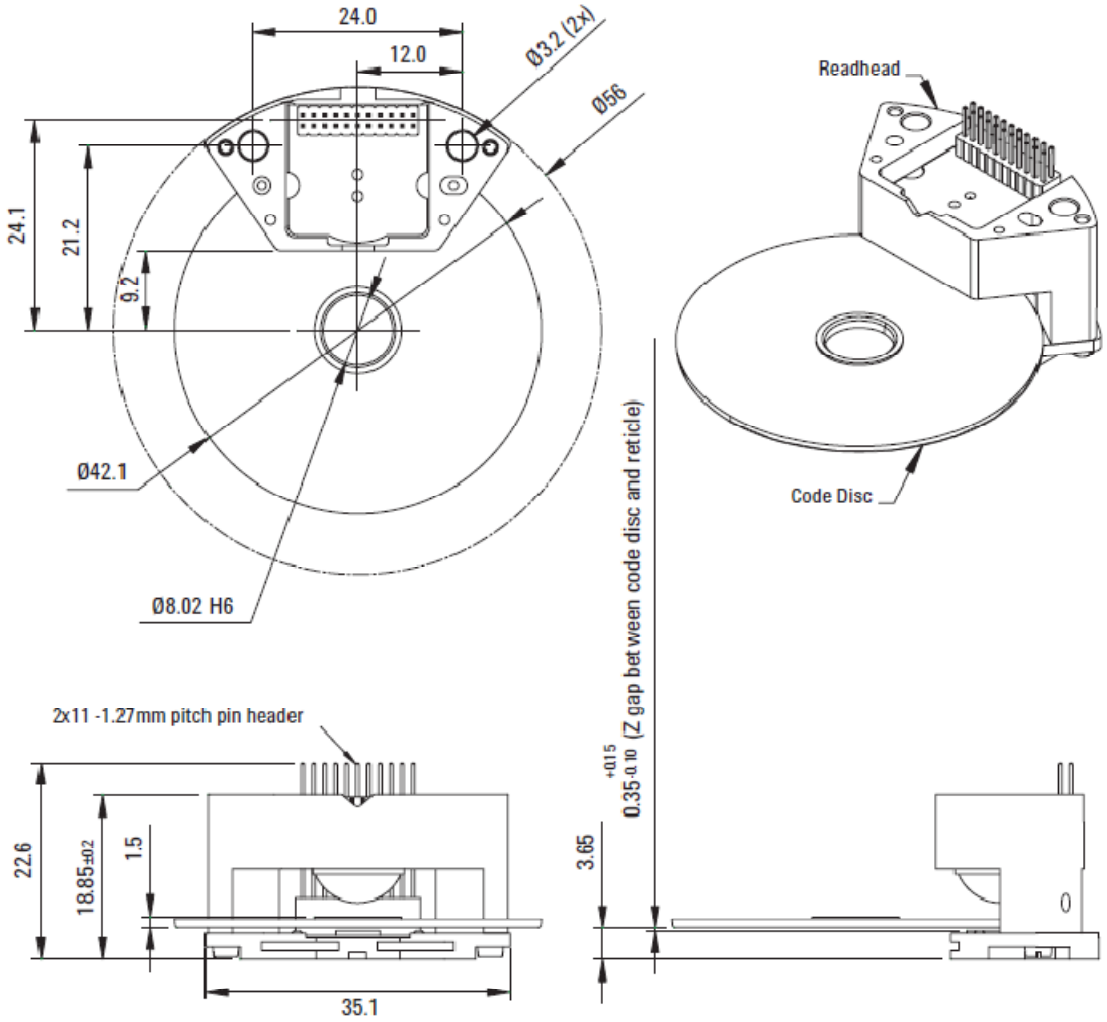
Electrical Characteristics (VD = 4.5 to 5 V, T_A = –40 to +85 °C)

No.	Parameters	Symbol	Conditions	Min.	Typ.	Max.	Units
Operating Currents							
1	Total Current	I total			25		mA
Digital Inputs							
1	Pull Down Current	Ipd		–20		–5	µA
2	Pull Up Current	Ipu		30		160	µA
Digital Outputs							
1	Ouput-H-Level	Voh	Ioh = 2 mA	VD - 0.5 V		VD	V
2	Output-L-Level	Vol	Iol = –2 mA	0		0.5	V
Serial Interface							
1	SCL Clock Frequency	fclock				16	MHz
2	Duty Cycle Fclock	T clock,LH	Fclock = 16 MHz	0.4		0.6	ns
3	Accuracy (1)		Fclock = 5MHz, RPM = 80		±2bits		
Analog-Signal-Conditioning– Signaltracks A0P, A0N, A09P, A09N							
1.	Signal Frequency A0, A09	Fsine,cos		0		250	KHz

Note 1:

Accuracy would be influenced by installation control and the bearing and shaft type being used.

Mounting Consideration



UNLESS SPECIFIED OTHERWISE DIMENSIONS ARE IN MILLIMETRES THIRD ANGLE PROJECTION		
XX.	0.3	STRIKE OUT
XX.X	0.1	OR FILL IN
XX.XX	0.03	AS NEEDED

Note: Codewheel mounting tolerances for radial, tangential and Z gap are:

Radial:	±50 um
Tangential:	±40 um
Z Gap:	±50 um

Plug & Play Hub-Shaft design

The following details the design of the hub-shaft of which the dimensions must be strictly followed for the plug & play feature of the AEAS-7000 to work. In order to secure the code disk to the hub, an adhesive must be utilised. Agilent recommends using DELO-DUOPOX, 1895 from DELO. Stainless steel is recommended as the hub-shaft material.

A complete instruction for AEAS-7000 Plug & Play installation consideration can be found in AEAS-7000 application note.

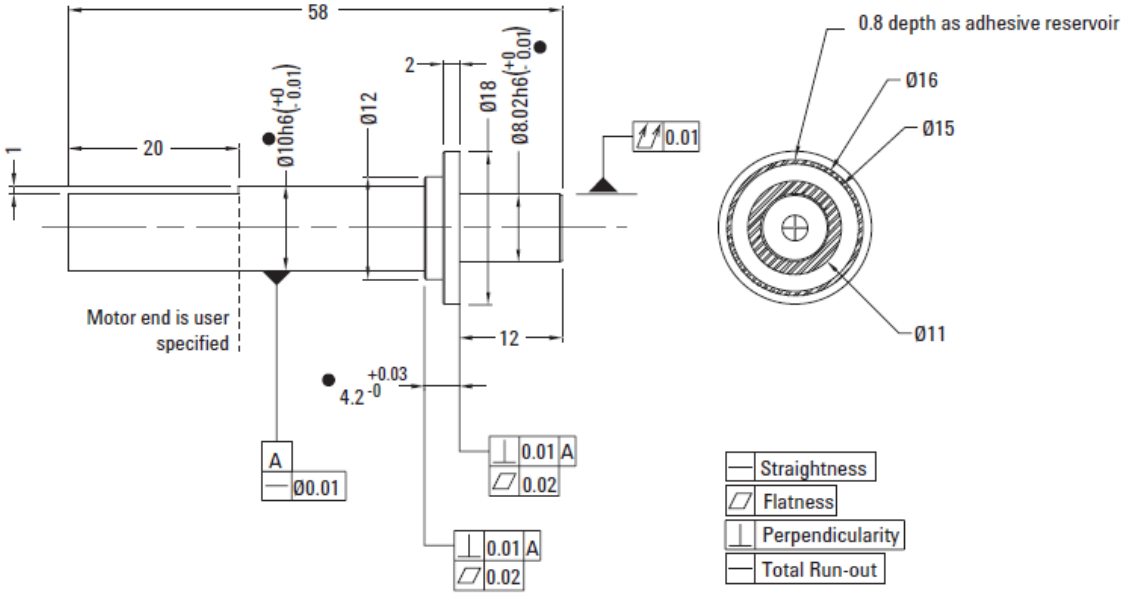
Ordering Information

AEA ☐ - 7000 - 1 G S ☐ 0

S - Standard (-25°C to +85°C)	D - 13 bits G - 16 bits

Legend

1 = 5V
G = gray code
S = serial output mode



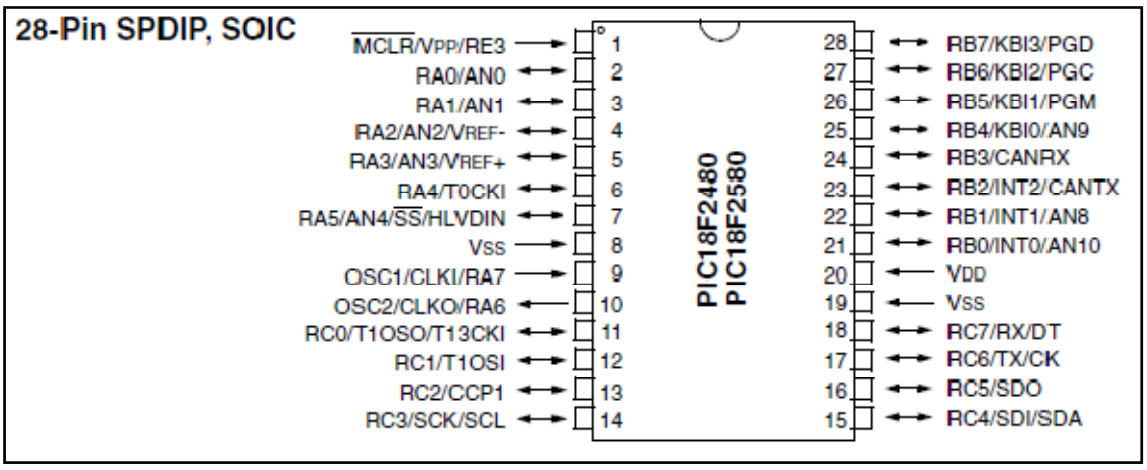
8.5 Microcontrolador PIC18F2580.

MICROCHIP

PIC18F2480/2580/4480/4580
Data Sheet

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

Pin diagram



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 I²C™ 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 (A/D) module, up to 100 ksps
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual Analog Comparators with Input Multiplexing

ECAN Technology 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 I²C™			
PIC18F2480	16K	8192	768	256	25	8	1/0	Y	Y	1	0	1/3
PIC18F2580	32K	16384	1536	256	25	8	1/0	Y	Y	1	0	1/3
PIC18F4480	16K	8192	768	256	36	11	1/1	Y	Y	1	2	1/3
PIC18F4580	32K	16384	1536	256	36	11	1/1	Y	Y	1	2	1/3

PIC18F2480/2580/4480/4580

1.0 DEVICE OVERVIEW

This document contains device specific information for the following devices:

- PIC18F2480
- PIC18F2580
- PIC18F4480
- PIC18F4580

This family of devices offers the advantages of all PIC18 microcontrollers – namely, high computational performance at an economical price – with the addition of high-endurance, Enhanced Flash program memory. In addition to these features, the PIC18F2480/2580/4480/4580 family introduces design enhancements that make these microcontrollers a logical choice for many high-performance, power sensitive applications.

1.1 New Core Features

1.1.1 nanoWatt TECHNOLOGY

All of the devices in the PIC18F2480/2580/4480/4580 family incorporate a range of features that can significantly reduce power consumption during operation. Key items include:

- **Alternate Run Modes:** By clocking the controller from the Timer1 source or the internal oscillator block, power consumption during code execution can be reduced by as much as 90%.
- **Multiple Idle Modes:** The controller can also run with its CPU core disabled but the peripherals still active. In these states, power consumption can be reduced even further, to as little as 4% of normal operation requirements.
- **On-the-Fly Mode Switching:** The power-managed modes are invoked by user code during operation, allowing the user to incorporate power-saving ideas into their application's software design.
- **Lower Consumption in Key Modules:** The power requirements for both Timer1 and the Watchdog Timer have been reduced by up to 80%, with typical values of 1.1 and 2.1 µA, respectively.
- **Extended Instruction Set:** In addition to the standard 75 instructions of the PIC18 instruction set, PIC18F2480/2580/4480/4580 devices also provide an optional extension to the core CPU functionality. The added features include eight additional instructions that augment indirect and indexed addressing operations and the implementation of Indexed Literal Offset Addressing mode for many of the standard PIC18 instructions.

1.1.2 MULTIPLE OSCILLATOR OPTIONS AND FEATURES

All of the devices in the PIC18F2480/2580/4480/4580 family offer ten different oscillator options, allowing users a wide range of choices in developing application hardware. These include:

- Four Crystal modes, using crystals or ceramic resonators
- Two External Clock modes, offering the option of using two pins (oscillator input and a divide-by-4 clock output) or one pin (oscillator input, with the second pin reassigned as general I/O)
- Two External RC Oscillator modes with the same pin options as the External Clock modes
- An internal oscillator block which provides an 8 MHz clock (±2% accuracy) and an INTRC source (approximately 31 kHz, stable over temperature and VDD), as well as a range of 6 user selectable clock frequencies, between 125 kHz to 4 MHz, for a total of 8 clock frequencies. This option frees the two oscillator pins for use as additional general purpose I/O.
- A Phase Lock Loop (PLL) frequency multiplier, available to both the high-speed crystal and internal oscillator modes, which allows clock speeds of up to 40 MHz. Used with the internal oscillator, the PLL gives users a complete selection of clock speeds, from 31 kHz to 32 MHz – all without using an external crystal or clock circuit.

Besides its availability as a clock source, the internal oscillator block provides a stable reference source that gives the family additional features for robust operation:

- **Fail-Safe Clock Monitor:** This option constantly monitors the main clock source against a reference signal provided by the internal oscillator. If a clock failure occurs, the controller is switched to the internal oscillator block, allowing for continued low-speed operation or a safe application shutdown.
- **Two-Speed Start-up:** This option allows the internal oscillator to serve as the clock source from Power-on Reset, or wake-up from Sleep mode, until the primary clock source is available.

PIC18F2480/2580/4480/4580

1.2 Other Special Features

- **Memory Endurance:** The Enhanced Flash cells for both program memory and data EEPROM are rated to last for many thousands of erase/write cycles – up to 100,000 for program memory and 1,000,000 for EEPROM. Data retention without refresh is conservatively estimated to be greater than 40 years.
- **Self-Programmability:** These devices can write to their own program memory spaces under internal software control. By using a bootloader routine located in the protected Boot Block at the top of program memory, it becomes possible to create an application that can update itself in the field.
- **Extended Instruction Set:** The PIC18F2480/2580/4480/4580 family introduces an optional extension to the PIC18 instruction set, which adds 8 new instructions and an Indexed Addressing mode. This extension, enabled as a device configuration option, has been specifically designed to optimize re-entrant application code originally developed in high-level languages, such as C.
- **Enhanced CCP Module:** In PWM mode, this module provides 1, 2 or 4 modulated outputs for controlling half-bridge and full-bridge drivers. Other features include auto-shutdown, for disabling PWM outputs on interrupt or other select conditions and auto-restart, to reactivate outputs once the condition has cleared.
- **Enhanced Addressable USART:** This serial communication module is capable of standard RS-232 operation and provides support for the LIN bus protocol. Other enhancements include automatic baud rate detection and a 16-bit Baud Rate Generator for improved resolution. When the microcontroller is using the internal oscillator block, the EUSART provides stable operation for applications that talk to the outside world without using an external crystal (or its accompanying power requirement).
- **10-Bit A/D Converter:** This module incorporates programmable acquisition time, allowing for a channel to be selected and a conversion to be initiated without waiting for a sampling period and thus, reduce code overhead.
- **Extended Watchdog Timer (WDT):** This enhanced version incorporates a 16-bit prescaler, allowing a time-out range from 4 ms to over 131 seconds, that is stable across operating voltage and temperature.

1.3 Details on Individual Family Members

Devices in the PIC18F2480/2580/4480/4580 family are available in 28-pin (PIC18F2X80) and 40/44-pin (PIC18F4X80) packages. Block diagrams for the two groups are shown in Figure 1-1 and Figure 1-2.

The devices are differentiated from each other in six ways:

1. Flash program memory (16 Kbytes for PIC18FX480 devices; 32 Kbytes for PIC18FX580 devices).
2. A/D channels (8 for PIC18F2X80 devices; 11 for PIC18F4X80 devices).
3. I/O ports (3 bidirectional ports and 1 input only port on PIC18F2X80 devices; 5 bidirectional ports on PIC18F4X80 devices).
4. CCP and Enhanced CCP implementation (PIC18F2X80 devices have 1 standard CCP module; PIC18F4X80 devices have one standard CCP module and one ECCP module).
5. Parallel Slave Port (present only on PIC18F4X80 devices).
6. PIC18F4X80 devices provide two comparators.

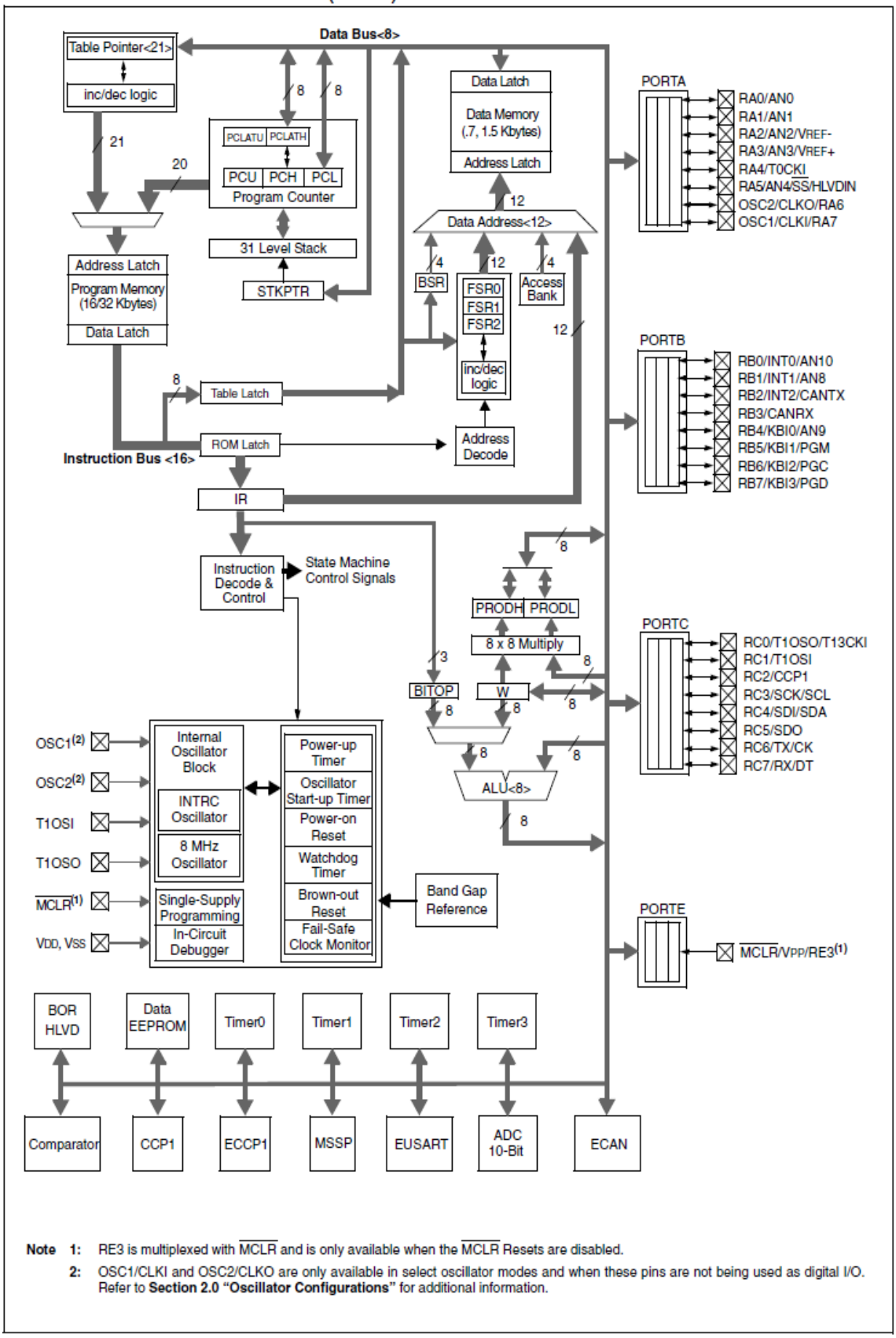
All other features for devices in this family are identical. These are summarized in Table 1-1.

The pinouts for all devices are listed in Table 1-2 and Table 1-3.

Like all Microchip PIC18 devices, members of the PIC18F2480/2580/4480/4580 family are available as both standard and low-voltage devices. Standard devices with Enhanced Flash memory, designated with an "F" in the part number (such as PIC18**F**2580), accommodate an operating VDD range of 4.2V to 5.5V. Low-voltage parts, designated by "LF" (such as PIC18**LF**2580), function over an extended VDD range of 2.0V to 5.5V.

PIC18F2480/2580/4480/4580

FIGURE 1-1: PIC18F2480/2580 (28-PIN) BLOCK DIAGRAM



Note 1: RE3 is multiplexed with $\overline{MCLR}$ and is only available when the $\overline{MCLR}$ Resets are disabled.

2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

PIC18F2480/2580/4480/4580

TABLE 1-1: DEVICE FEATURES

Features	PIC18F2480	PIC18F2580	PIC18F4480	PIC18F4580
Operating Frequency	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz
Program Memory (Bytes)	16384	32768	16384	32768
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	1	1	1	1
Enhanced Capture/ Compare/PWM Modules	0	0	1	1
ECAN Module	1	1	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Parallel Communications (PSP)	No	No	Yes	Yes
10-Bit Analog-to-Digital Module	8 Input Channels	8 Input Channels	11 Input Channels	11 Input Channels
Comparators	0	0	2	2
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT
Programmable High/ Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set Enabled	75 Instructions; 83 with Extended Instruction Set Enabled	75 Instructions; 83 with Extended Instruction Set Enabled	75 Instructions; 83 with Extended Instruction Set Enabled
Packages	28-pin SPDIP 28-pin SOIC 28-pin QFN	28-pin SPDIP 28-pin SOIC 28-pin QFN	40-pin PDIP 44-pin QFN 44-pin TQFP	40-pin PDIP 44-pin QFN 44-pin TQFP

TABLE 1-2: PIC18F2480/2580 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	SPDIP, SOIC	QFN			
MCLR/VPP/RE3	1	26			Master Clear (input) or programming voltage (input).
MCLR			I	ST	Master Clear (Reset) input. This pin is an active-low Reset to the device.
VPP			P		Programming voltage input.
RE3			I	ST	Digital input.
OSC1/CLKI/RA7	9	6			Oscillator crystal or external clock input.
OSC1			I	ST	Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; CMOS otherwise.
CLKI			I	CMOS	External clock source input. Always associated with pin function OSC1. (See related OSC1/CLKI, OSC2/CLKO pins.)
RA7			I/O	TTL	General purpose I/O pin.
OSC2/CLKO/RA6	10	7			Oscillator crystal or clock output.
OSC2			O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO			O	—	In RC mode, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
RA6			I/O	TTL	General purpose I/O pin.

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

TABLE 1-2: PIC18F2480/2580 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	SPDIP, SOIC	QFN			
					PORTA is a bidirectional I/O port.
RA0/AN0	2	27			
RA0			I/O	TTL	Digital I/O.
AN0			I	Analog	Analog input 0.
RA1/AN1	3	28			
RA1			I/O	TTL	Digital I/O.
AN1			I	Analog	Analog input 1.
RA2/AN2/VREF-	4	1			
RA2			I/O	TTL	Digital I/O.
AN2			I	Analog	Analog input 2.
VREF-			I	Analog	A/D reference voltage (low) input.
RA3/AN3/VREF+	5	2			
RA3			I/O	TTL	Digital I/O.
AN3			I	Analog	Analog input 3.
VREF+			I	Analog	A/D reference voltage (high) input.
RA4/T0CKI	6	3			
RA4			I/O	TTL	Digital I/O.
T0CKI			I	ST	Timer0 external clock input.
RA5/AN4/$\overline{SS}$/ HLVDIN	7	4			
RA5			I/O	TTL	Digital I/O.
AN4			I	Analog	Analog input 4.
$\overline{SS}$			I	TTL	SPI slave select input.
HLVDIN			I	Analog	High/Low-Voltage Detect input.
RA6					See the OSC2/CLKO/RA6 pin.
RA7					See the OSC1/CLKI/RA7 pin.
					PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT0/ AN10	21	18			
RB0			I/O	TTL	Digital I/O.
INT0			I	ST	External interrupt 0.
AN10			I	Analog	Analog input 10.
RB1/INT1/AN8	22	19			
RB1			I/O	TTL	Digital I/O.
INT1			I	ST	External interrupt 1.
AN8			I	Analog	Analog input 8.
RB2/INT2/CANTX	23	20			
RB2			I/O	TTL	Digital I/O.
INT2			I	ST	External interrupt 2.
CANTX			O	TTL	CAN bus TX.
RB3/CANRX	24	21			
RB3			I/O	TTL	Digital I/O.
CANRX			I	TTL	CAN bus RX.
RB4/KBI0/AN9	25	22			
RB4			I/O	TTL	Digital I/O.
KBI0			I	TTL	Interrupt-on-change pin.
AN9			I	Analog	Analog input 9.
RB5/KBI1/PGM	26	23			
RB5			I/O	TTL	Digital I/O.
KBI1			I	TTL	Interrupt-on-change pin.
PGM			I/O	ST	Low-Voltage ICSP™ Programming enable pin.
RB6/KBI2/PGC	27	24			
RB6			I/O	TTL	Digital I/O.
KBI2			I	TTL	Interrupt-on-change pin.
PGC			I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
RB7/KBI3/PGD	28	25			
RB7			I/O	TTL	Digital I/O.
KBI3			I	TTL	Interrupt-on-change pin.
PGD			I/O	ST	In-Circuit Debugger and ICSP programming data pin.

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

PIC18F2480/2580/4480/4580

TABLE 1-2: PIC18F2480/2580 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	SPDIP, SOIC	QFN			
					PORTC is a bidirectional I/O port.
RC0/T1OSO/T13CKI	11	8			
RC0			I/O	ST	Digital I/O.
T1OSO			O	—	Timer1 oscillator output.
T13CKI			I	ST	Timer1/Timer3 external clock input.
RC1/T1OSI	12	9			
RC1			I/O	ST	Digital I/O.
T1OSI			I	CMOS	Timer1 oscillator input.
RC2/CCP1	13	10			
RC2			I/O	ST	Digital I/O.
CCP1			I/O	ST	Capture 1 input/Compare 1 output/PWM1 output.
RC3/SCK/SCL	14	11			
RC3			I/O	ST	Digital I/O.
SCK			I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL			I/O	ST	Synchronous serial clock input/output for I^2C™ mode.
RC4/SDI/SDA	15	12			
RC4			I/O	ST	Digital I/O.
SDI			I	ST	SPI data in.
SDA			I/O	ST	I^2C data I/O.
RC5/SDO	16	13			
RC5			I/O	ST	Digital I/O.
SDO			O	—	SPI data out.
RC6/TX/CK	17	14			
RC6			I/O	ST	Digital I/O.
TX			O	—	EUSART asynchronous transmit.
CK			I/O	ST	EUSART synchronous clock (see related RX/DT).
RC7/RX/DT	18	15			
RC7			I/O	ST	Digital I/O.
RX			I	ST	EUSART asynchronous receive.
DT			I/O	ST	EUSART synchronous data (see related TX/CK).
RE3	—	—	—	—	See $\overline{MCLR}$/VPP/RE3 pin.
VSS	8, 19	5, 16	P	—	Ground reference for logic and I/O pins.
VDD	20	17	P	—	Positive supply for logic and I/O pins.

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

17.0 MASTER SYNCHRONOUS SERIAL PORT (MSSP) MODULE

17.1 Master SSP (MSSP) Module Overview

The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D Converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)
  - Full Master mode
  - Slave mode (with general address call)

The I^2C interface supports the following modes in hardware:

- Master mode
- Multi-Master mode
- Slave mode

17.2 Control Registers

The MSSP module has three associated registers. These include a status register (SSPSTAT) and two control registers (SSPCON1 and SSPCON2). The use of these registers and their individual configuration bits differ significantly depending on whether the MSSP module is operated in SPI or I^2C mode.

Additional details are provided under the individual sections.

17.3 SPI Mode

The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

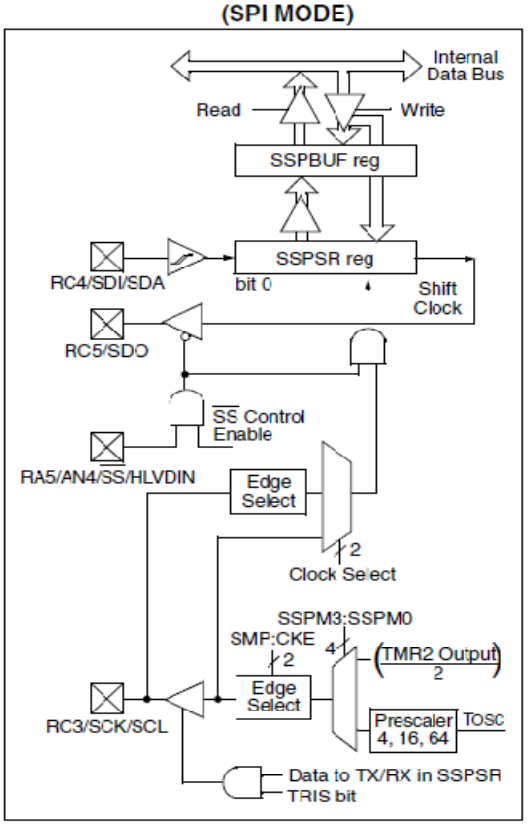
- Serial Data Out (SDO) – RC5/SDO
- Serial Data In (SDI) – RC4/SDI/SDA
- Serial Clock (SCK) – RC3/SCK/SCL

Additionally, a fourth pin may be used when in a Slave mode of operation:

- Slave Select ($\overline{SS}$) – RA5/AN4/$\overline{SS}$/HLVDIN

Figure 17-1 shows the block diagram of the MSSP module when operating in SPI mode.

FIGURE 17-1: MSSP BLOCK DIAGRAM (SPI MODE)



17.3.1 REGISTERS

The MSSP module has four registers for SPI mode operation. These are:

- MSSP Control Register 1 (SSPCON1)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR) – Not directly accessible

SSPCON1 and SSPSTAT are the control and status registers in SPI mode operation. The SSPCON1 register is readable and writable. The lower 6 bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

In receive operations, SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

23.0 ECAN MODULE

PIC18F2480/2580/4480/4580 devices contain an Enhanced Controller Area Network (ECAN) module. The ECAN module is fully backward compatible with the CAN module available in PIC18CXX8 and PIC18FXX8 devices.

The Controller Area Network (CAN) module is a serial interface which is useful for communicating with other peripherals or microcontroller devices. This interface, or protocol, was designed to allow communications within noisy environments.

The ECAN module is a communication controller, implementing the CAN 2.0A or B protocol as defined in the BOSCH specification. The module will support CAN 1.2, CAN 2.0A, CAN 2.0B Passive and CAN 2.0B Active versions of the protocol. The module implementation is a full CAN system; however, the CAN specification is not covered within this data sheet. Refer to the BOSCH CAN specification for further details.

The module features are as follows:

- Implementation of the CAN protocol CAN 1.2, CAN 2.0A and CAN 2.0B
- DeviceNet™ data bytes filter support
- Standard and extended data frames
- 0-8 bytes data length
- Programmable bit rate up to 1 Mbit/sec
- Fully backward compatible with PIC18XXX8 CAN module
- Three modes of operation:
  - Mode 0 – Legacy mode
  - Mode 1 – Enhanced Legacy mode with DeviceNet support
  - Mode 2 – FIFO mode with DeviceNet support
- Support for remote frames with automated handling
- Double-buffered receiver with two prioritized received message storage buffers
- Six buffers programmable as RX and TX message buffers
- 16 full (standard/extended identifier) acceptance filters that can be linked to one of four masks
- Two full acceptance filter masks that can be assigned to any filter
- One full acceptance filter that can be used as either an acceptance filter or acceptance filter mask
- Three dedicated transmit buffers with application specified prioritization and abort capability
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loopback mode supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to timer module for time-stamping and network synchronization
- Low-power Sleep mode

23.1 Module Overview

The CAN bus module consists of a protocol engine and message buffering and control. The CAN protocol engine automatically handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate data registers. Status and errors can be checked by reading the appropriate registers. Any message detected on the CAN bus is checked for errors and then matched against filters to see if it should be received and stored in one of the two receive registers.

The CAN module supports the following frame types:

- Standard Data Frame
- Extended Data Frame
- Remote Frame
- Error Frame
- Overload Frame Reception
- Interframe Space Generation/Detection

The CAN module uses the RB2/CANTX and RB3/CANRX pins to interface with the CAN bus. In normal mode, the CAN module automatically overrides TRISB<2>. The user must ensure that TRISB<3> is set.

23.1.1 MODULE FUNCTIONALITY

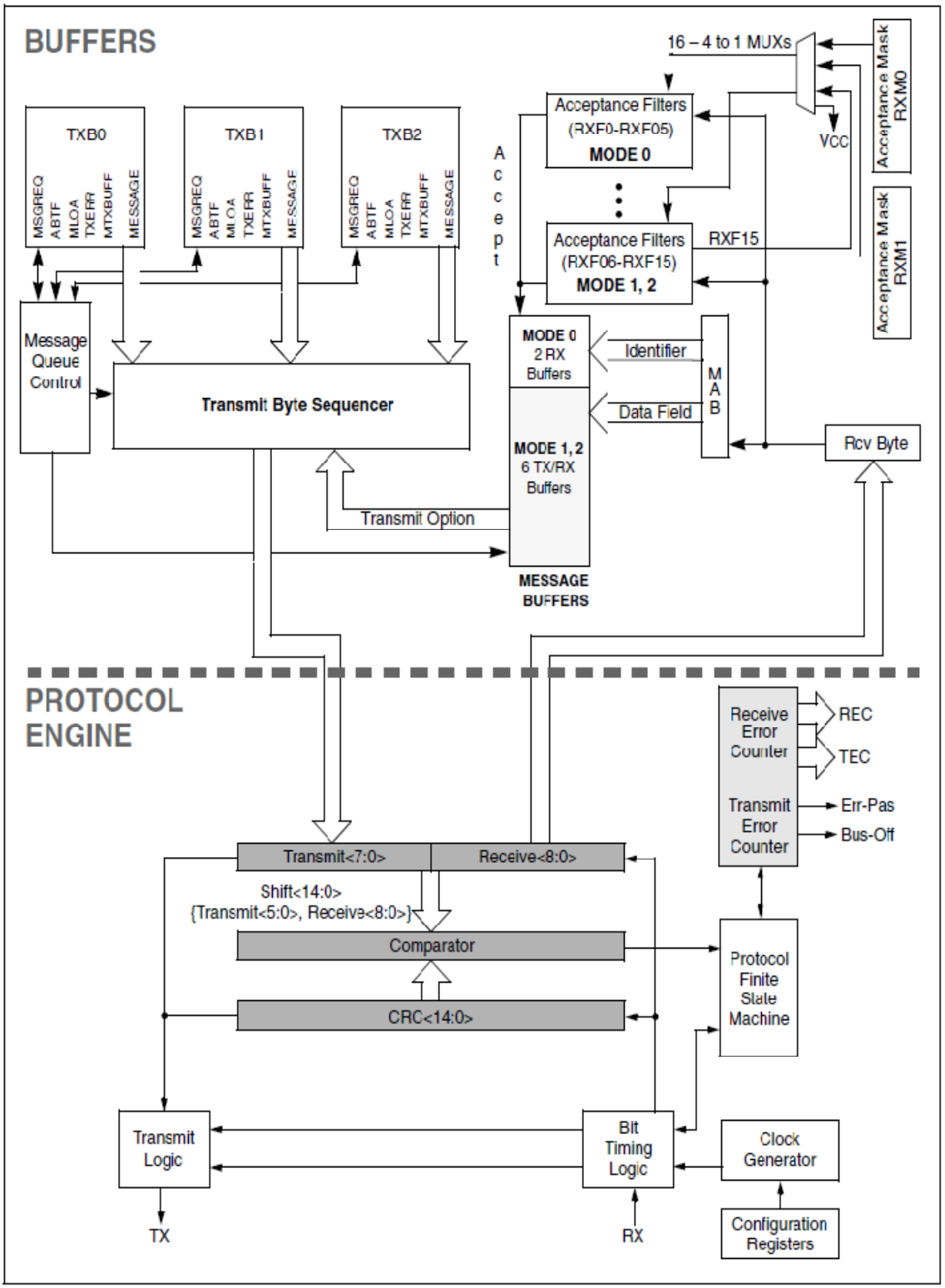
The CAN bus module consists of a protocol engine, message buffering and control (see Figure 23-1). The protocol engine can best be understood by defining the types of data frames to be transmitted and received by the module.

The following sequence illustrates the necessary initialization steps before the ECAN module can be used to transmit or receive a message. Steps can be added or removed depending on the requirements of the application.

1. Ensure that the ECAN module is in Configuration mode.
2. Select ECAN Operational mode.
3. Set up the baud rate registers.
4. Set up the filter and mask registers.
5. Set the ECAN module to normal mode or any other mode required by the application logic.

PIC18F2480/2580/4480/4580

FIGURE 23-1: CAN BUFFERS AND PROTOCOL ENGINE BLOCK DIAGRAM



27.1 DC Characteristics: Supply Voltage
PIC18F2480/2580/4480/4580 (Industrial, Extended)
PIC18LF2480/2580/4480/4580 (Industrial)

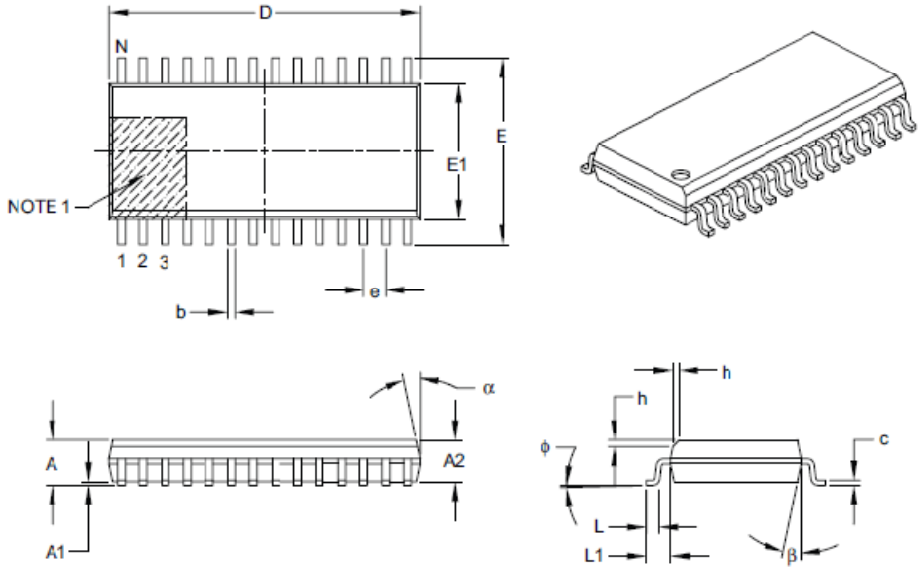
PIC18LF2480/2580/4480/4580 (Industrial)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC18F2480/2580/4480/4580 (Industrial, Extended)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
	VDD	**Supply Voltage**					
D001		PIC18LF2X80/4X80	2.0	—	5.5	V	
		PIC18F2X80/4X80	4.2	—	5.5	V	
D002	VDR	**RAM Data Retention Voltage(1)**	1.5	—	—	V	
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	—	0.7	V	See section on Power-on Reset for details
D004	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	See section on Power-on Reset for details
	VBOR	**Brown-out Reset Voltage**					
D005		PIC18LF2X80/4X80					
		BORV1:BORV0 = 11	2.00	2.05	2.16	V	
		BORV1:BORV0 = 10	2.65	2.79	2.93	V	
D005		All Devices					
		BORV1:BORV0 = 01	4.11	4.33	4.55	V	
		BORV1:BORV0 = 00	4.36	4.59	4.82	V	

Legend: Shading of rows is to assist in readability of the table.
Note 1: This is the limit to which VDD can be lowered in Sleep mode, or during a device Reset, without losing RAM data.

PIC18F2480/2580/4480/4580

28-Lead Plastic Small Outline (SO) – Wide, 7.50 mm Body [SOIC]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		28	
Pitch	e		1.27 BSC	
Overall Height	A	–	–	2.65
Molded Package Thickness	A2	2.05	–	–
Standoff §	A1	0.10	–	0.30
Overall Width	E		10.30 BSC	
Molded Package Width	E1		7.50 BSC	
Overall Length	D		17.90 BSC	
Chamfer (optional)	h	0.25	–	0.75
Foot Length	L	0.40	–	1.27
Footprint	L1		1.40 REF	
Foot Angle Top	φ	0°	–	8°
Lead Thickness	c	0.18	–	0.33
Lead Width	b	0.31	–	0.51
Mold Draft Angle Top	α	5°	–	15°
Mold Draft Angle Bottom	β	5°	–	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

   BSC: Basic Dimension. Theoretically exact value shown without tolerances.

   REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-052B

8.6 Transceiver MCP2551.

MICROCHIP

MCP2551

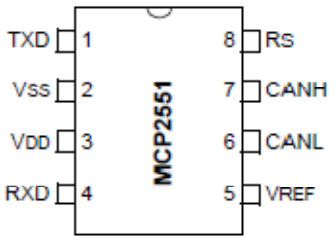
High-Speed CAN Transceiver

Features

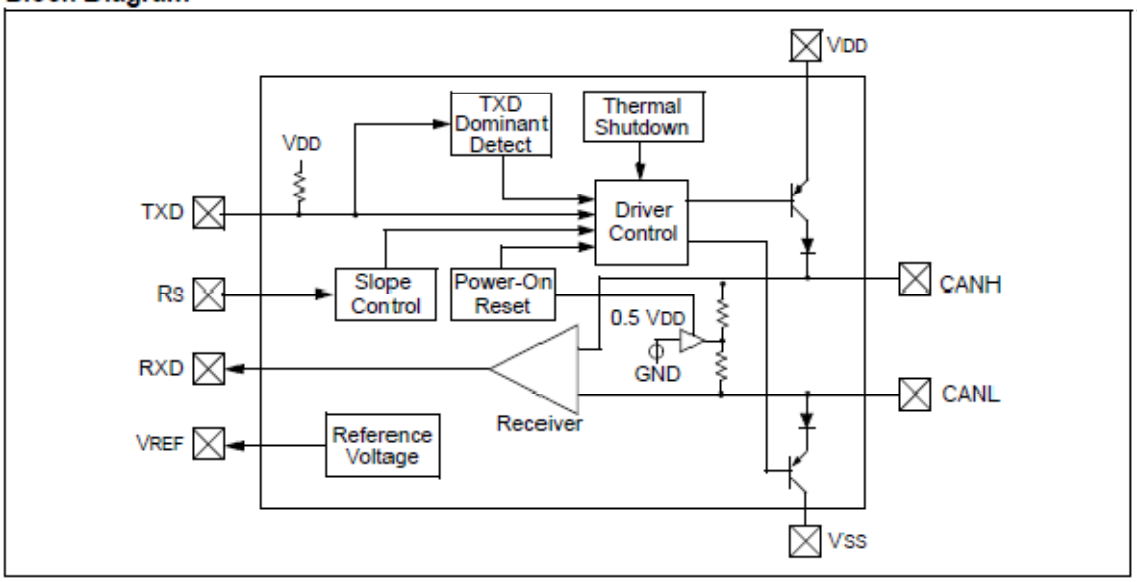
- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent dominant) on TXD input
- Power-on reset and voltage brown-out protection
- An unpowered 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



MCP2551

1.0 DEVICE OVERVIEW

The MCP2551 is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The MCP2551 provides differential transmit and receive capability for the CAN protocol controller and is fully compatible with the ISO-11898 standard, including 24V requirements. It will operate at speeds of up to 1 Mb/s.

Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus cabling (differential output). It also provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources (EMI, ESD, electrical transients, etc.).

1.1 Transmitter Function

The CAN bus has two states: Dominant and Recessive. A dominant state occurs when the differential voltage between CANH and CANL is greater than a defined voltage (e.g.,1.2V). A recessive state occurs when the differential voltage is less than a defined voltage (typically 0V). The dominant and recessive states correspond to the low and high state of the TXD input pin, respectively. However, a dominant state initiated by another CAN node will override a recessive state on the CAN bus.

1.1.1 MAXIMUM NUMBER OF NODES

The MCP2551 CAN outputs will drive a minimum load of 45Ω, allowing a maximum of 112 nodes to be connected (given a minimum differential input resistance of 20 kΩ and a nominal termination resistor value of 120Ω).

1.2 Receiver Function

The RXD output pin reflects the differential bus voltage between CANH and CANL. The low and high states of the RXD output pin correspond to the dominant and recessive states of the CAN bus, respectively.

1.3 Internal Protection

CANH and CANL are protected against battery short-circuits and electrical transients that can occur on the CAN bus. This feature prevents destruction of the transmitter output stage during such a fault condition.

The device is further protected from excessive current loading by thermal shutdown circuitry that disables the output drivers when the junction temperature exceeds a nominal limit of 165°C. All other parts of the chip remain operational and the chip temperature is lowered due to the decreased power dissipation in the transmitter outputs. This protection is essential to protect against bus line short-circuit-induced damage.

1.4 Operating Modes

The RS pin allows three modes of operation to be selected:

- High-Speed
- Slope-Control
- Standby

These modes are summarized in Table 1-1.

When in High-speed or Slope-control mode, the drivers for the CANH and CANL signals are internally regulated to provide controlled symmetry in order to minimize EMI emissions.

Additionally, the slope of the signal transitions on CANH and CANL can be controlled with a resistor connected from pin 8 (RS) to ground, with the slope proportional to the current output at RS, further reducing EMI emissions.

1.4.1 HIGH-SPEED

High-speed mode is selected by connecting the RS pin to VSS. In this mode, the transmitter output drivers have fast output rise and fall times to support high-speed CAN bus rates.

1.4.2 SLOPE-CONTROL

Slope-control mode further reduces EMI by limiting the rise and fall times of CANH and CANL. The slope, or slew rate (SR), is controlled by connecting an external resistor (REXT) between RS and VOL (usually ground). The slope is proportional to the current output at the RS pin. Since the current is primarily determined by the slope-control resistance value REXT, a certain slew rate is achieved by applying a respective resistance. Figure 1-1 illustrates typical slew rate values as a function of the slope-control resistance value.

1.4.3 STANDBY MODE

The device may be placed in standby or "SLEEP" mode by applying a high-level to RS. In SLEEP mode, the transmitter is switched off and the receiver operates at a lower current. The receive pin on the controller side (RXD) is still functional but will operate at a slower rate. The attached microcontroller can monitor RXD for CAN bus activity and place the transceiver into normal operation via the RS pin (at higher bus rates, the first CAN message may be lost).

MCP2551

TABLE 1-1: MODES OF OPERATION

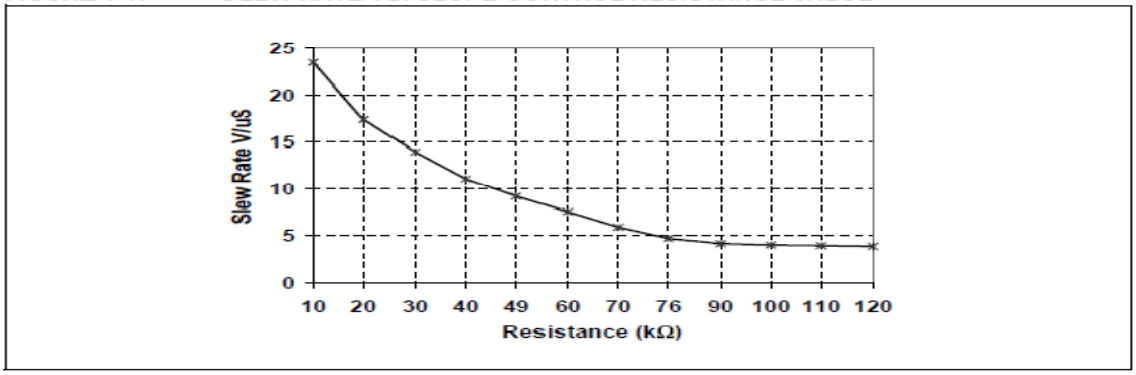
Mode	Current at R_S Pin	Resulting Voltage at Rs Pin
Standby	-IRS < 10 µA	VRS > 0.75 VDD
Slope-control	10 µA < -IRS < 200 µA	0.4 VDD < VRS < 0.6 VDD
High-speed	-IRS < 610 µA	0 < VRS < 0.3VDD

TABLE 1-2: TRANSCEIVER TRUTH TABLE

VDD	VRS	TXD	CANH	CANL	Bus State(1)	RXD(1)
4.5V ≤ VDD ≤ 5.5V	VRS < 0.75 VDD	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	VRS > 0.75 VDD	X	Not Driven	Not Driven	Recessive	1
VPOR < VDD < 4.5V (See Note 3)	VRS < 0.75 VDD	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	VRS > 0.75 VDD	X	Not Driven	Not Driven	Recessive	1
0 < VDD < VPOR	X	X	Not Driven/ No Load	Not Driven/ No Load	High Impedance	X

Note 1: If another bus node is transmitting a dominant bit on the CAN bus, then RXD is a logic '0'.
2: X = "don't care".
3: Device drivers will function, although outputs are not ensured to meet the ISO-11898 specification.

FIGURE 1-1: SLEW RATE VS. SLOPE-CONTROL RESISTANCE VALUE



1.5 TXD Permanent Dominant Detection

If the MCP2551 detects an extended low state on the TXD input, it will disable the CANH and CANL output drivers in order to prevent the corruption of data on the CAN bus. The drivers are disabled if TXD is low for more than 1.25 ms (minimum). This implies a maximum bit time of 62.5 µs (16 kb/s bus rate), allowing up to 20 consecutive transmitted dominant bits during a multiple bit error and error frame scenario. The drivers remain disabled as long as TXD remains low. A rising edge on TXD will reset the timer logic and enable the CANH and CANL output drivers.

1.6 Power-on Reset

When the device is powered on, CANH and CANL remain in a high-impedance state until VDD reaches the voltage-level VPORH. In addition, CANH and CANL will remain in a high-impedance state if TXD is low when VDD reaches VPORH. CANH and CANL will become active only after TXD is asserted high. Once powered on, CANH and CANL will enter a high-impedance state if the voltage level at VDD falls below VPORL, providing voltage brown-out protection during normal operation.

1.7 Pin Descriptions

The 8-pin pinout is listed in Table 1-3.

TABLE 1-3: MCP2551 PINOUT

Pin Number	Pin Name	Pin Function
1	TXD	Transmit Data Input
2	VSS	Ground
3	VDD	Supply Voltage
4	RXD	Receive Data Output
5	VREF	Reference Output Voltage
6	CANL	CAN Low-Level Voltage I/O
7	CANH	CAN High-Level Voltage I/O
8	RS	Slope-Control Input

1.7.1 TRANSMITTER DATA INPUT (TXD)

TXD is a TTL-compatible input pin. The data on this pin is driven out on the CANH and CANL differential output pins. It is usually connected to the transmitter data output of the CAN controller device. When TXD is low, CANH and CANL are in the dominant state. When TXD is high, CANH and CANL are in the recessive state, provided that another CAN node is not driving the CAN bus with a dominant state. TXD has an internal pull-up resistor (nominal 25 kΩ to VDD).

1.7.2 GROUND SUPPLY (VSS)

Ground supply pin.

1.7.3 SUPPLY VOLTAGE (VDD)

Positive supply voltage pin.

1.7.4 RECEIVER DATA OUTPUT (RXD)

RXD is a CMOS-compatible output that drives high or low depending on the differential signals on the CANH and CANL pins and is usually connected to the receiver data input of the CAN controller device. RXD is high when the CAN bus is recessive and low in the dominant state.

1.7.5 REFERENCE VOLTAGE (VREF)

Reference Voltage Output (Defined as VDD/2).

1.7.6 CAN LOW (CANL)

The CANL output drives the low side of the CAN differential bus. This pin is also tied internally to the receive input comparator.

1.7.7 CAN HIGH (CANH)

The CANH output drives the high-side of the CAN differential bus. This pin is also tied internally to the receive input comparator.

1.7.8 SLOPE RESISTOR INPUT (RS)

The RS pin is used to select High-speed, Slope-control or Standby modes via an external biasing resistor.

2.0 ELECTRICAL CHARACTERISTICS

2.1 Terms and Definitions

A number of terms are defined in ISO-11898 that are used to describe the electrical characteristics of a CAN transceiver device. These terms and definitions are summarized in this section.

2.1.1 BUS VOLTAGE

V_{CANL} and V_{CANH} denote the voltages of the bus line wires CANL and CANH relative to ground of each individual CAN node.

2.1.2 COMMON MODE BUS VOLTAGE RANGE

Boundary voltage levels of V_{CANL} and V_{CANH} with respect to ground, for which proper operation will occur, if up to the maximum number of CAN nodes are connected to the bus.

2.1.3 DIFFERENTIAL INTERNAL CAPACITANCE, C_{DIFF} (OF A CAN NODE)

Capacitance seen between CANL and CANH during the recessive state when the CAN node is disconnected from the bus (see Figure 2-1).

2.1.4 DIFFERENTIAL INTERNAL RESISTANCE, R_{DIFF} (OF A CAN NODE)

Resistance seen between CANL and CANH during the recessive state when the CAN node is disconnected from the bus (see Figure 2-1).

2.1.5 DIFFERENTIAL VOLTAGE, V_{DIFF} (OF CAN BUS)

Differential voltage of the two-wire CAN bus, value $V_{DIFF} = V_{CANH} - V_{CANL}$.

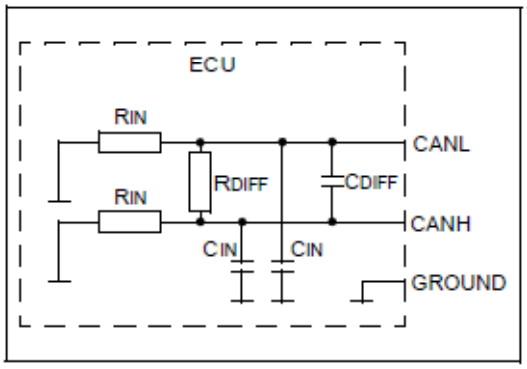
2.1.6 INTERNAL CAPACITANCE, C_{IN} (OF A CAN NODE)

Capacitance seen between CANL (or CANH) and ground during the recessive state when the CAN node is disconnected from the bus (see Figure 2-1).

2.1.7 INTERNAL RESISTANCE, R_{IN} (OF A CAN NODE)

Resistance seen between CANL (or CANH) and ground during the recessive state when the CAN node is disconnected from the bus (see Figure 2-1).

FIGURE 2-1: PHYSICAL LAYER DEFINITIONS



Absolute Maximum Ratings†

V_{DD}	7.0V
DC Voltage at TXD, RXD, V_{REF} and V_S	-0.3V to V_{DD} + 0.3V
DC Voltage at CANH, CANL (**Note 1**)	-42V to +42V
Transient Voltage on Pins 6 and 7 (**Note 2**)	-250V to +250V
Storage temperature	-55°C to +150°C
Operating ambient temperature	-40°C to +125°C
Virtual Junction Temperature, T_{VJ} (**Note 3**)	-40°C to +150°C
Soldering temperature of leads (10 seconds)	+300°C
ESD protection on CANH and CANL pins (**Note 4**)	6 kV
ESD protection on all other pins (**Note 4**)	4 kV

Note 1: Short-circuit applied when TXD is high and low.

2: In accordance with ISO-7637.

3: In accordance with IEC 60747-1.

4: Classification A: Human Body Model.

† NOTICE: Stresses above those listed under "Maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

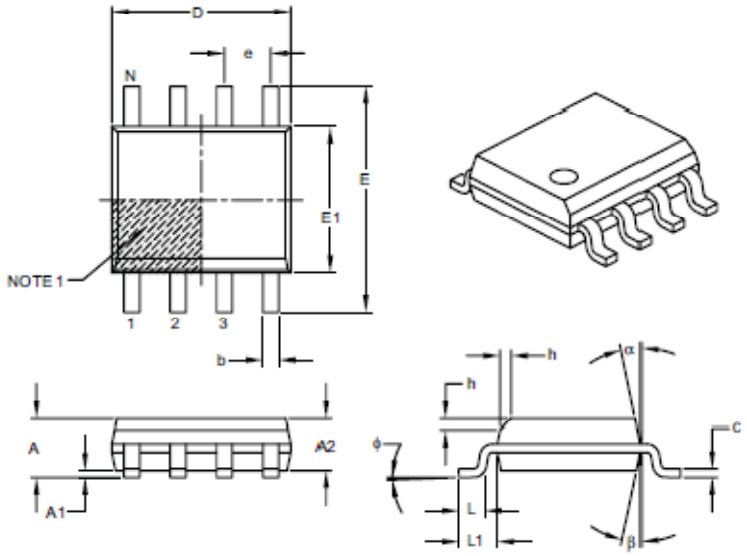
2.2 DC Characteristics

DC Specifications			Electrical Characteristics: Industrial (I): TAMB = -40°C to +85°C VDD = 4.5V to 5.5V Extended (E): TAMB = -40°C to +125°C VDD = 4.5V to 5.5V			
Param No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
Supply						
D1	IDD	Supply Current	—	75	mA	Dominant; VTXD = 0.8V; VDD
D2			—	10	mA	Recessive; VTXD = +2V; RS = 47 kΩ
D3			—	365	µA	-40°C ≤ T_{AMB} ≤ +85°C, Standby; **(Note 2)**
			—	465	µA	-40°C ≤ T_{AMB} ≤ +125°C, Standby; **(Note 2)**
D4	VPORH	High-level of the power-on reset comparator	3.8	4.3	V	CANH, CANL outputs are active when VDD > VPORH
D5	VPORL	Low-level of the power-on reset comparator	3.4	4.0	V	CANH, CANL outputs are not active when VDD < VPORL
D6	VPORD	Hysteresis of power-on reset comparator	0.3	0.8	V	**Note 1**
Bus Line (CANH; CANL) Transmitter						
D7	$V_{CANH(r)}$, $V_{CANL(r)}$	CANH, CANL Recessive bus voltage	2.0	3.0	V	VTXD = VDD; no load.
D8	IO(CANH)(reces) IO(CANL)(reces)	Recessive output current	-2	+2	mA	-2V < V(CAHL,CANH) < +7V, 0V <VDD < 5.5V
D9			-10	+10	mA	-5V < V(CANL,CANH) < +40V, 0V <VDD < 5.5V
D10	VO(CANH)	CANH dominant output voltage	2.75	4.5	V	VTXD = 0.8V
D11	VO(CANL)	CANL dominant output voltage	0.5	2.25	V	VTXD = 0.8V
D12	VDIFF(r)(o)	Recessive differential output voltage	-500	+50	mV	VTXD = 2V; no load
D13	VDIFF(d)(o)	Dominant differential output voltage	1.5	3.0	V	VTXD = 0.8V; VDD = 5V 40Ω < RL < 60Ω **(Note 2)**
D14	IO(SC)(CANH)	CANH short-circuit output current	—	-200	mA	VCANH = -5V
D15			—	-100 (typical)	mA	VCANH = -40V, +40V. **(Note 1)**
D16	IO(SC)(CANL)I	CANL short-circuit output current	—	200	mA	VCANL = -40V, +40V. **(Note 1)**
Bus Line (CANH; CANL) Receiver: [TXD = 2V; pins 6 and 7 externally driven]						
D17	VDIFF(r)(i)	Recessive differential input voltage	-1.0	+0.5	V	-2V < V(CANL, CANH) < +7V **(Note 3)**
			-1.0	+0.4	V	-12V < V(CANL, CANH) < +12V **(Note 3)**
D18	VDIFF(d)(i)	Dominant differential input voltage	0.9	5.0	V	-2V < V(CANL, CANH) < +7V **(Note 3)**
			1.0	5.0	V	-12V < V(CANL, CANH) < +12V **(Note 3)**
D19	VDIFF(h)(i)	Differential input hysteresis	100	200	mV	see **Figure 2-3**. **(Note 1)**
D20	RIN	CANH, CANL common-mode input resistance	5	50	kΩ	
D21	RIN(d)	Deviation between CANH and CANL common-mode input resistance	-3	+3	%	VCANH = VCANL

Note 1: This parameter is periodically sampled and not 100% tested.
2: ITXD = IRXD = IVREF = 0 mA; 0V < VCANL < VDD; 0V < VCANH < VDD; VRS = VDD.
3: This is valid for the receiver in all modes; High-speed, Slope-control and Standby.

8-Lead Plastic Small Outline (SN) – Narrow, 3.90 mm Body [SOIC]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		MILLMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		1.27 BSC	
Overall Height	A	–	–	1.75
Molded Package Thickness	A2	1.25	–	–
Standoff §	A1	0.10	–	0.25
Overall Width	E		6.00 BSC	
Molded Package Width	E1		3.90 BSC	
Overall Length	D		4.90 BSC	
Chamfer (optional)	h	0.25	–	0.50
Foot Length	L	0.40	–	1.27
Footprint	L1		1.04 REF	
Foot Angle	φ	0°	–	8°
Lead Thickness	c	0.17	–	0.25
Lead Width	b	0.31	–	0.51
Mold Draft Angle Top	α	5°	–	15°
Mold Draft Angle Bottom	β	5°	–	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-057B